



Modern key distribution with *ClaimChains*

A decentralized Public Key Infrastructure that supports privacy-friendly social verification

NEXTLEAP

Bogdan Kulynych
Carmela Troncoso

Marios Isaakidis
George Danezis

BLOCKCHAINS



BLOCKCHAINS EVERYWHERE

BLOCKCHAINS



HIGH-INTEGRITY

Tamper proof
Authenticity

BLOCKCHAINS EVERYWHERE

BLOCKCHAINS

HIGH-INTEGRITY

Tamper proof
Authenticity

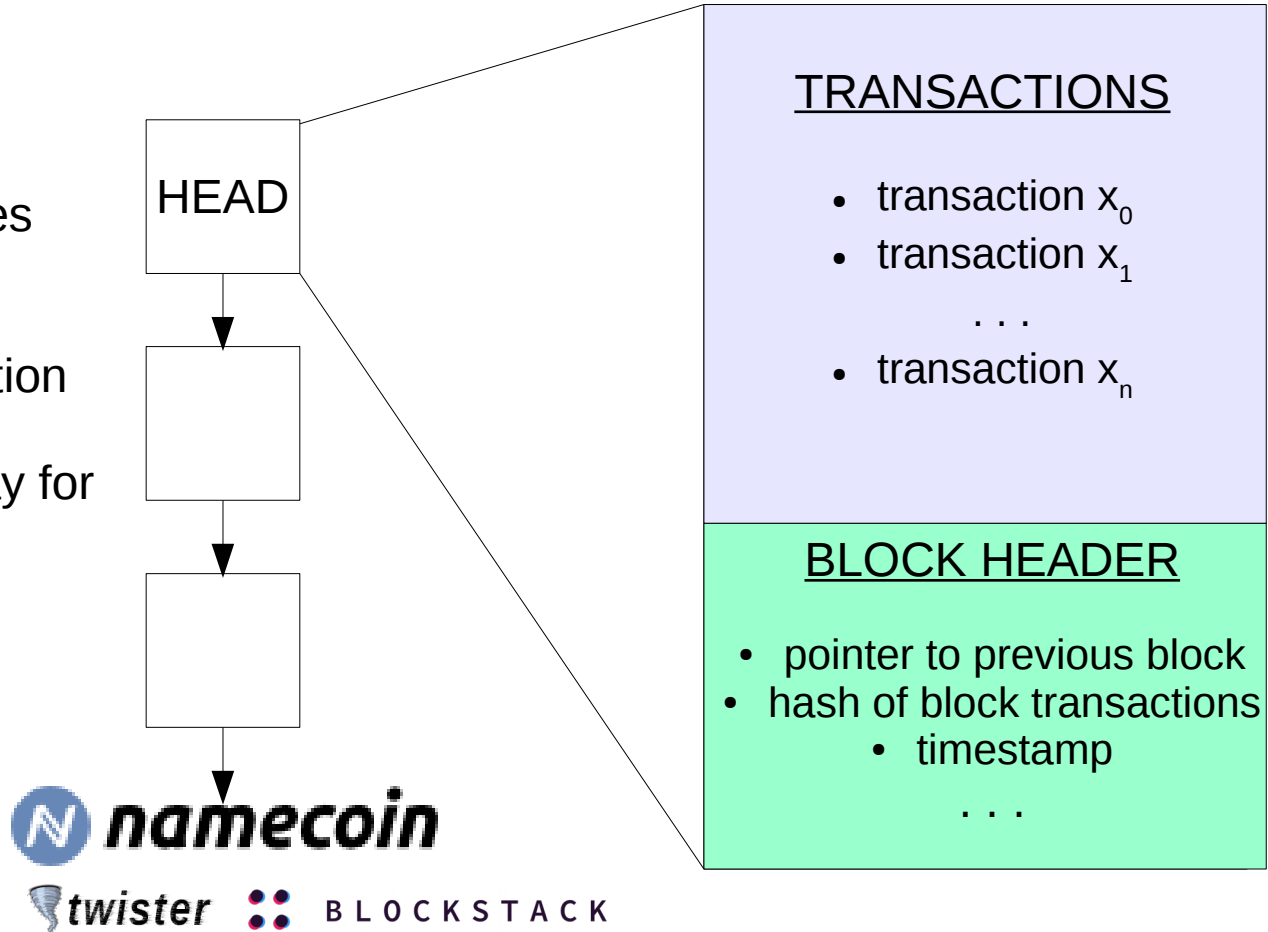
DECENTRALIZATION

Availability
Censorship-resistant
Global consensus

BLOCKCHAINS EVERYWHERE

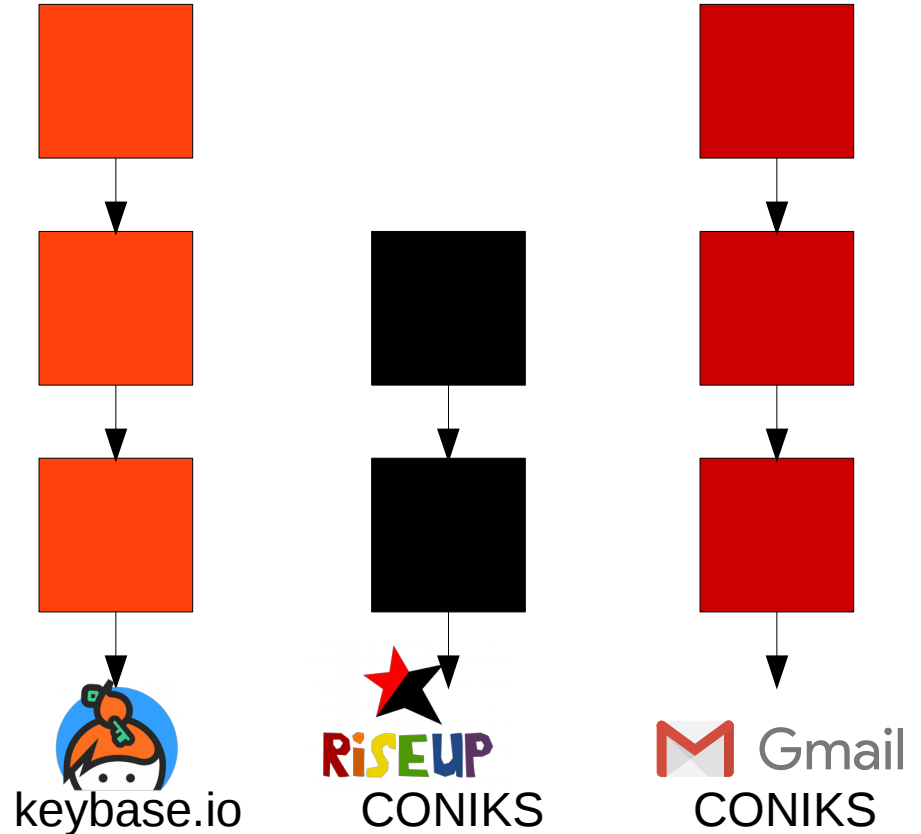
Cryptocurrency chains

- ✓ Powerful abstraction for identities
- ✓ Global namespace
- ✗ No mechanism for social validation
- ✗ All transactions are public
- ✗ Users need to buy coins and pay for transaction fees
- ✗ Resource expensive



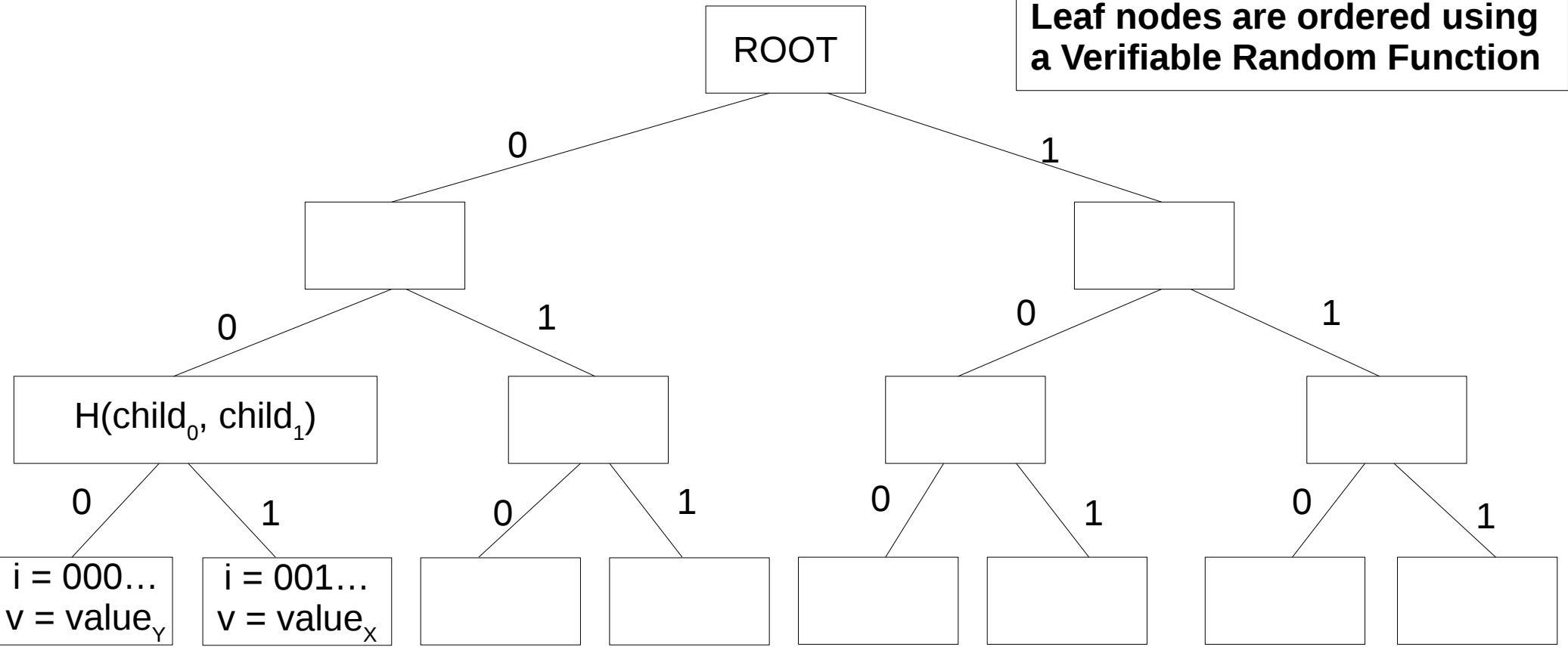
Federated “Merkle prefix tree” chains

- ✓ Accountability
- ✓ Easy discovery
- ✓ Efficient
- ✗ Do not prevent equivocation
- ✗ Centralization
 - Single point of failure
 - Surveillance



