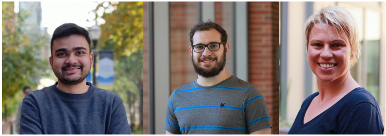
Polar Coded Merkle Tree: Improved Detection of Data Availability Attacks in Blockchain Systems

Debarnab Mitra, Lev Tauz, and Lara Dolecek

Electrical and Computer Engineering
University of California, Los Angeles

ISIT 2022



UCLA Samueli School of Engineering

Blockchain

- Distributed Ledger
- Decentralized trust platforms

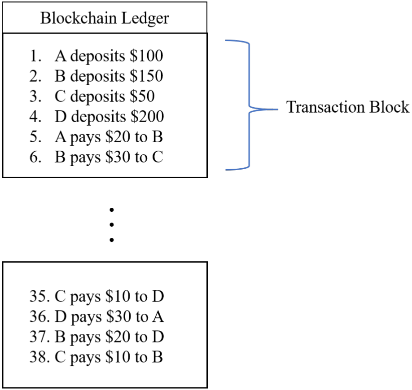
Blockchain

- Distributed Ledger
- Decentralized trust platforms
- Main Application:
    - Finance and currency

Blockchain

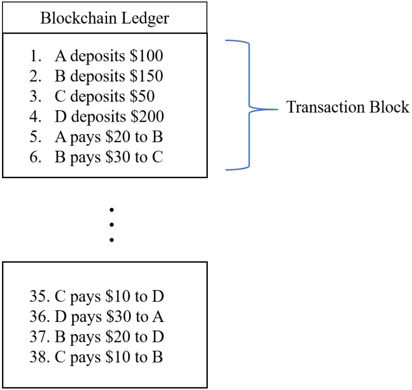
- Distributed Ledger
- Decentralized trust platforms
- Main Application:
  - Finance and currency
- Emerging Applications:
  - Healthcare services
  - Supply chain management
  - Industrial IoT
  - e-voting

Blockchain



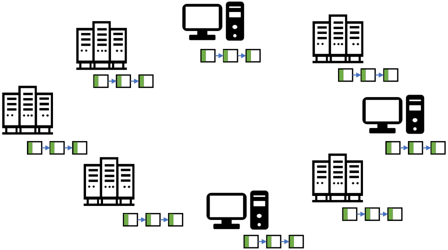
- Ledger of transactions

Blockchain



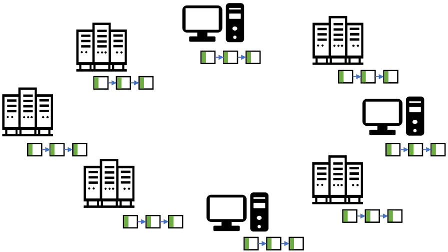
- Ledger of transactions
- Arranged in the form of blocks

Blockchain



- Ledger of transactions
- Arranged in the form of blocks
- Stored by a network of nodes
- Full nodes: store a copy of the entire ledger

Blockchain

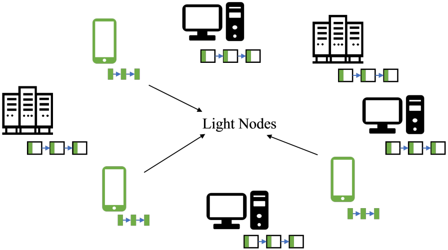


- Ledger of transactions
- Arranged in the form of blocks
- Stored by a network of nodes
- Full nodes: store a copy of the entire ledger

- Bitcoin ledger size $\sim$ 400GB[1]
- Ethereum ledger size $\sim$ 730GB[2]

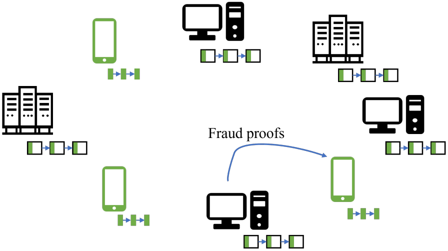
As of 6/5/2022, [1]https://www.blockchain.com/charts/blocks-size
[2]https://etherscan.io/chartsync/chaindefault

