Hello, SBC!
I’m Boyma.

(@boymanjor)

Urkel Tree:

An optimized and cryptographically provable key-value store for decentralized naming.

Handshake

Authenticated Data Structures

Bitcoin

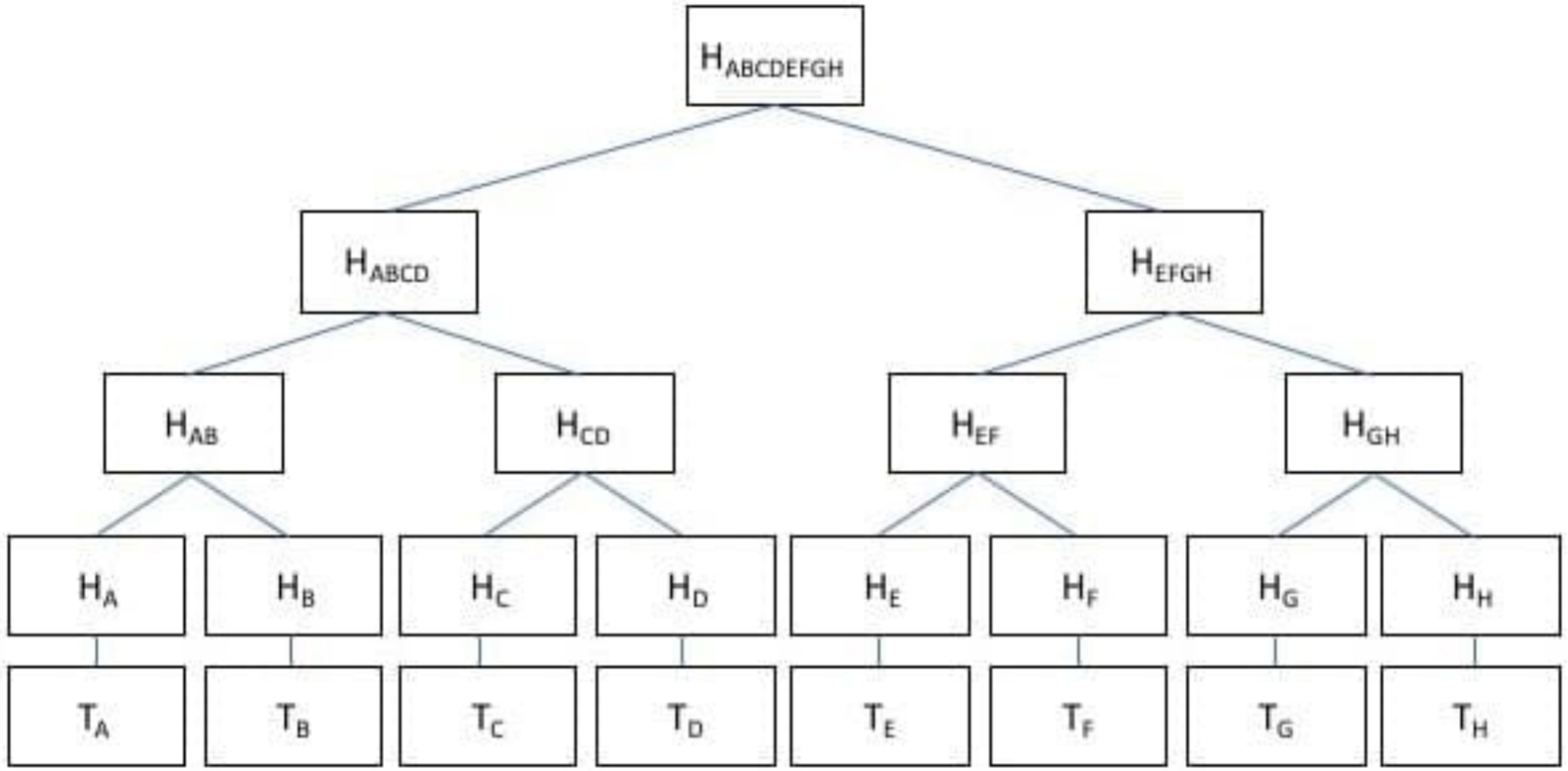
Ethereum

WARNING:

