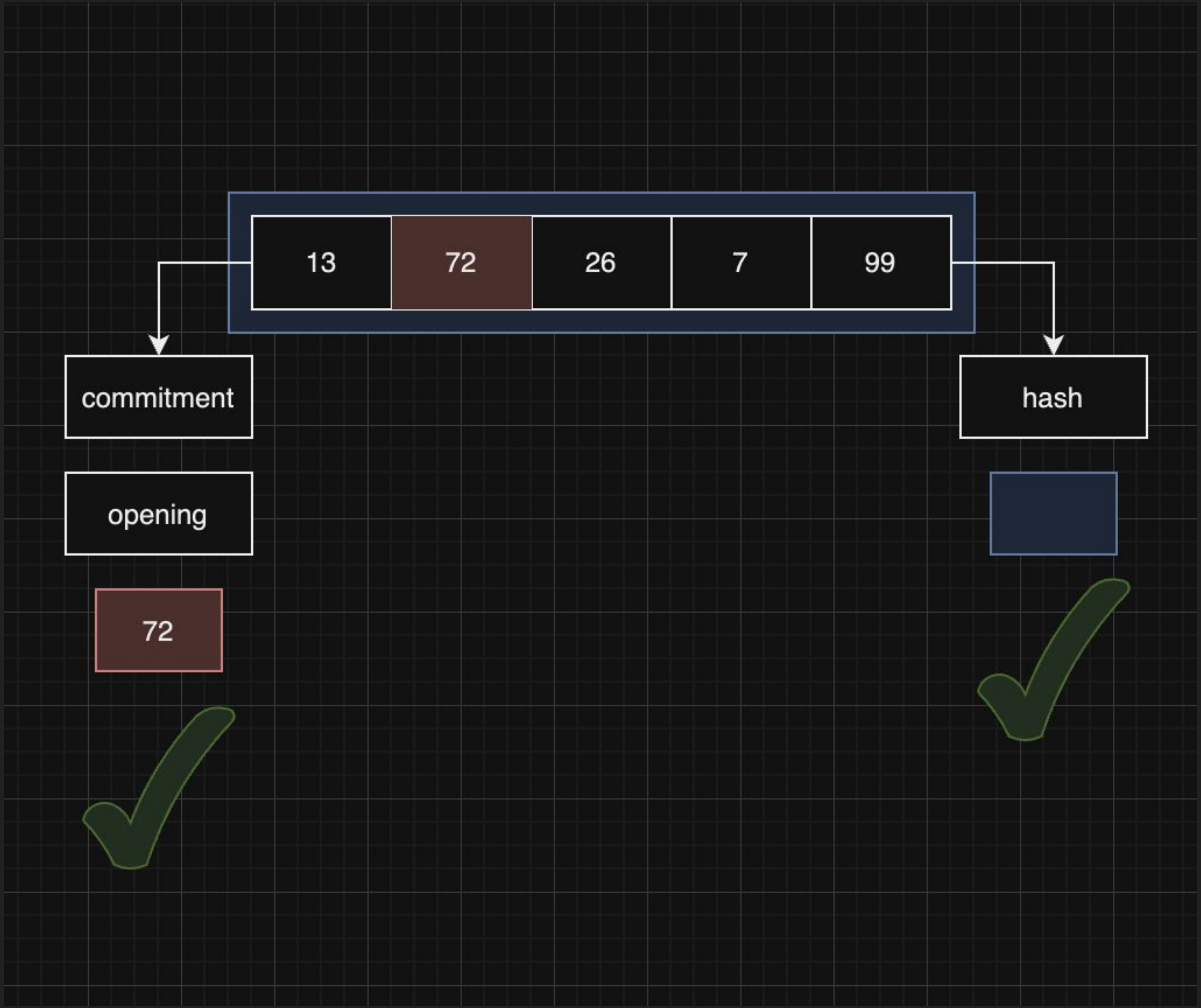
Verkle Tries, Statelessness and The Verge

Agnish Ghosh,
Engineer at TrueZK,
Protocol Fellow at Ethereum Foundation
Ethereum Devconnect Scholarship Recipient