Merkle binary prefix trees

Leaf nodes are ordered using a Verifiable Random Function



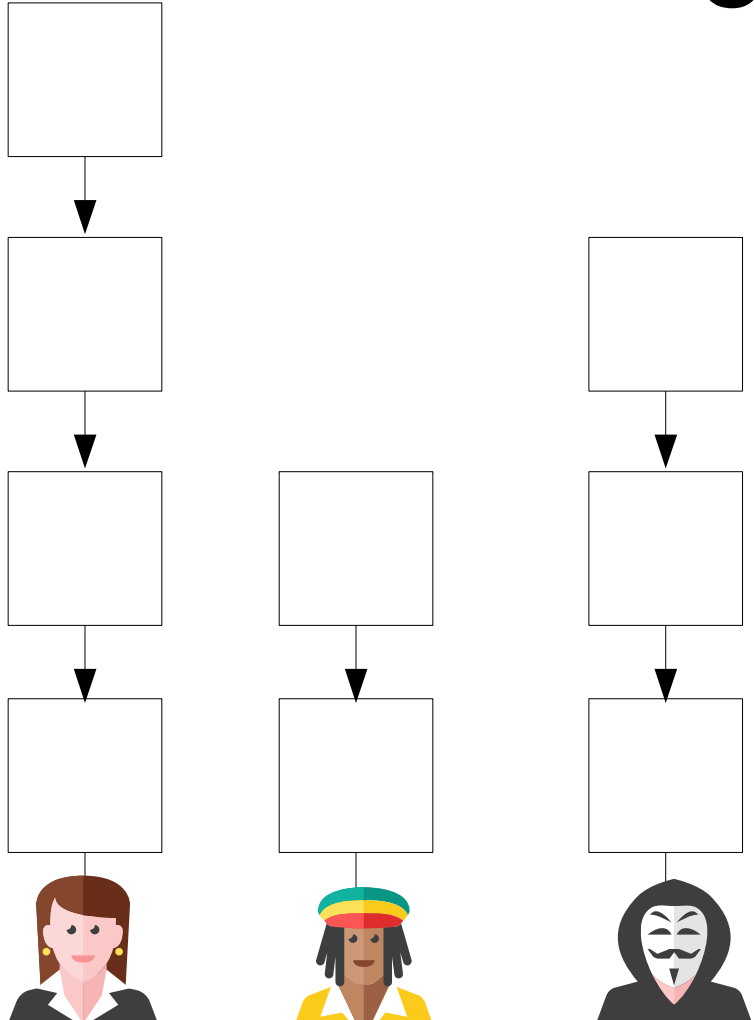
ClaimChains

claimchain.github.io

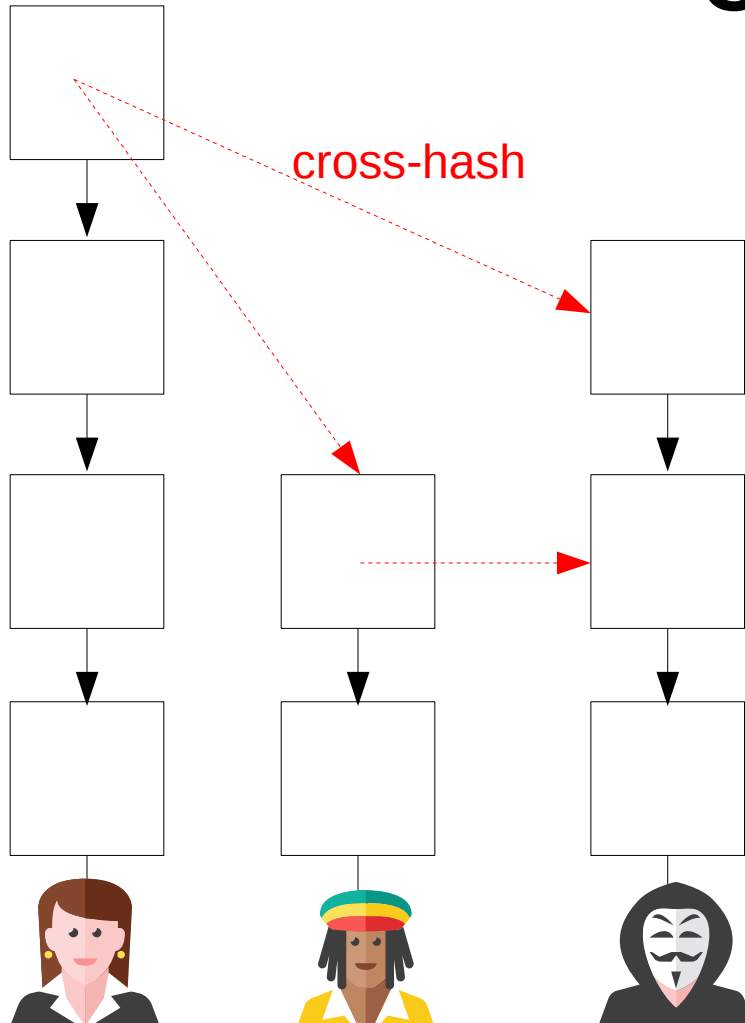


ClaimChains

- A ClaimChain for each user/device/identity
- Blocks appended as needed
- Compromises appear as ClaimChain forks
- Owner selects who can read a specific claim – all readers get the same content

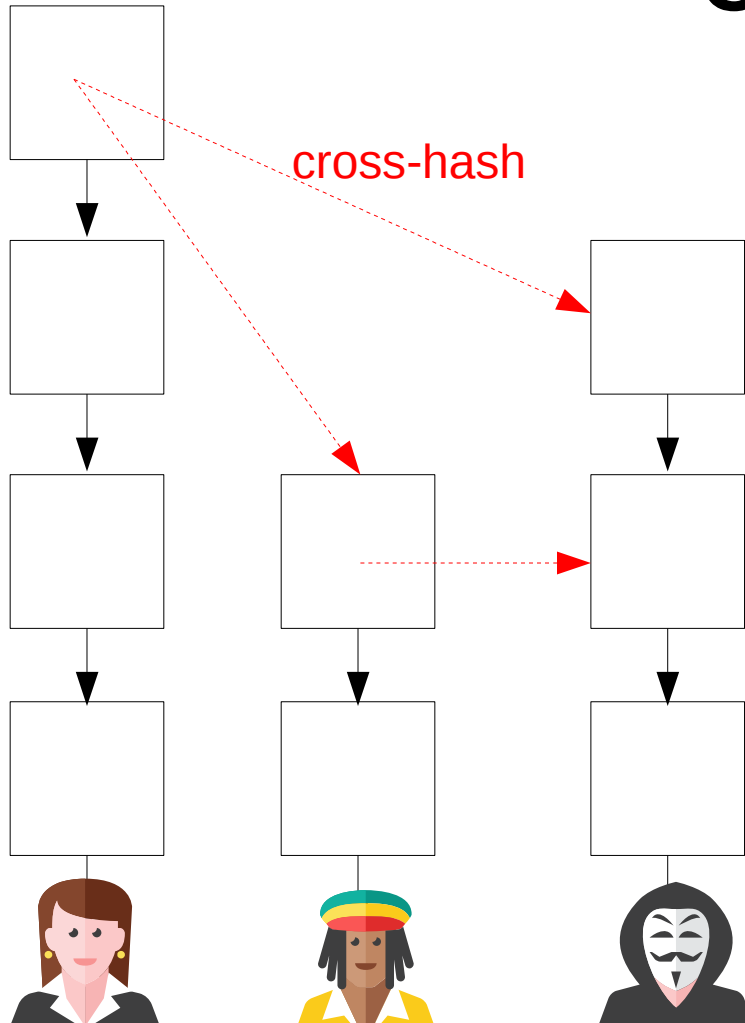


ClaimChains



- A ClaimChain for each user/device/identity
- Blocks appended as needed
- Compromises appear as ClaimChain forks
- Owner selects who can read a specific claim – all readers get the same content

ClaimChains



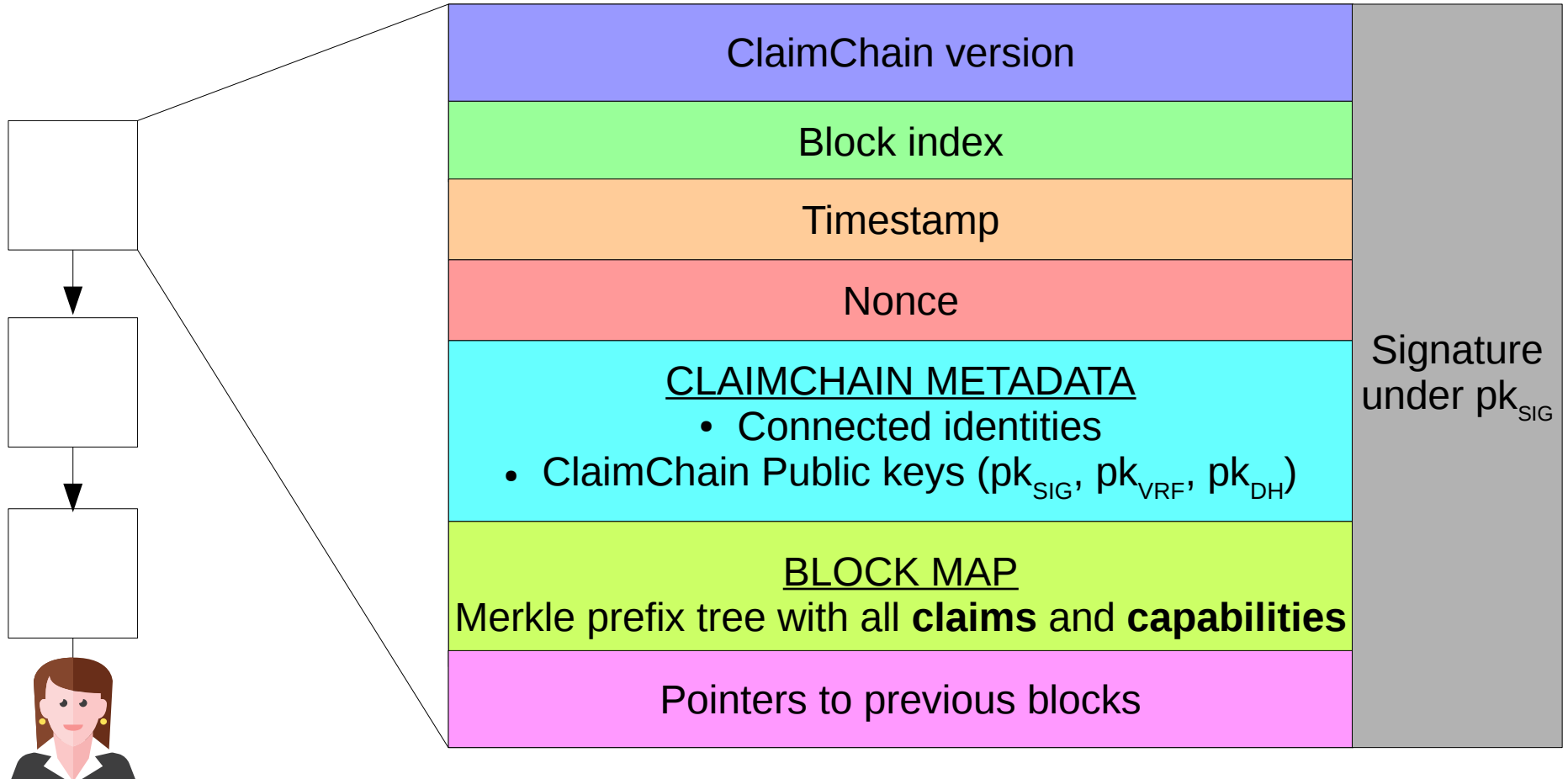
- A ClaimChain for each user/device/identity
- Blocks appended as needed
- Compromises appear as ClaimChain forks
- Owner selects who can read a specific claim – all readers get the same content