Basic Explanation

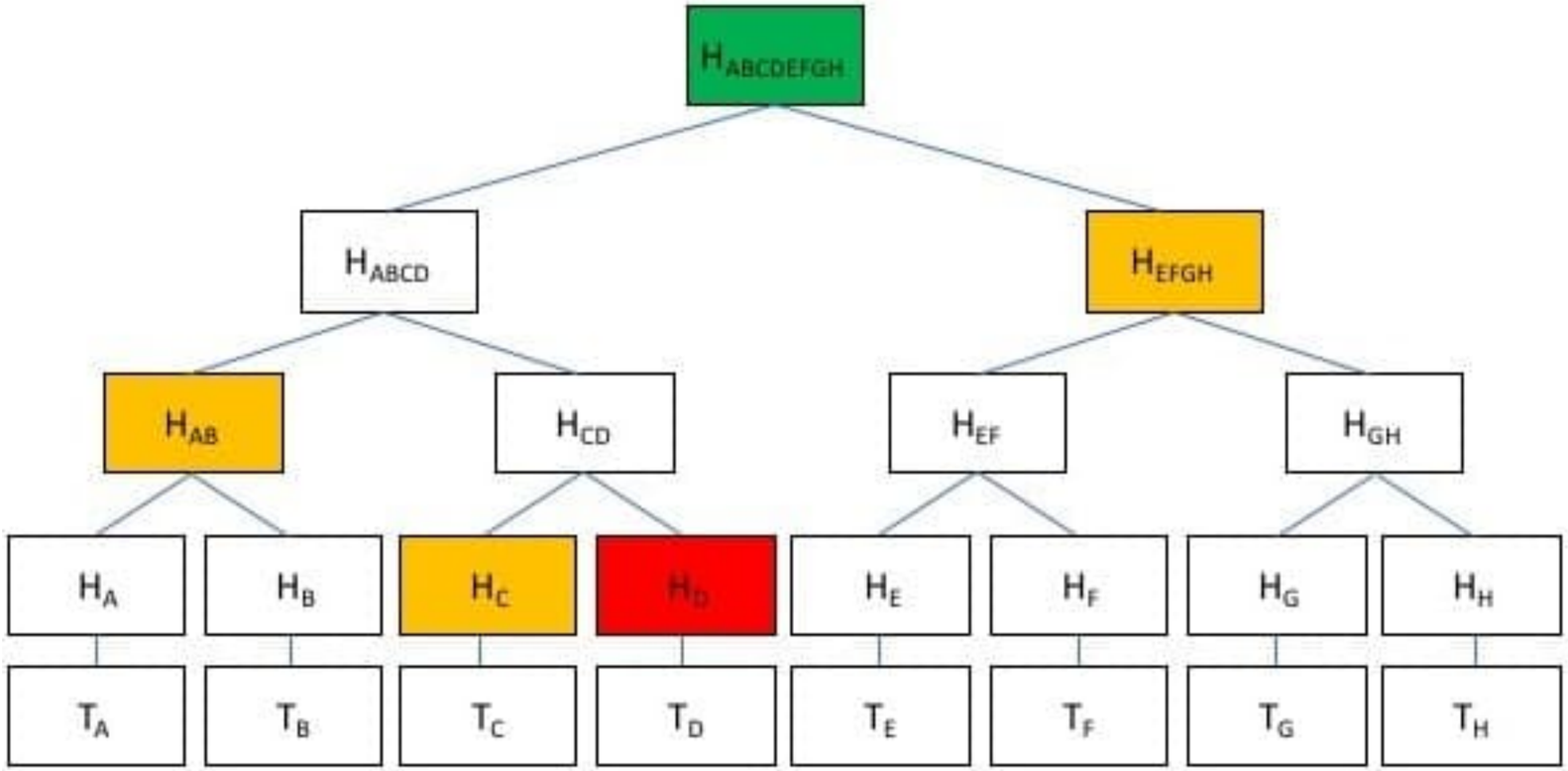
Merkle Tree

Creation



Merkle Tree

Inclusion Proof



What about ~~Bob~~ Ethereum?