Vector Commitments vs Hash Functions

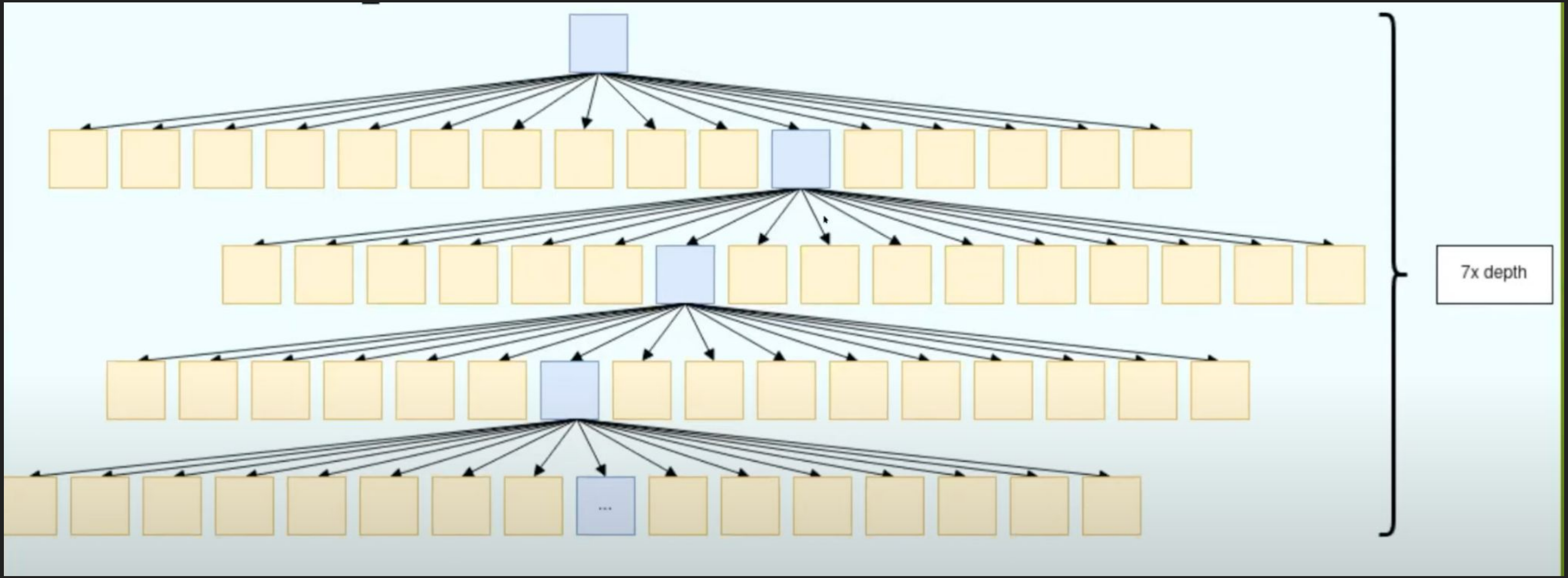
- Verifying data via Hash functions results in revealing the whole data.
- However, if I use Vector Commitments, I can simply prove that the 2nd index of this vector is indeed 72, verification happens by making an “opening” at that point, eventually verifying that point.
- Vector Commitments are inherently based on the concept of Polynomial Commitment Schemes, just that here we represent each of entries of the vector as a linear combination of coefficients of an agreed upon polynomial.