- Propagation of key updates in “cliques” of user
- Vouch for the latest state of a friend’s ClaimChain
- Friend introductions - Social validation – Web of Trust
... while preserving privacy

Overview

- ClaimChains are high-integrity, authenticated data stores that can support generic claims
- Privacy: a capabilities mechanism for fine-grained claim-specific access control
- Non-equivocation: all readers of a private claim get the same view
- Cross-hashing enables the propagation and vouching of the latest state of linked ClaimChains
- Equivocation attempts a compromise produce non-repudiable cryptographic evidence (“ClaimChain forks”)
- Flexible in terms of deployment
- Efficient “selective sharing” of claims

ClaimChains block structure

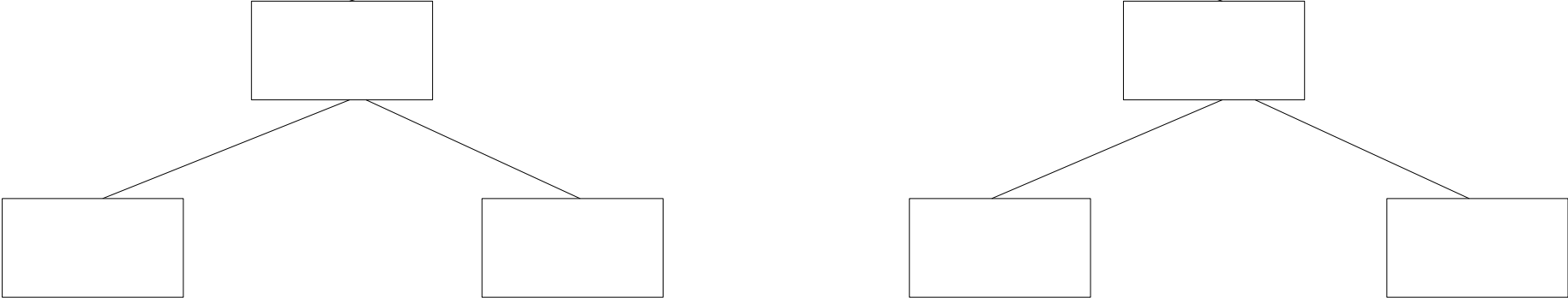


Block claim map: Adding a claim

label = bob@riseup.net
claim = 0515b693e5



ROOT



Block claim map: Adding a claim

label = bob@riseup.net
claim = 0515b693e5



ROOT



1) Compute claim key $k = \text{VRF}_{\text{woman}}(\text{bob@riseup.net} || \text{nonce})$



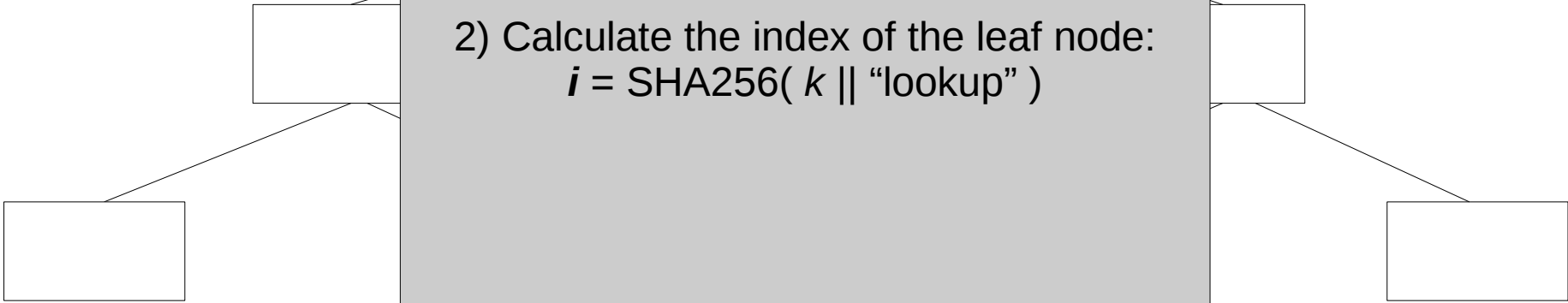
Block claim map: Adding a claim

label = bob@riseup.net
claim = 0515b693e5



ROOT 

- 1) Compute claim key $k = \text{VRF}_{\text{woman}}(\text{person} \parallel \text{nonce})$
- 2) Calculate the index of the leaf node:
 $i = \text{SHA256}(k \parallel \text{"lookup"})$



Block claim map: Adding a claim

label = bob@riseup.net
claim = 0515b693e5



ROOT



1) Compute claim key $k = \text{VRF}_{\text{👤}}(\text{👤} || \text{nonce})$

2) Calculate the index of the leaf node:
 $i = \text{SHA256}(k || \text{"lookup"})$

3) Generate a symm. enc. key
 $K = \text{SHA256}(k || \text{"enc"})$

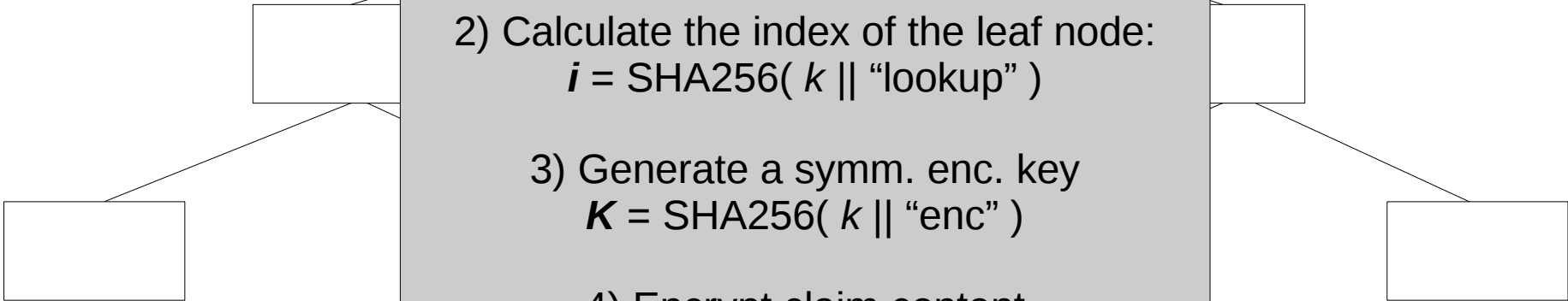


Block claim map: Adding a claim

label = bob@riseup.net
claim = 0515b693e5 

ROOT 

- 1) Compute claim key $k = \text{VRF}_{\text{person}}(\text{person} \parallel \text{nonce})$
- 2) Calculate the index of the leaf node:
 $i = \text{SHA256}(k \parallel \text{"lookup"})$
- 3) Generate a symm. enc. key
 $K = \text{SHA256}(k \parallel \text{"enc"})$
- 4) Encrypt claim content
 $C = \text{Enc}_K(\text{VRFproof} + \text{"0515b693e5"})$