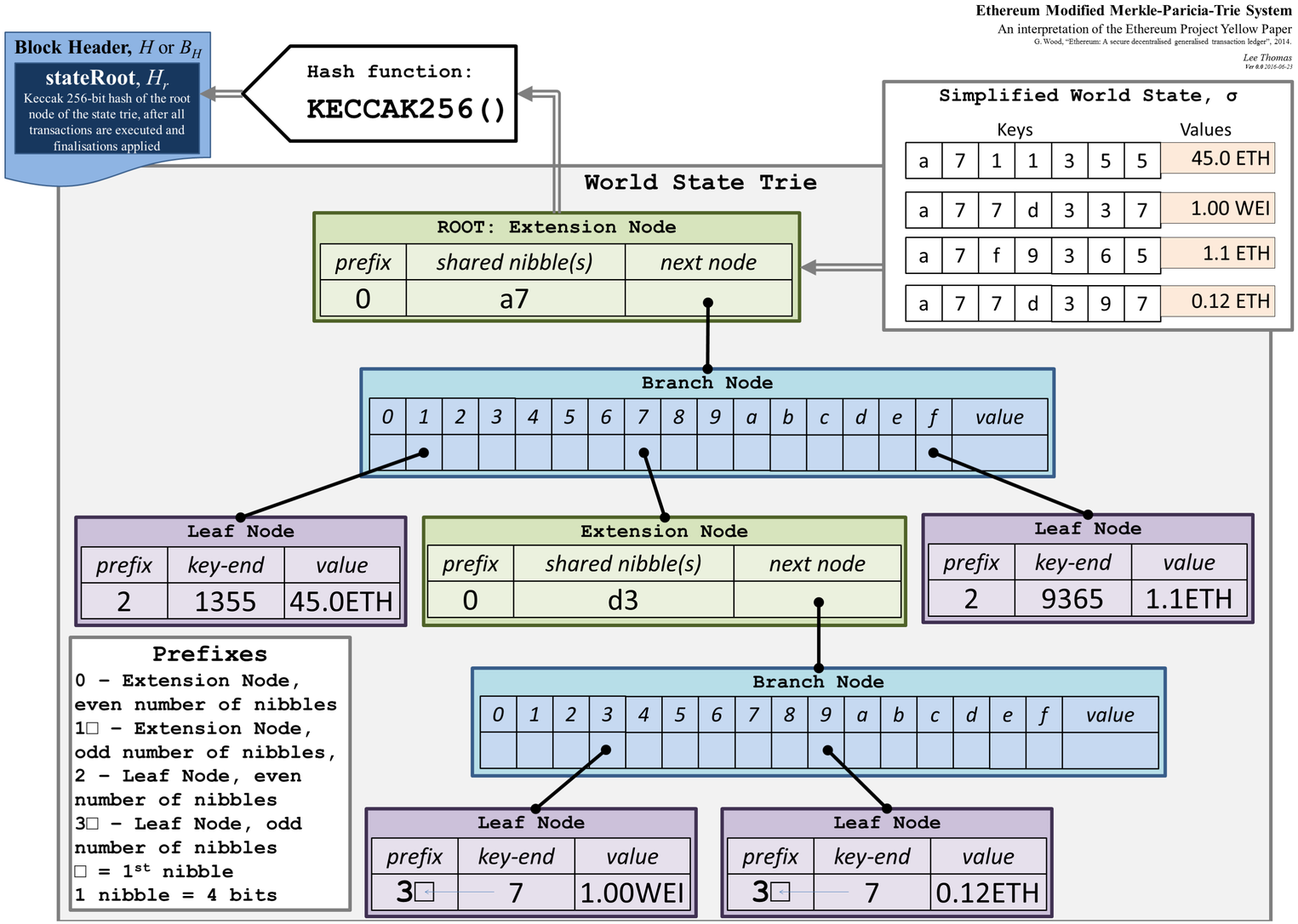
Ethereum Wish List

- key-value store
- allow updates w/o entire tree reconstruction
- bounded depth
- history independence

WARNING:

Another Basic Explanation

Merkle Patricia Tree



What about ~~Bob~~ Handshake?

Handshake Wish List

- key-value store with minimal storage
- exceptional performance on SSDs
- small proof size (< 1kb)
- history independence

The Candidates

- ~~re-balancing data structures~~
- Merkle Patricia Tree
- Sparse Merkle Tree

Merkle Patricia Tree

- Proof sizes too large
- Storage requirements too large

Sparse Merkle Tree

- Too many database lookups
- Too much hashing

The Decision

?

Urkel Tree:

An optimized and cryptographically provable key-value store for decentralized naming.