13
72
26
7
99
commitment
opening
72
hash

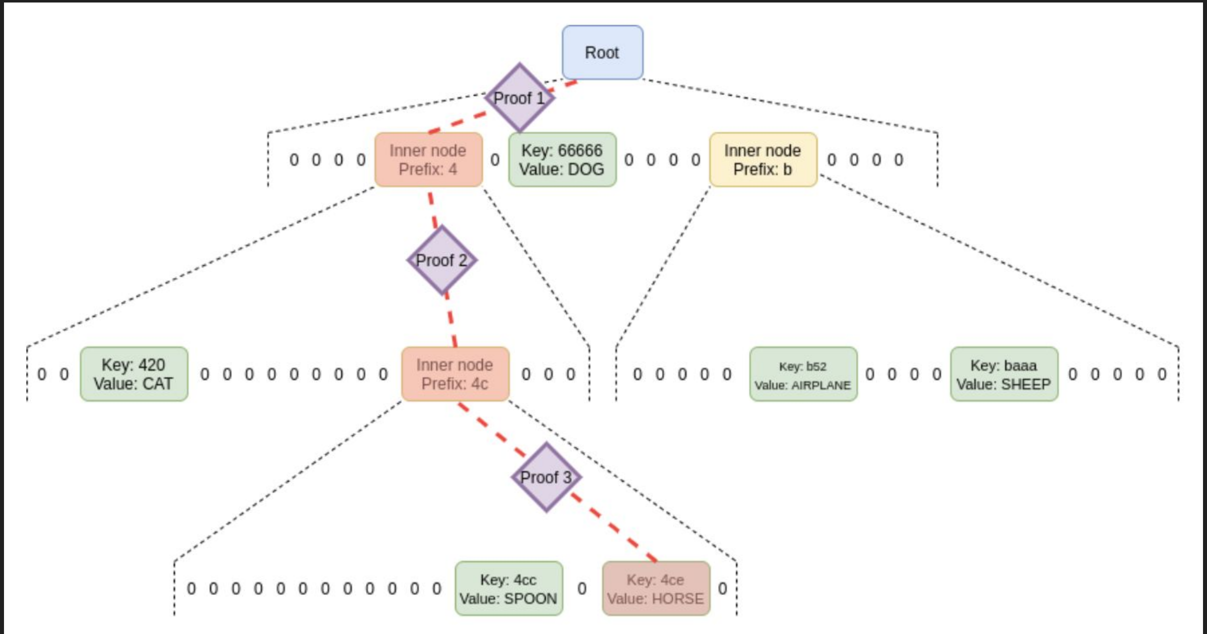
Downsides of using the Hash function

Merkle Proofs using Hash functions are including data of **all** the sister nodes at each level



Why vector commitments?

No sister nodes are needed in the proof, the proof generation takes a more "path-specific" approach as **they link each commitment in the path to the next.**



Advantages?

Good enough….

Trie	Specifics in each proof	Levels	Proof Size	Cryptographic Reqs
Merkle Patricia Trie	leaf data + 15 siblings, 32 bytes for each level	~7	~3.5 MB for 1,000 leaves	Collision-resistant hash functions
Verkle Trie	leaf data + *Polynomial Comm* + value + `index of child, which is 32 bytes`, another 32 bytes + 1 byte + *small constant size data*	~4	~150 KB for 1,000 leaves	**Earlier (for Eth1 State)**: KZG Polynomial Comms (based on Billinear Pairings) with BLS12_381 curve, **Now (for Eth2)**: Pedersen Commitments with Bandersnatch/Banderwagon

Verkle Tries + the Verkle Cryptography API

Initial Mathematical Approach (for Eth1)

- Using Multipoint proofs based on KZG Polynomial Commitment Schemes, for a Verkle Trie of depth say ‘d’, we could compute a commitment to individual vectors at each level, which is a_0, a_1,...,a_2^d - 1.
- Where the defn of the Polynomial Commitment is p(X) of degree n a function :
- Here p_i are the coefficients of the individual polynomials
- Degree of p(X) is the depth of the tree, i.e, 2^d - 1.
- Commitments are ideally **48 bytes** long, computed on the

  BLS12_381 curve.

$$\sum_{i=0}^{n} p_i X^i$$

Demerits of this approach

- As commitments in KZG were essentially group elements, that were dependent on bilinear pairings
- Pairing based cryptography, in this case, needed a Trusted Setup, which was usually implemented via secure Multi-Party Computation
- A trusted setup ceremony usually takes some time.

- On the other hand, a new approach based on Inner Product Arguments (kind of like Bulletproofs) and Pedersen Commitments, were mainly based on the “Discrete Log Problem”.
- This design did NOT require a Trusted Setup.