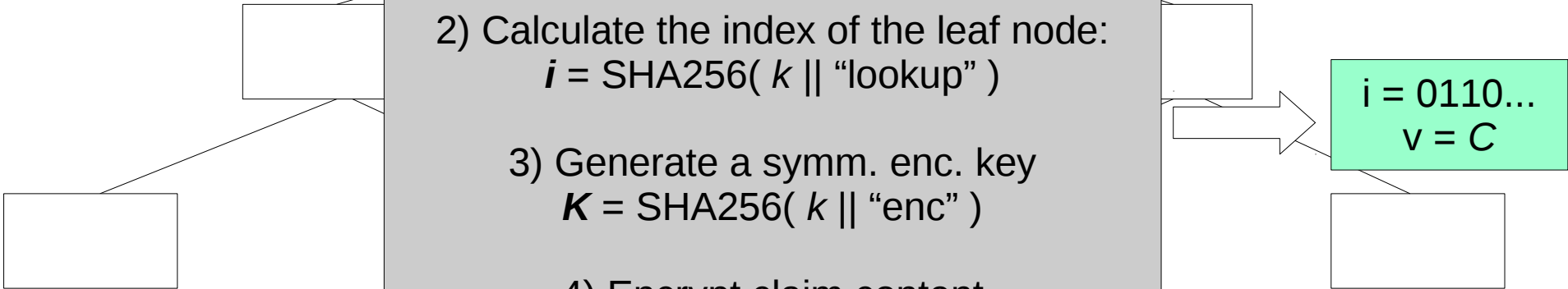
Block claim map: Adding a claim

label = bob@riseup.net
claim = 0515b693e5 

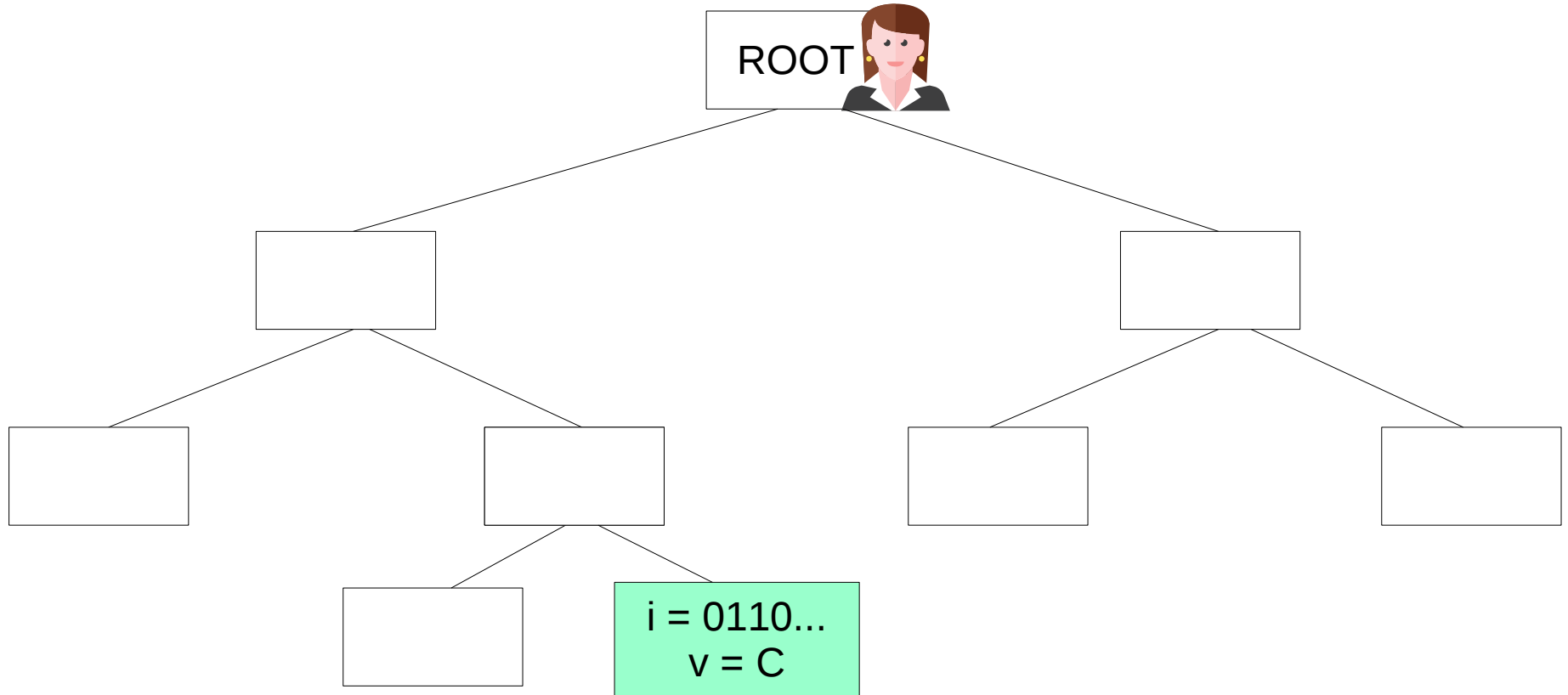
ROOT 

- 1) Compute claim key $k = \text{VRF}_{\text{root}}(\text{label} \parallel \text{nonce})$
- 2) Calculate the index of the leaf node:
 $i = \text{SHA256}(k \parallel \text{"lookup"})$
- 3) Generate a symm. enc. key
 $K = \text{SHA256}(k \parallel \text{"enc"})$
- 4) Encrypt claim content
 $C = \text{Enc}_K(\text{VRFproof} + \text{"0515b693e5"})$

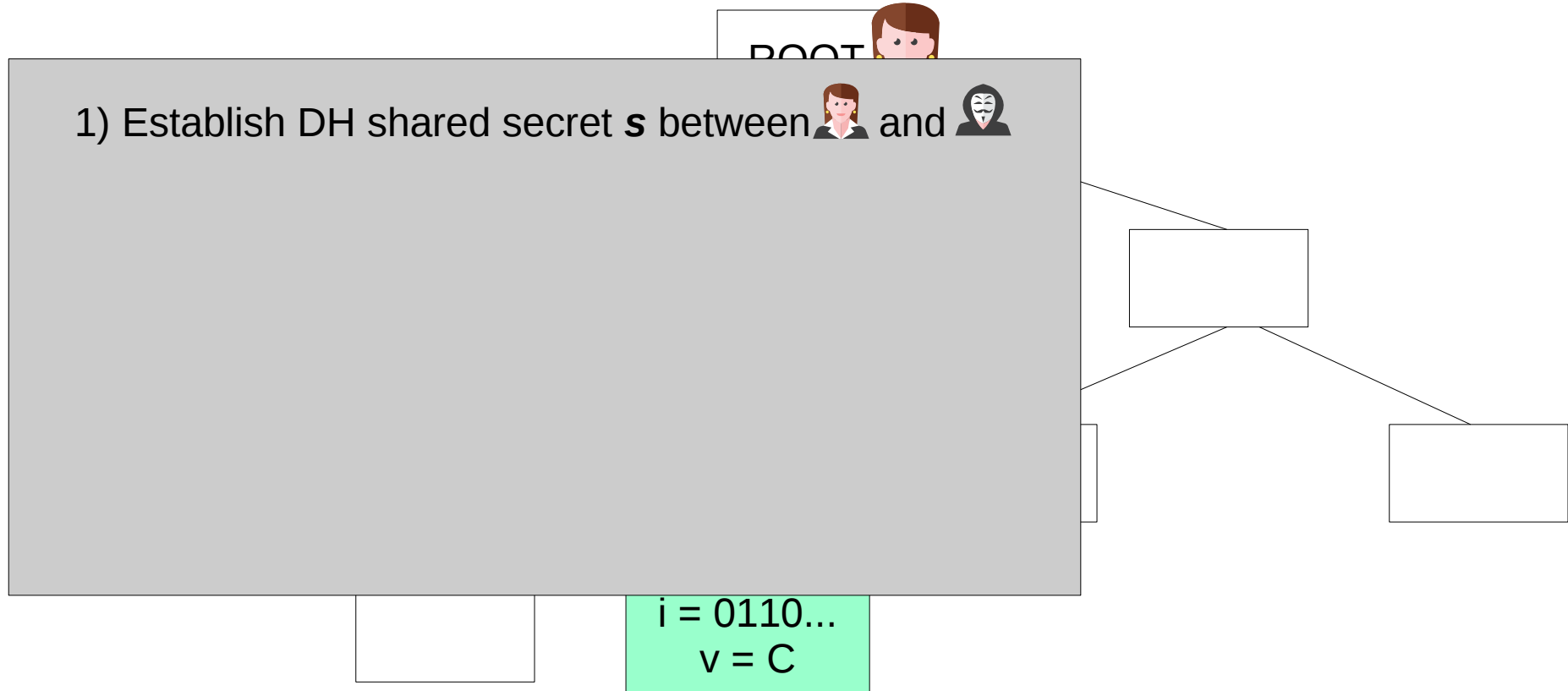
$i = 0110\dots$
 $v = C$



Block claim map: Adding a capability for to read



Block claim map: Adding a capability for to read



Block claim map: Adding a capability for to read

ROOT 



1) Establish DH shared secret s between  and 

2) Derive the capability lookup key
 $i = \text{SHA256}(\textit{nonce} \parallel s \parallel \textit{"lookup"})$

$i = 0110\dots$
 $v = C$

Block claim map: Adding a capability for to read

ROOT 

1) Establish DH shared secret s between  and 

2) Derive the capability lookup key
 $i = \text{SHA256}(\textit{nonce} \parallel s \parallel \textit{"lookup"})$

3) Derive the symm. enc. key
 $K = \text{SHA256}(\textit{nonce} \parallel s \parallel \textit{"enc"})$

$i = 0110\dots$
 $v = C$



Block claim map: Adding a capability for to read

ROOT 

1) Establish DH shared secret s between  and 

2) Derive the capability lookup key
 $i = \text{SHA256}(\textit{nonce} \parallel s \parallel \textit{"lookup"})$



3) Derive the symm. enc. key
 $K = \text{SHA256}(\textit{nonce} \parallel s \parallel \textit{"enc"})$

4) Encrypt claim key VRF ( \parallel nonce)
 $C = \text{Enc}_K(k)$ 

$i = 0110\dots$
 $v = C$



Block claim map: Adding a capability for to read

ROOT 

1) Establish DH shared secret s between  and 

2) Derive the capability lookup key
 $i = \text{SHA256}(\text{nonce} \parallel s \parallel \text{"lookup"})$