Urkel Tree

- Base-2 merklelized trie
- Two types of nodes: internal and leaf
- append-only files

Structure on Disk

```
struct internal_node_s {
  uint8_t left_hash[32];
  uint16_t left_file;
  uint32_t left_position;
  uint8_t right_hash[32];
  uint16_t right_file;
  uint32_t right_position;
} internal_node;
```

```
struct leaf_node_s {
  uint8_t key[32];
  uint16_t value_file;
  uint32_t value_position;
  uint16_t value_size;
} leaf_node;
```

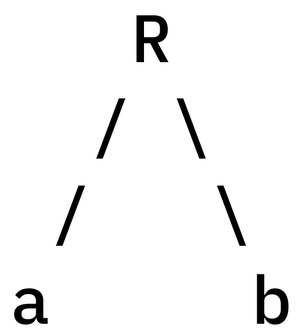
Urkel Tree:

Insertion

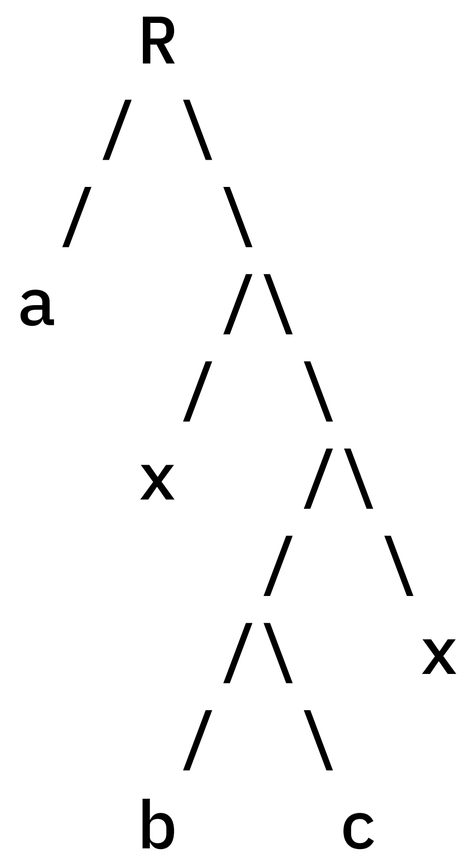
Map:
 0000 = a

a

Map:
 0000 = a
 0000 = b



Map:
 0000 = a
 1100 = b
 1101 = c



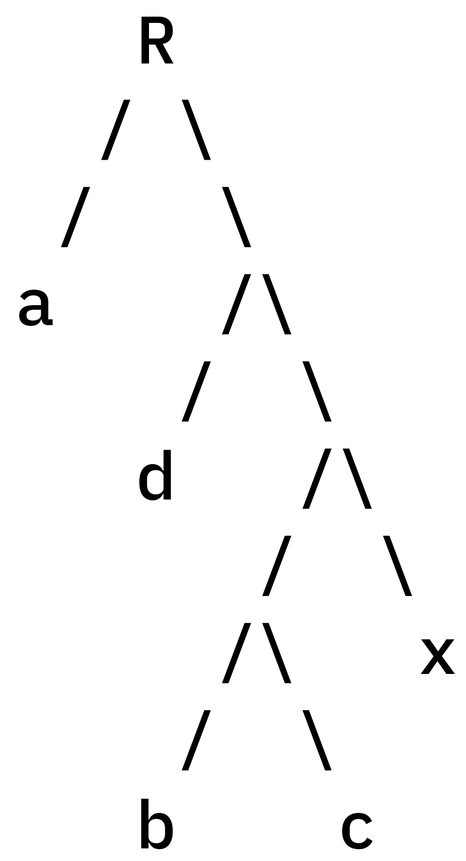
Map:

0000 = a

1100 = b

1101 = c

1000 = d



Map:

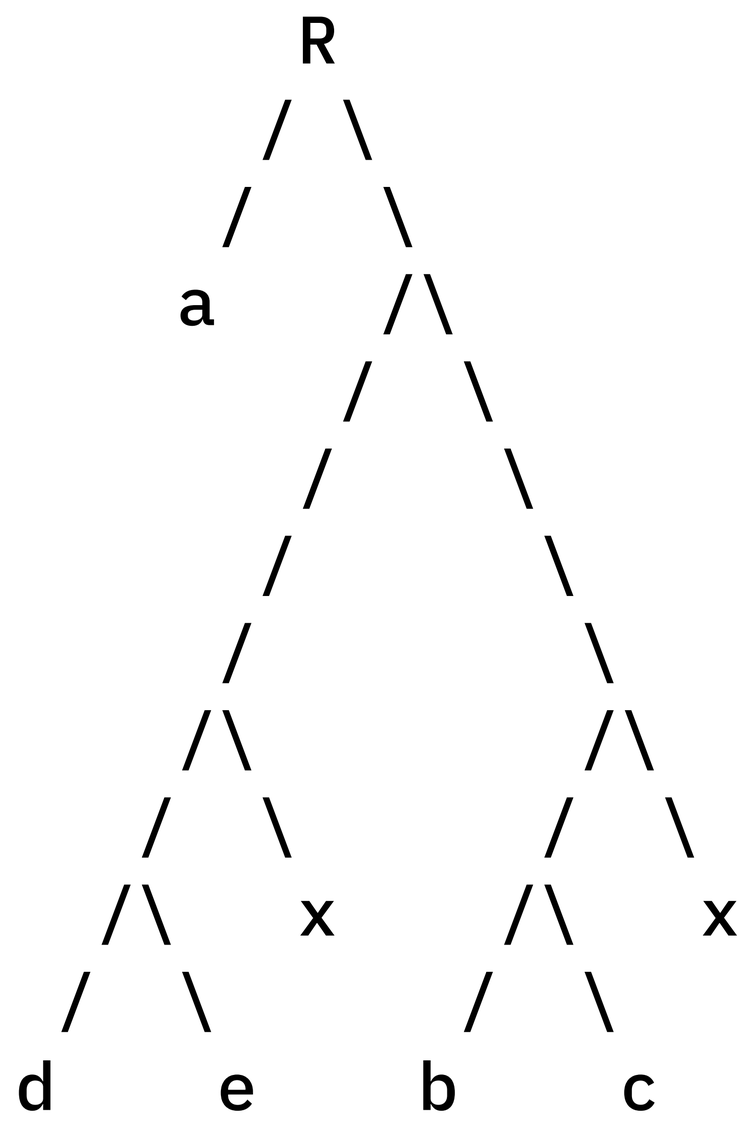
0000 = a

1100 = b

1101 = c

1000 = d

1001 = e



Urkel Tree:

Removal

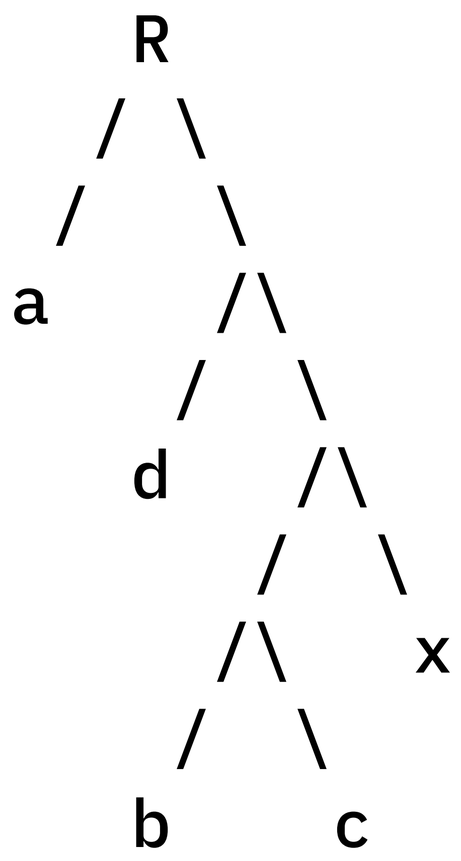
Map:

0000 = a

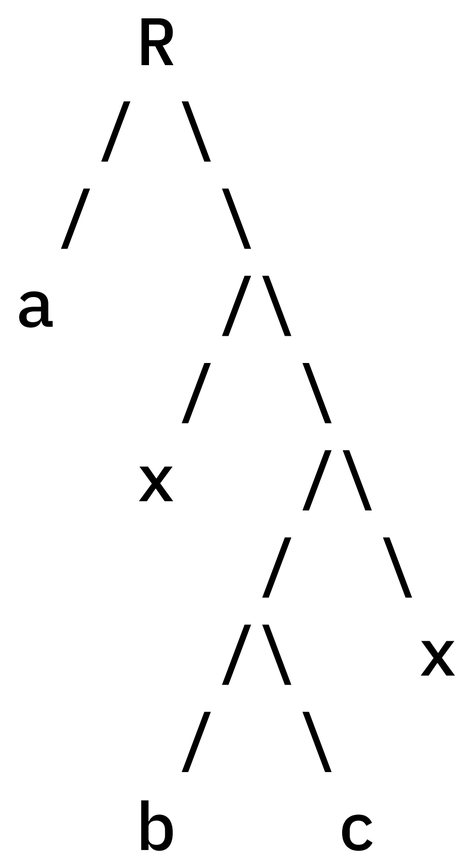
1100 = b

1101 = c

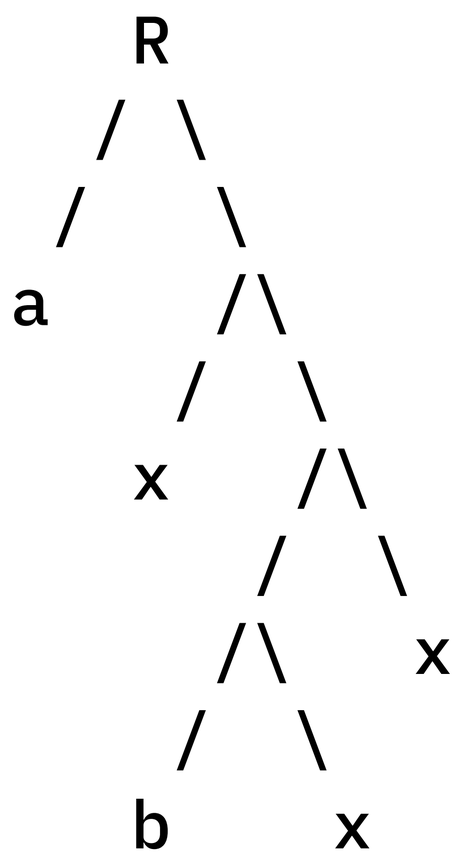
1000 = d



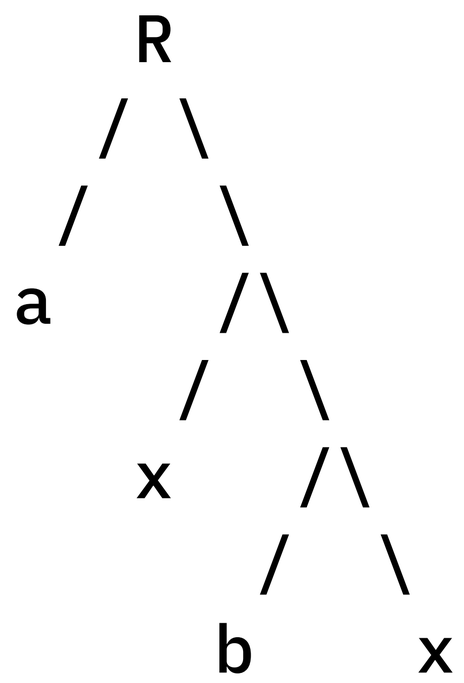
Map:
 0000 = a
 1100 = b
 1101 = c



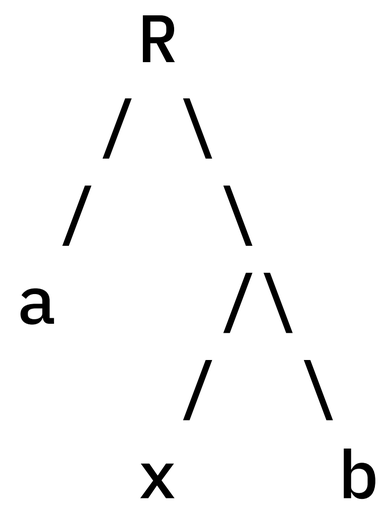
Map:
 0000 = a
 1100 = b



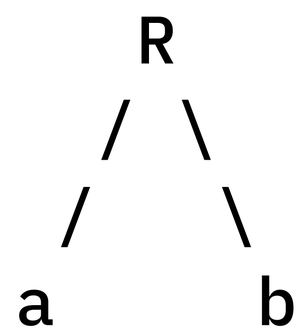
Map:
 0000 = a
 1100 = b



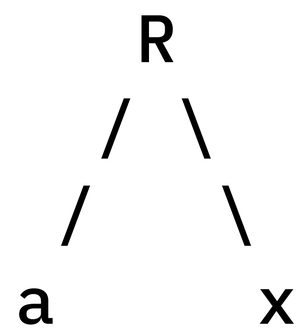
Map:
 0000 = a
 1100 = b



Map:
 0000 = a
 1100 = b



Map:
 0000 = a



Map:
 0000 = a

a

Urkel Tree:

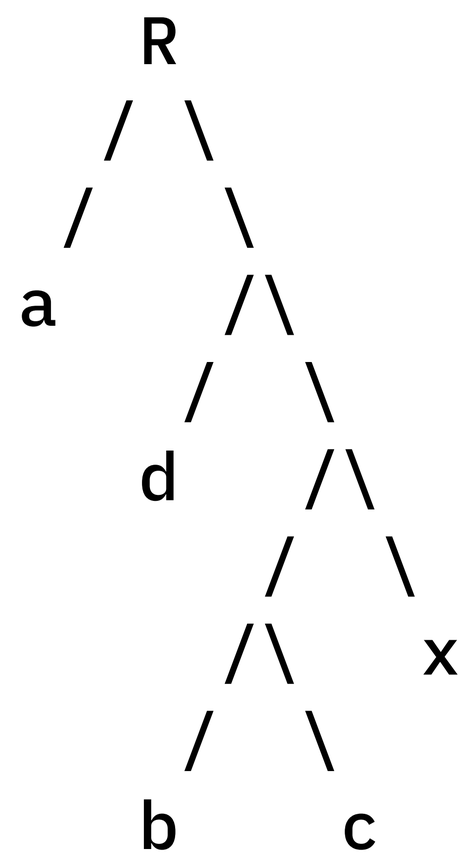
Proofs

Urkel Tree:

Leaf Hash

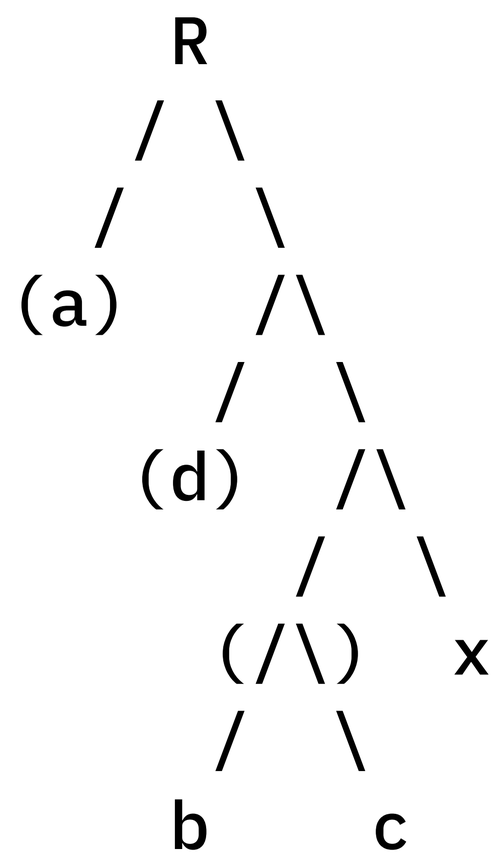
```
HASH(0x00 || 256-bit-key || HASH(value))
```

Map:
 0000 = a
 1100 = b
 1101 = c
 1000 = d



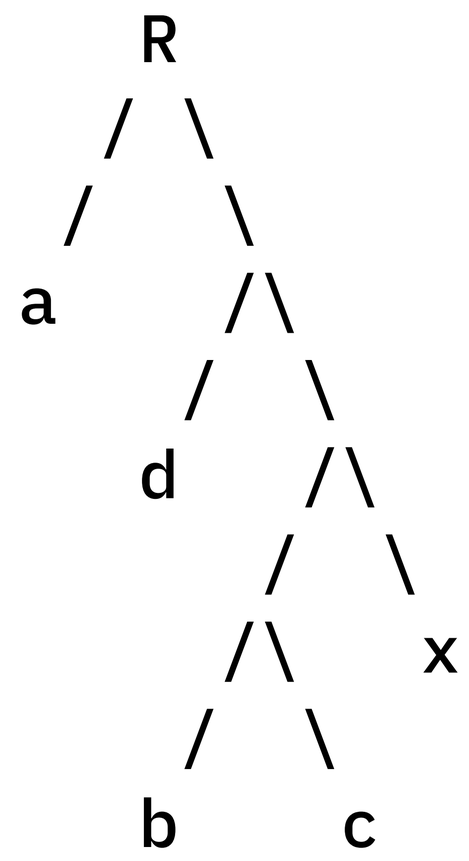
Proof of non-inclusion for 1110.