A few more things on Pedersen Commitments

- This is a collision resistant hash based commitment scheme, which is used to commit to a value x ∈ Z_p, where p is a prime, hence we're dealing with a prime field here.

  Consider a cryptographic group

$$G = \{1, g, g^2, \dots g^{q-1}\};\ q = |G|,\ \text{assume } q \text{ is prime}$$

$$\text{where } g^i . g^j = g^{(i+j \ mod \ q)}$$

Hold on…

- Now we can define g, h ∊ R = {0,1,....,q-1}, for m, r ∊ R. Then the hash function would simply be

$$H(m,r) = g^m . h^r \in G$$

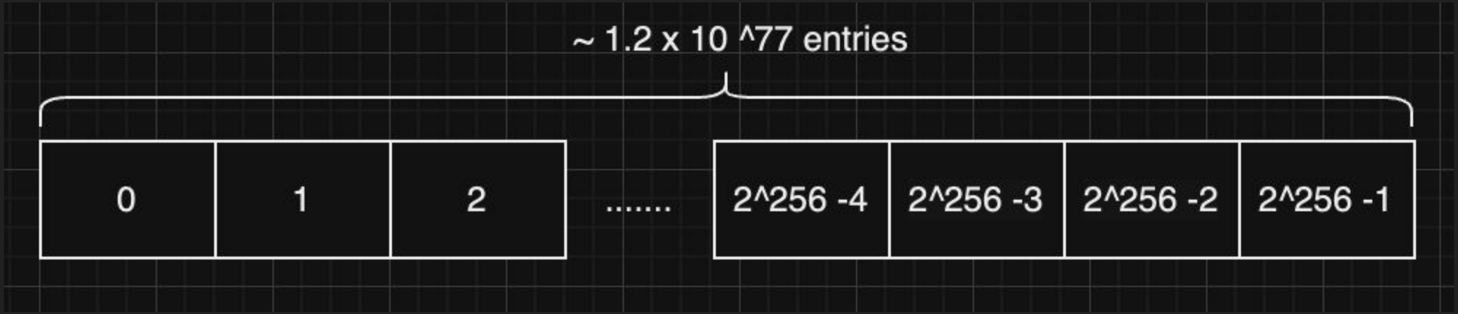
- Moreover, these commitments are **homomorphic** in nature, which means, comm(m1, r1) + comm(m2, r2) = comm (m1+m2, r1+r2).
- These gives the us the **real** leverage to use this in multipoint vector commitments, thereby making verkle proofs mainly path dependent.

Quick Recap

- Verkle uses commitments, so most of the data isn’t revealed.
- Verkle cares about path and not the siblings -> smaller proofs
- Verkle cares about depth, so more children means lower depth, which means more data aggregated into a single proof -> smaller proof
- Account and storage tries are merged into one, hence again smaller proof.

Why can't we use a single commitment?

- Getting a vector commitment for the whole state makes the proof generation very latent.
- We usually think about a trade-off between proof size vs computation time
- Ideally the verkle tries are 4 levels deep and contains 256 children.