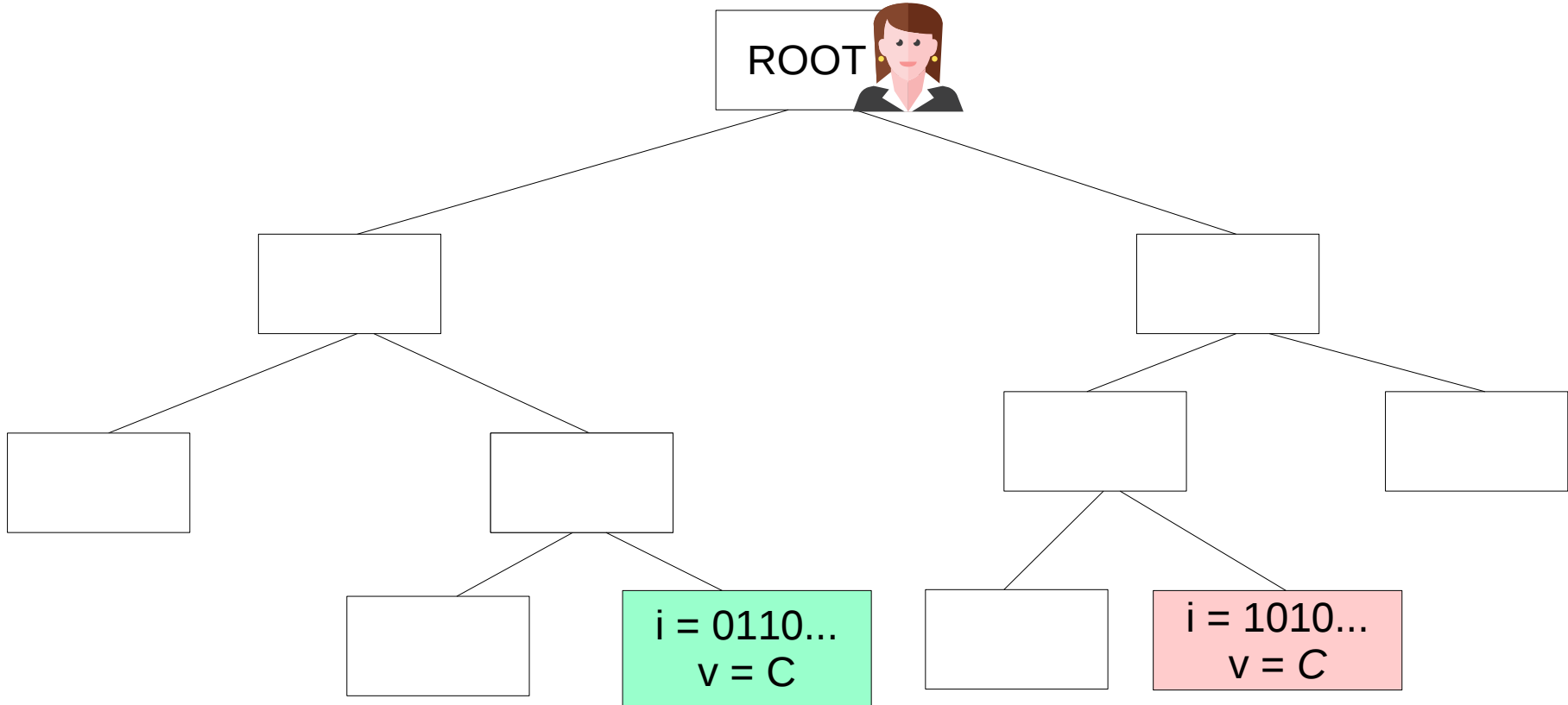
3) Derive the symm. enc. key
 $K = \text{SHA256}(\text{nonce} \parallel s \parallel \text{"enc"})$

4) Encrypt claim key VRF (  \parallel nonce)
 $C = \text{Enc}_K(k)$



$i = 1010\dots$
 $v = C$

$i = 0110\dots$
 $v = C$

Block claim map: retrieving the latest update for



Block claim map: retrieving the latest update for



1) Establish DH shared secret s between  and 



$v = C$

$v = C$

Block claim map: retrieving the latest update for

1) Establish DH shared secret s between  and 



2) Derive the capability lookup key
 $i = \text{SHA256}(\textit{nonce} \parallel s \parallel \textit{"lookup"})$



$v = C$

$v = C$

Block claim map: retrieving the latest update for

1) Establish DH shared secret s between  and 

2) Derive the capability lookup key
 $i = \text{SHA256}(\textit{nonce} \parallel s \parallel \textit{"lookup"})$



3) Derive the symm. enc. key
 $K = \text{SHA256}(\textit{nonce} \parallel s \parallel \textit{"enc"})$



$v = C$


$v = C$

Block claim map: retrieving the latest update for

1) Establish DH shared secret s between  and 

2) Derive the capability lookup key
 $i = \text{SHA256}(\textit{nonce} \parallel s \parallel \textit{"lookup"})$

3) Derive the symm. enc. key
 $K = \text{SHA256}(\textit{nonce} \parallel s \parallel \textit{"enc"})$



4) Retrieve capability block and decrypt it with K
Result: key for 's claim

$i = 1010\dots$
 $v = C$

$v = C$


$v = C$


Block claim map: retrieving the latest update for

1) Establish DH shared secret s between  and 

2) Derive the capability lookup key
 $i = \text{SHA256}(\textit{nonce} \parallel s \parallel \textit{"lookup"})$

3) Derive the symm. enc. key
 $K = \text{SHA256}(\textit{nonce} \parallel s \parallel \textit{"enc"})$

4) Retrieve capability block and decrypt it with K
Result: key for 's claim

5) Retrieve 's claim and decrypt it



$i = 1010\dots$
 $v = C$

$i = 0110\dots$
 $v = C$

$v = C$


$v = C$

Block claim map: retrieving the latest update for

1) Establish DH shared secret s between  and 

2) Derive the capability lookup key
 $i = \text{SHA256}(\textit{nonce} \parallel s \parallel \textit{"lookup"})$

3) Derive the symm. enc. key
 $K = \text{SHA256}(\textit{nonce} \parallel s \parallel \textit{"enc"})$

4) Retrieve capability block and decrypt it with K
Result: key for 's claim

5) Retrieve 's claim and decrypt it

6) Verify $\textit{VRFproof}$

$i = 1010\dots$
 $v = C$

$i = 0110\dots$
 $v = C$

$v = C$

$v = C$

Resilience

- Field research to understand user needs
- Collaboration with related communities
- Applied research:
 - Cryptographic games to define security and privacy properties
 - Formally verified implementation
- Simulations using real world data
- Interoperability and plans for gradual deployment
- User-centric design
- Multidisciplinarity
- Open Innovation (open access and extendability)



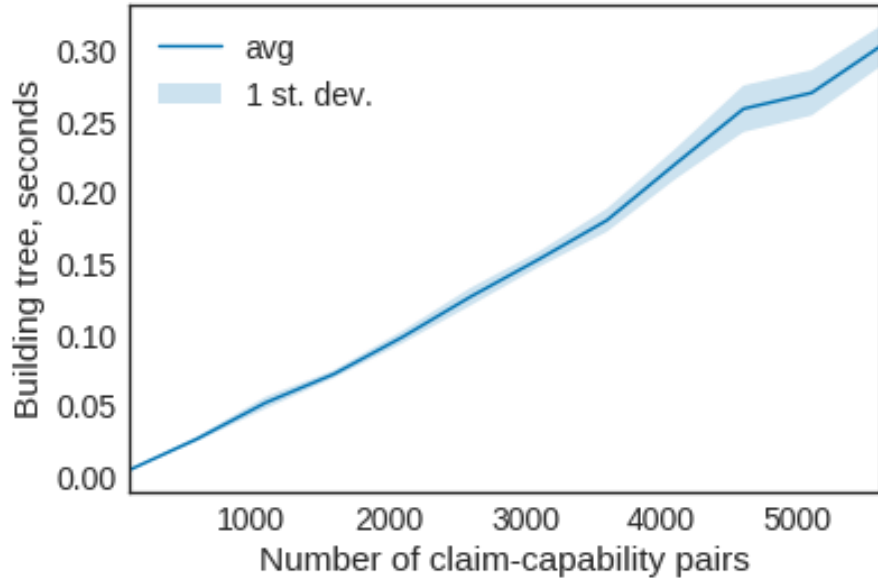
תודה!