Map:
 0000 = a
 1100 = b
 1101 = c
 1000 = d



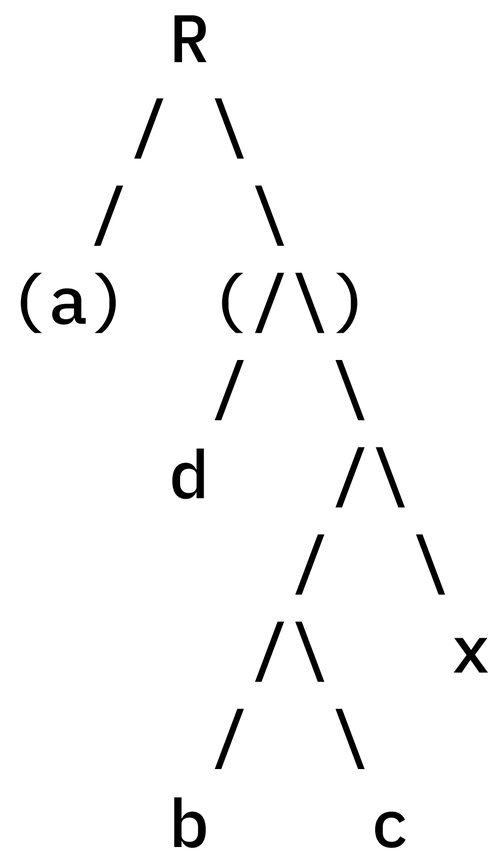
Proof of non-inclusion for 1110.

Map:
 0000 = a
 1100 = b
 1101 = c
 1000 = d



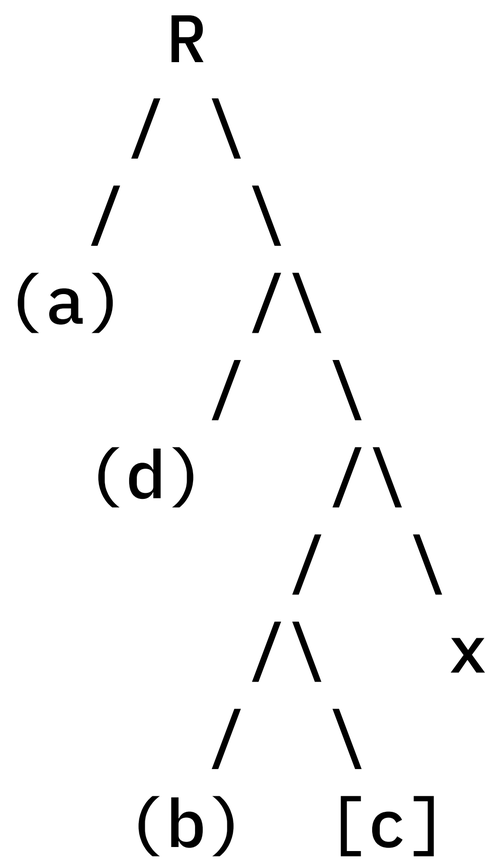
Proof of non-inclusion for 0100.

Map:
 0000 = a
 1100 = b
 1101 = c
 1000 = d



Proof of non-inclusion for 0100.

Map:
 0000 = a
 1100 = b
 1101 = c
 1000 = d



Proof of inclusion for 1101.

Benchmarks

- Intel Core i7-7500U with 2.7GH
- NVMe PCIe SSD
- 50 million 300-byte leaves
- 500 leave batches
- periodic commissions of 44,000 values

Benchmarks (cont.)

- 500 value batches averaged 100-150ms
- 44,0000 value commissions averaged 400-600ms
- average node depth of 27 or 28 bits
- ~800 bytes proof size
- 1-2ms proof creation time

Merklix Variant

- Attackers can grind keys to grow tree
- Bitcoin produces 72-80 bit collisions on block headers
- base-2 merkelized radix tree for DoS protection
- adds complexity to code

Summary

Primary Advantages

- Performance
- Simplicity
- Storage
- Proof size