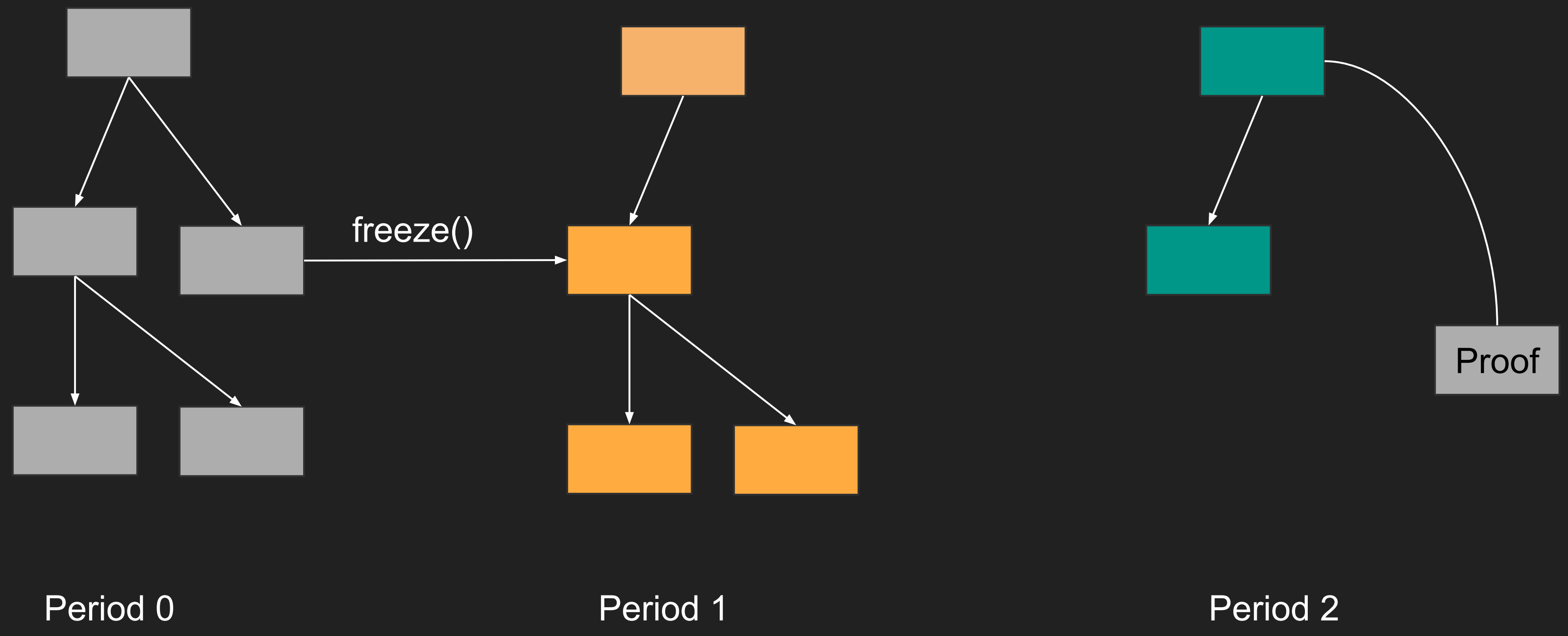
Statelessness why???

1. Reduced amount of data to participate in the network
2. Easy and less time-consuming network syncing
3. Lesser baseline for storage specs
4. Reduced amount of data required to process, which shall be easier for smaller devices
5. Faster access to info

State Expiry

- Every year, the period resets with a new tree
- Data for the current and previous year is stored
- Data for previous years are discarded, except for the state roots
- Accessing data before 2 periods requires **data resurrection,** and a valid proof.

State Expiry -> State Resurrection



Address Space Extension

1. Increase the no. of bytes in an address from 20 to 32
2. Use bytes 3-5 for the period at which the address was first accessed
3. Use bytes 6-31 (26 bytes) to store the hash of the public key
4. Use byte 0 for version, bytes1-2 are reserved (0)

Period	Address	Balance
0	01000000000000000000000000...0000000000000000000000000000	1234
1	01000000000100000000000000....0000000000000000000000000000	9101
2	01000000000200000000000000....0000000000000000000000000042 01000000000000000000000000....00000000000000000000000	1121 1234

Connecting ASE with State Expiry

- Simplifying data storage by subdividing state tries into periods, charging for data storage in exchange of a fee.

- Potential threat: risk of breaking existing contracts, contract bridging by Ipsilon team

State Networks -> The Portal Network (*by Piper*)

- A node stores only a subset of the entire data
- Data is requested over the network as per need
- Proofs provided to ensure that the provided data is correct
- Pros: be a validator in Eth2, without storing any data at all!

Some of the existing research going on in EPF

- Verkle Trie library and migration with the existing Eth1 and Eth2 Nimbus Client, using the Constantine crypto library written in Nim.
- Verkle Trie library for Besu, currently interfacing Arkworks.

Some potential explorations

- Exploring the need for developing a Java cryptographic primitives library
- Possible changes in type 1 (Ethereum Equivalent zkRollups)