Thank you

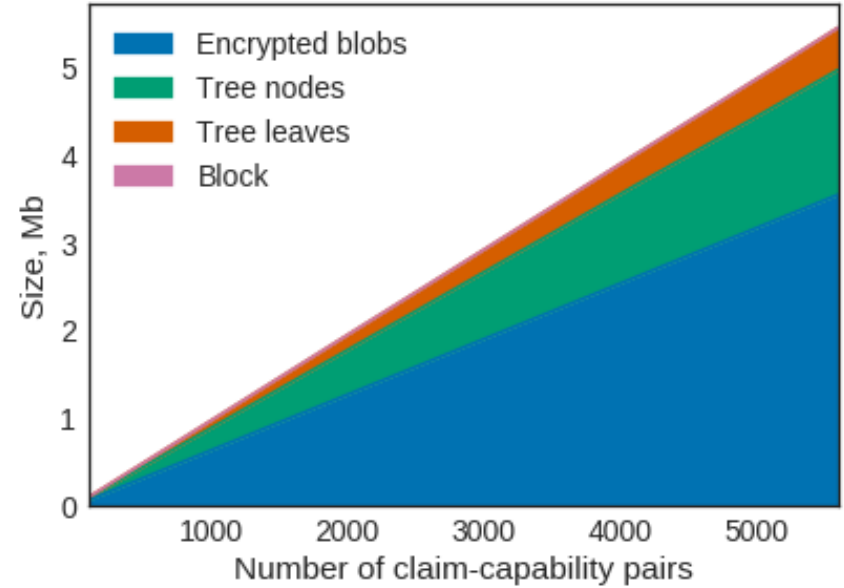
@misaakidis

claimchain.github.io

Evaluation of scalability

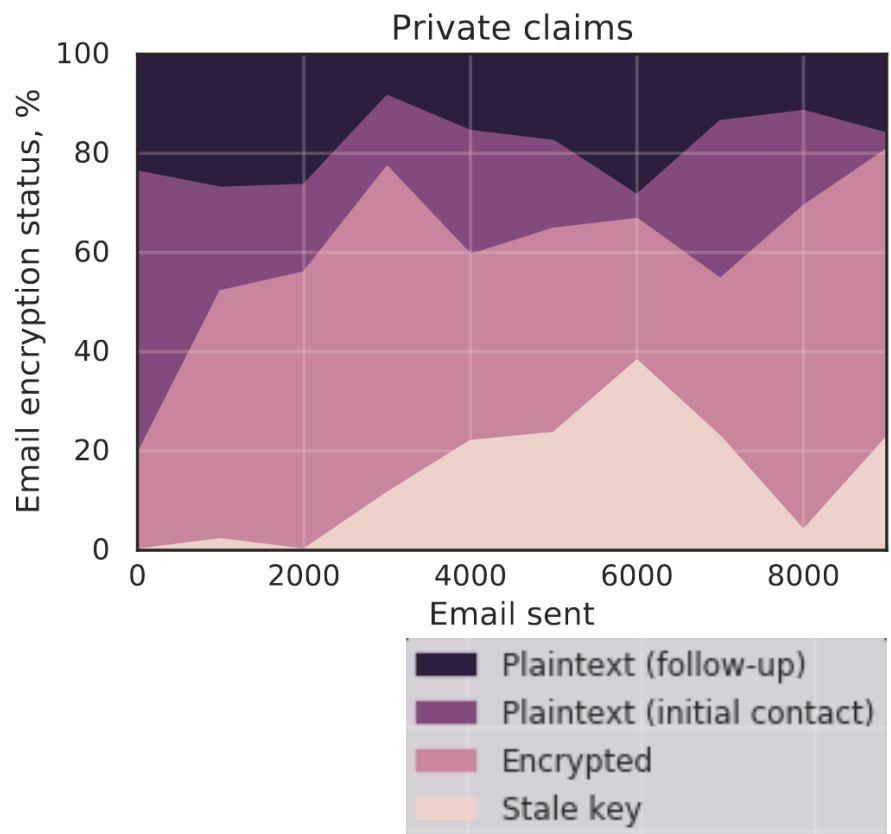


Claim map construction time

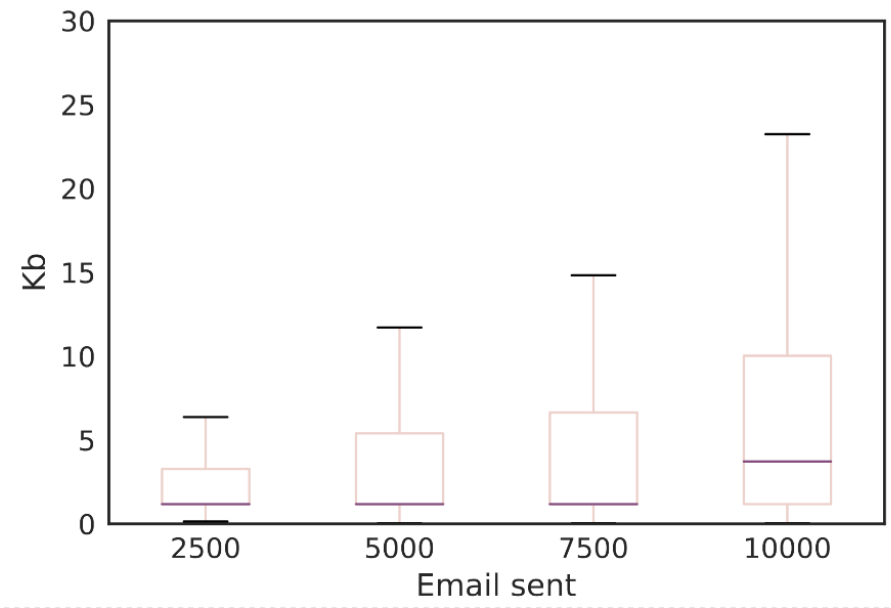


Cumulative block storage size

Key propagation in a fully decentralized setting

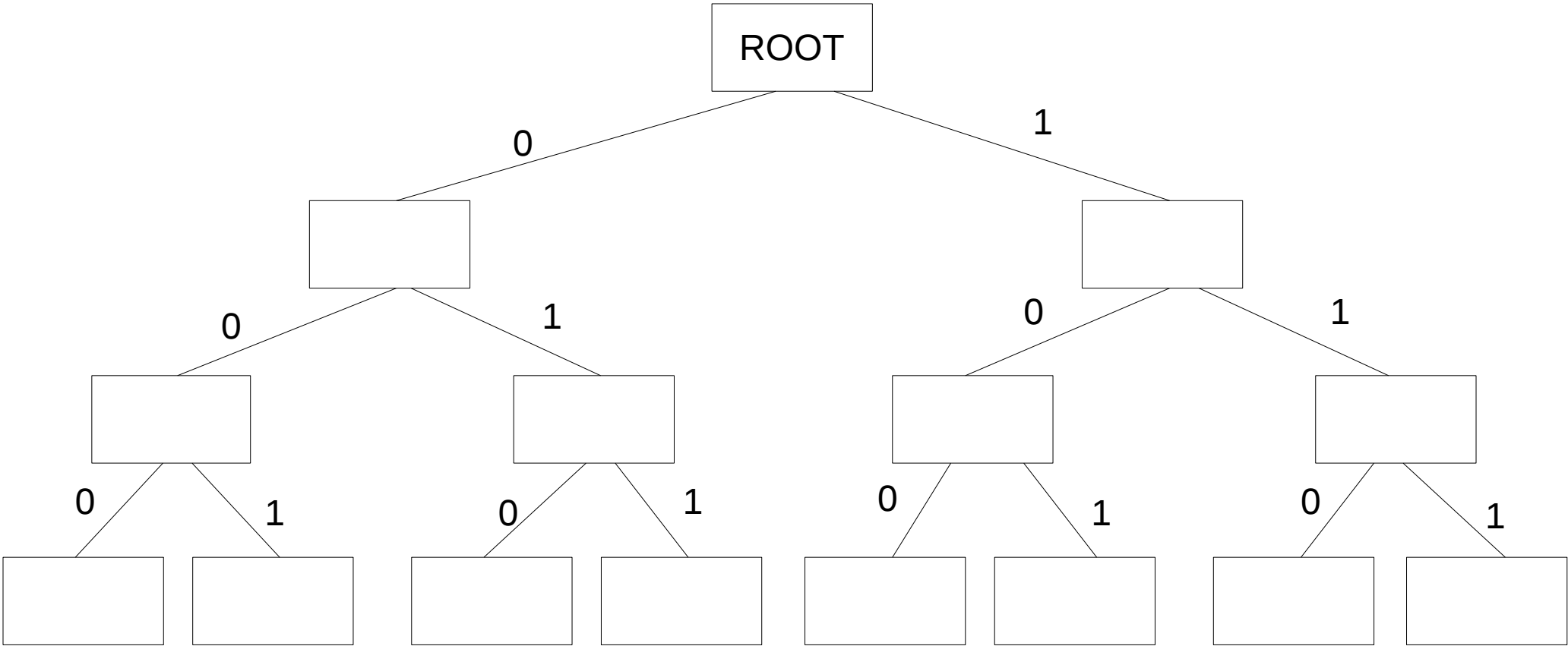


Email encryption status (%)



Outgoing bandwidth cost

Merkle binary prefix trees: Proof of inclusion



Merkle binary prefix trees: Proof of inclusion

(alice@riseup.net, 0x1A2B3C)

$\text{VRF}_{\text{pkVRF}}(\text{alice@riseup.net}) = 01011\dots$

ROOT

0

1

0

1

0

1

0

1

0

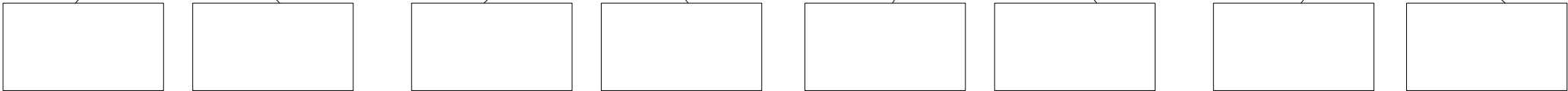
1

0

1

0

1



Merkle binary prefix trees: Proof of inclusion

(alice@riseup.net, 0x1A2B3C)

$\text{VRF}_{\text{pkVRF}}(\text{alice@riseup.net}) = 01011\dots$

ROOT

0

1

0

1

0

1

0

1

0

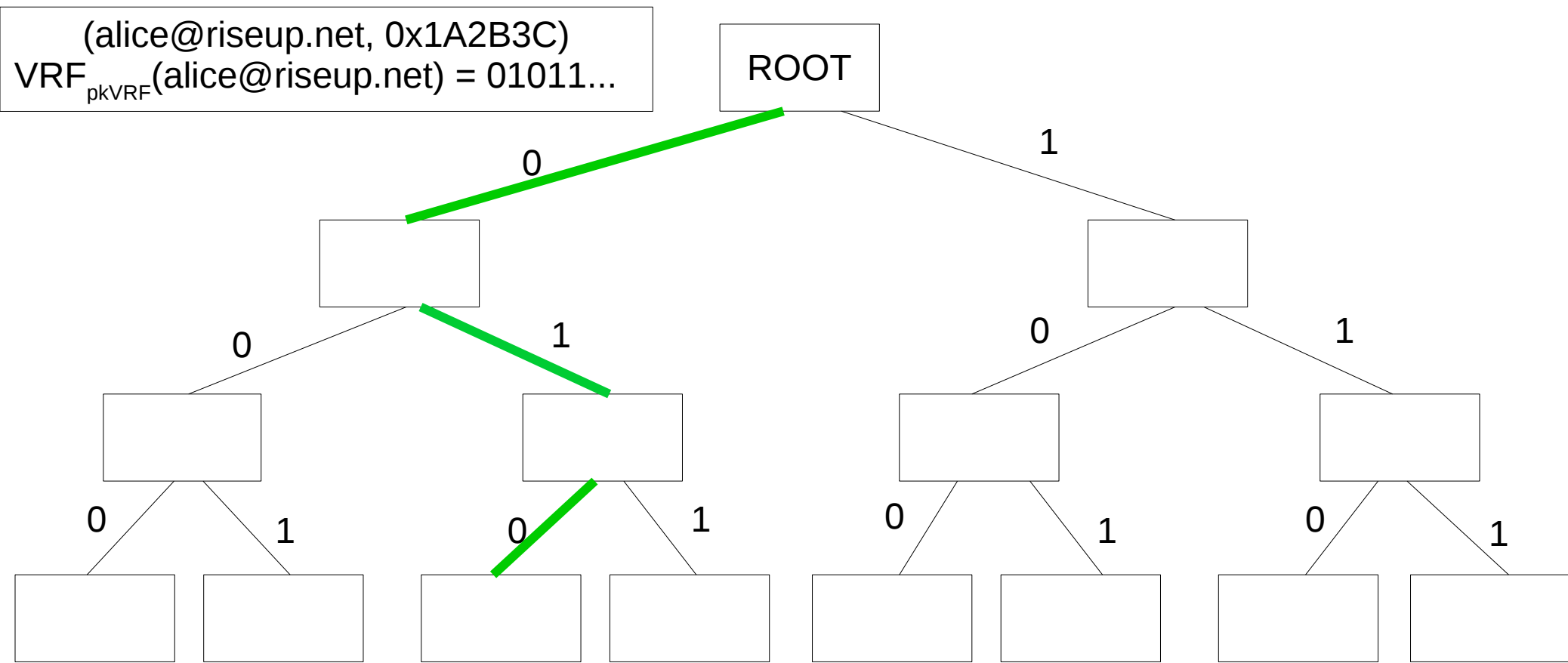
1

0

1

0

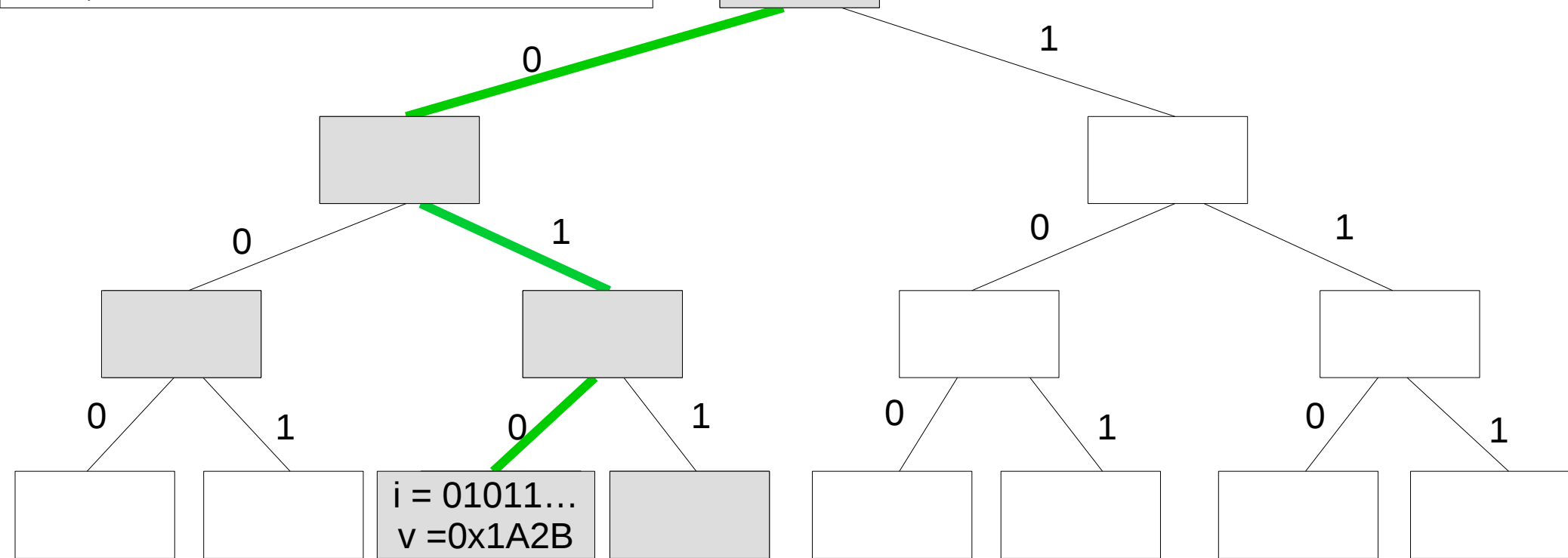
1



Merkle binary prefix trees: Proof of inclusion

(alice@riseup.net, 0x1A2B3C)
 $\text{VRF}_{\text{pkVRF}}(\text{alice@riseup.net}) = 01011\dots$

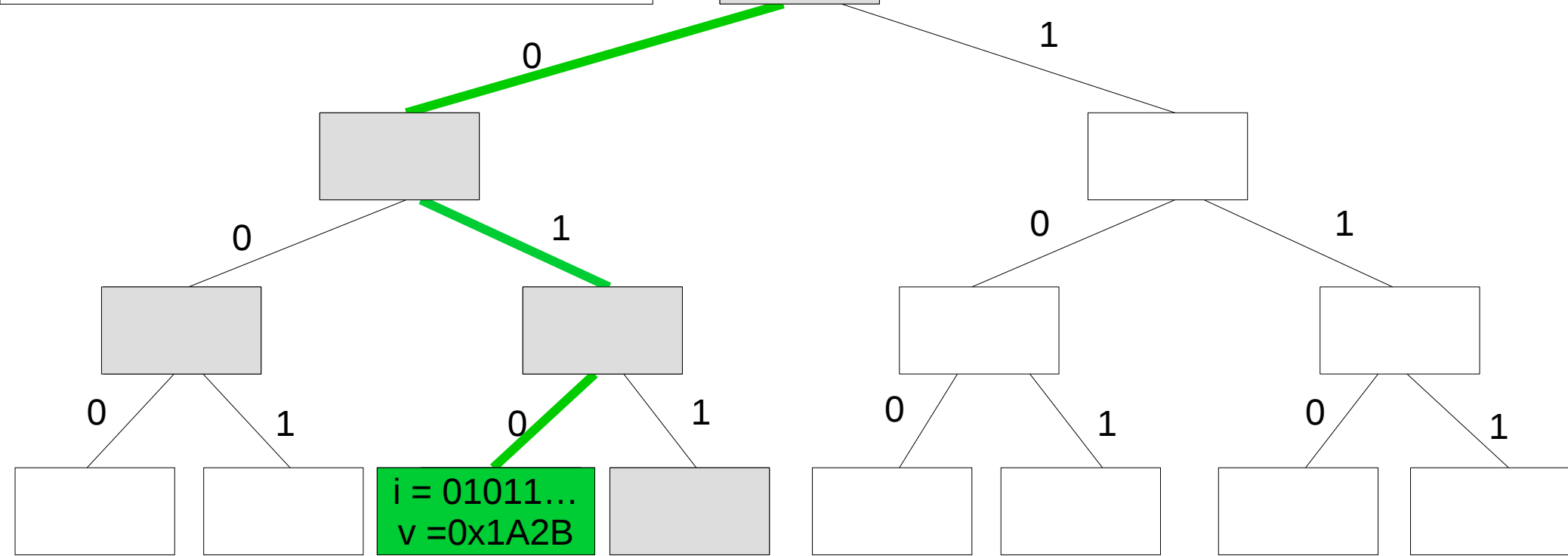
ROOT



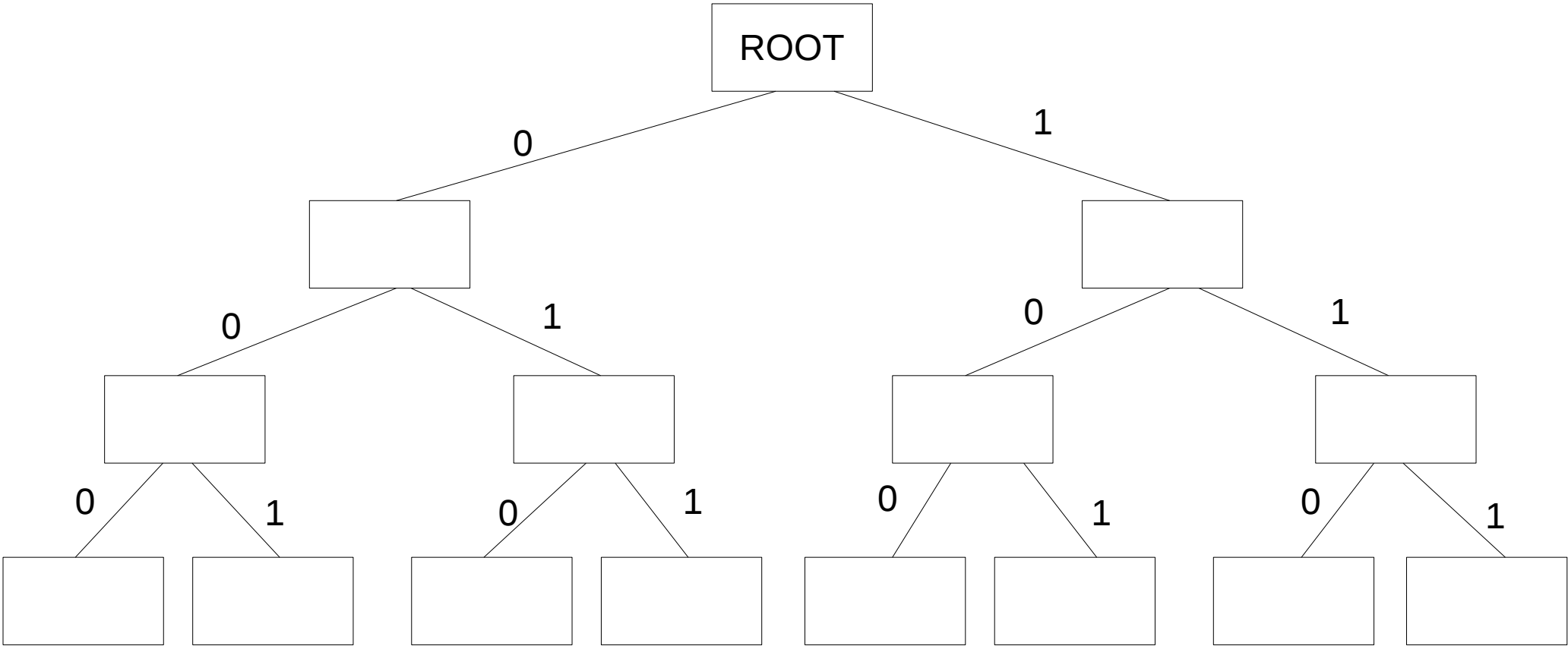
Merkle binary prefix trees: Proof of inclusion

(alice@riseup.net, 0x1A2B3C)
 $\text{VRF}_{\text{pkVRF}}(\text{alice@riseup.net}) = 01011\dots$

ROOT



Merkle binary prefix trees: Proof of absence



Merkle binary prefix trees: Proof of absence

$\text{VRF}_{\text{pkVRF}}(\text{bob@riseup.net}) = 11001\dots$

ROOT

0

1

0

1

0

1

0

1

0

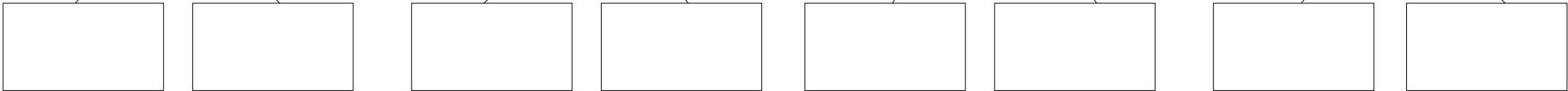
1

0

1

0

1



Merkle binary prefix trees: Proof of absence

$\text{VRF}_{\text{pkVRF}}(\text{bob@riseup.net}) = 11001\dots$

ROOT

0

1

0

1

0

1

0

1

0

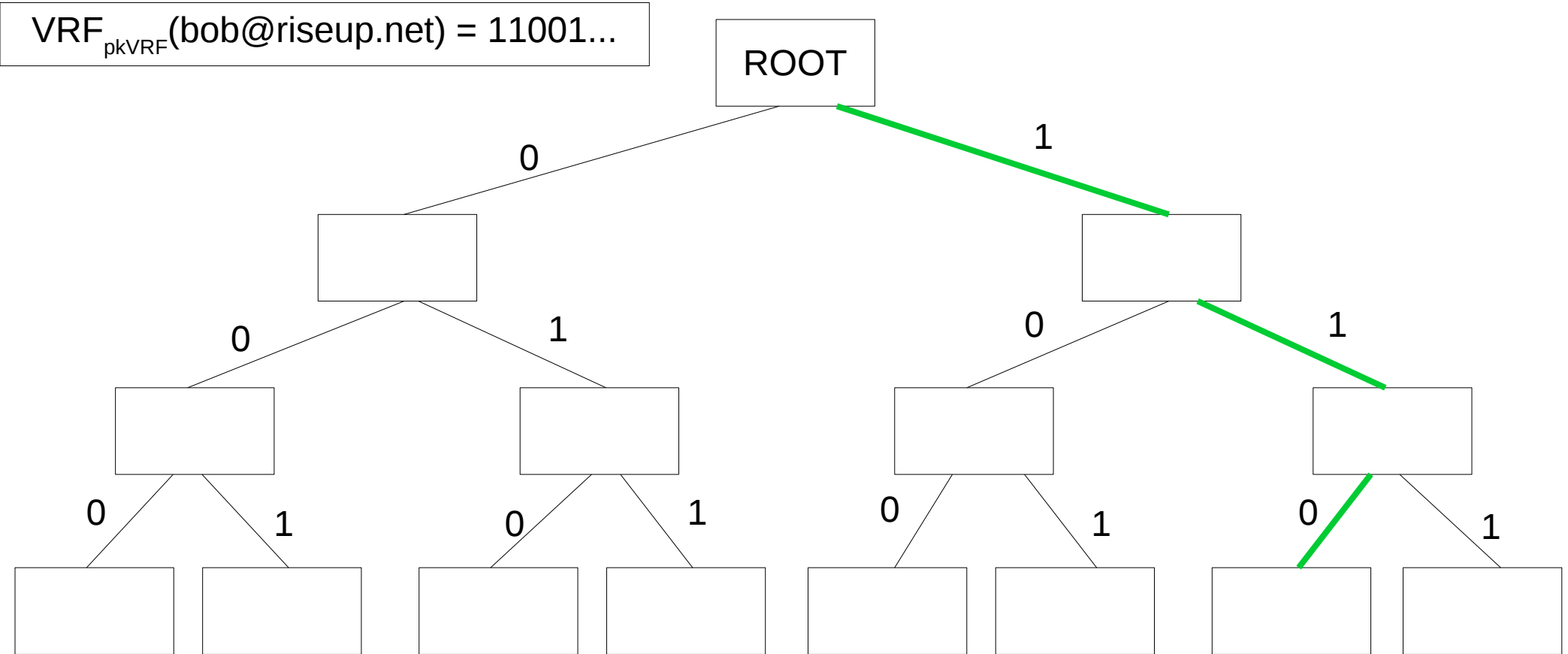
1

0

1

0

1



Merkle binary prefix trees: Proof of absence

$\text{VRF}_{\text{pkVRF}}(\text{bob@riseup.net}) = 11001\dots$

ROOT

0

1

0

1

0

1

0

1

0

1

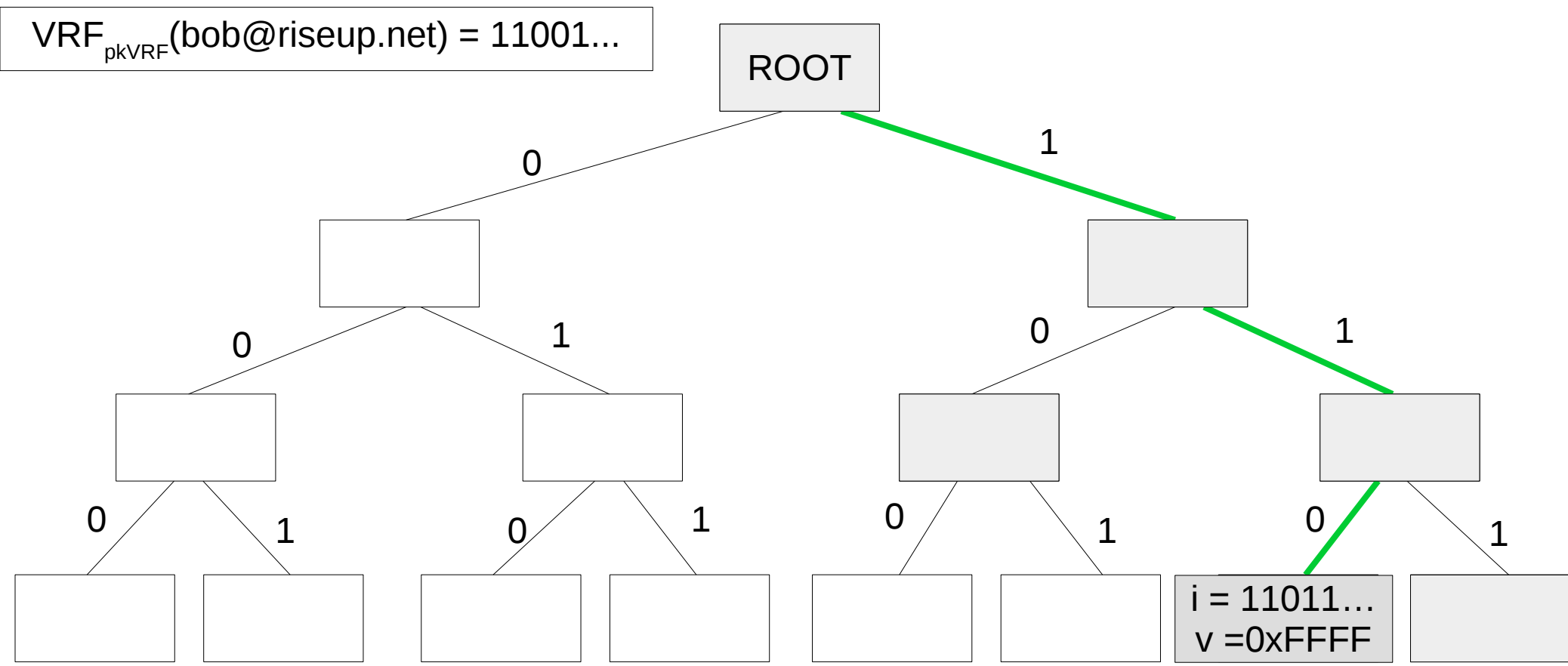
0

1

0

1

$i = 11011\dots$
 $v = 0xFFFF$



Merkle binary prefix trees: Proof of absence

$\text{VRF}_{\text{pkVRF}}(\text{bob@riseup.net}) = 11001\dots$

ROOT

0

1

0

1

0

1

0

1

0

1

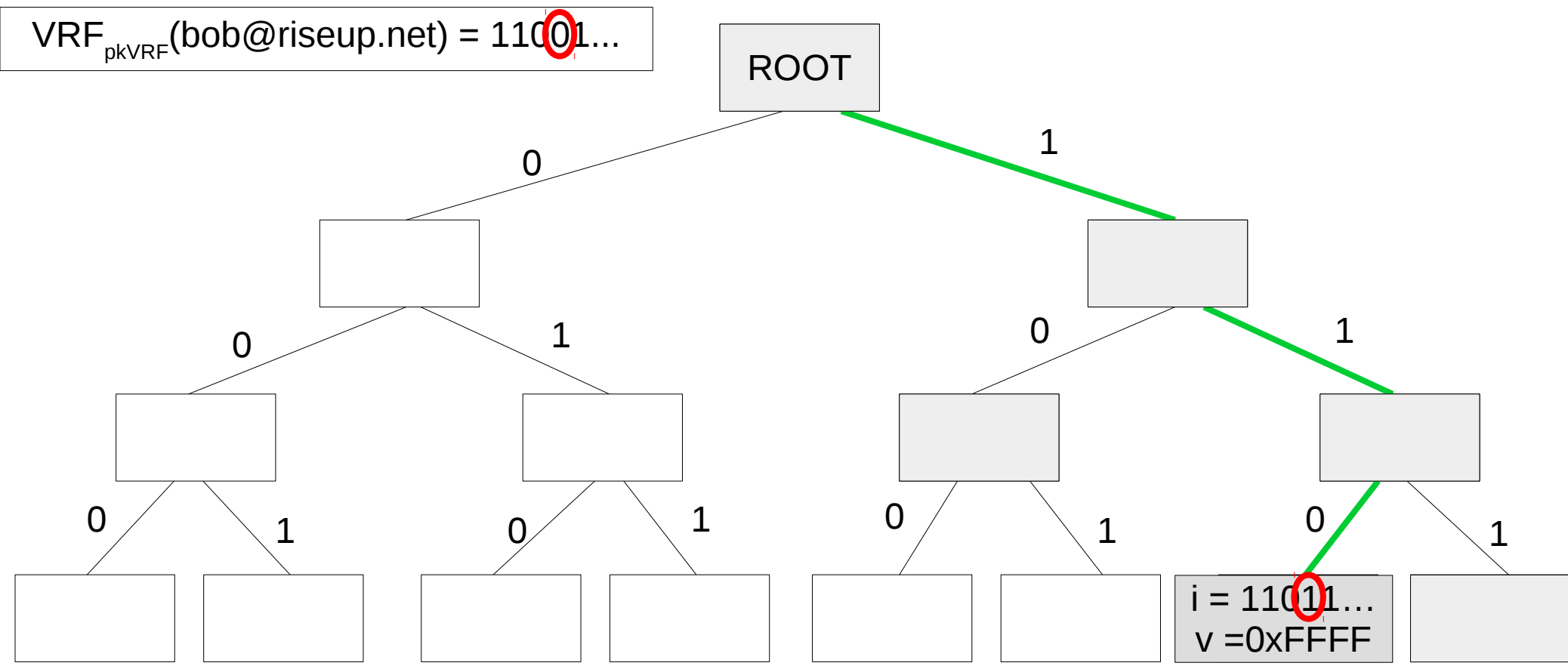
0

1

0

1

$i = 11011\dots$
 $v = 0xFFFF$



Merkle binary prefix trees: Proof of absence

$\text{VRF}_{\text{pkVRF}}(\text{bob@riseup.net}) = 11001\dots$

ROOT

0

1

0

1

0

1

0

1

0

1

0

1

0

1

$i = 11011\dots$
 $v = 0xFFFF$