Blockchain



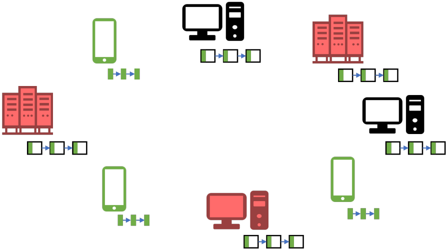
- Ledger of transactions
- Arranged in the form of blocks
- Stored by a network of nodes
- Full nodes: store a copy of the entire ledger
- Light nodes: only store block headers

Blockchain



- Ledger of transactions
- Arranged in the form of blocks
- Stored by a network of nodes
- Full nodes: store a copy of the entire ledger
- Light nodes: only store block headers $\rightarrow$ rely on honest full nodes for fraud proofs

Blockchain

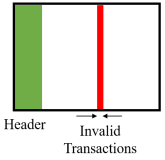


- Ledger of transactions
- Arranged in the form of blocks
- Stored by a network of nodes
- Full nodes: store a copy of the entire ledger
- Light nodes: only store block headers $\rightarrow$ rely on honest full nodes for fraud proofs

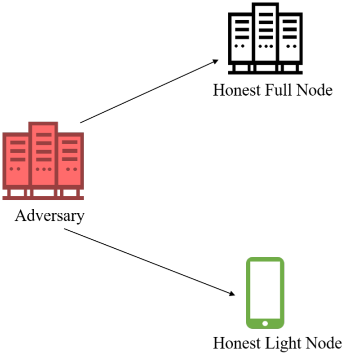
Systems with light nodes and a dishonest majority of full nodes are vulnerable to data availability attacks [Al-Bassam '18], [Yu '19]

Data Availability (DA) Attack

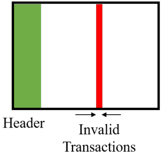
Adversary creates an invalid block



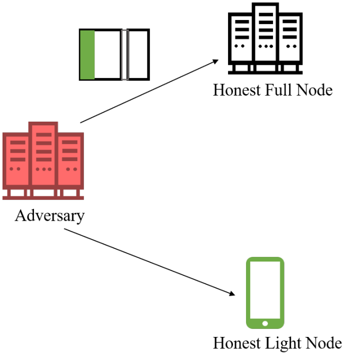
Data Availability (DA) Attack



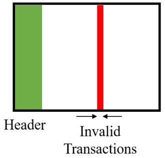
Adversary creates an invalid block



Data Availability (DA) Attack

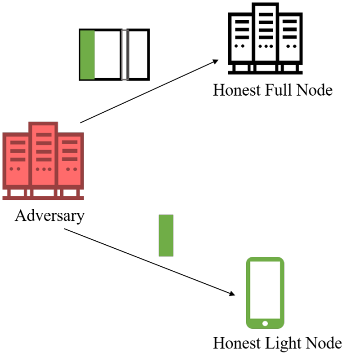


Adversary creates an invalid block

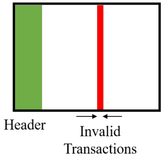


- Adversary: Provides block to Full node but hides invalid portion

Data Availability (DA) Attack

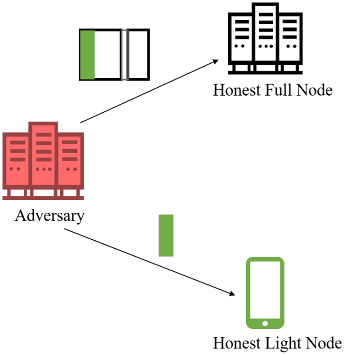


Adversary creates an invalid block

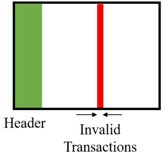


- Adversary: Provides block to Full node but hides invalid portion
  Provides header to Light node

Data Availability (DA) Attack

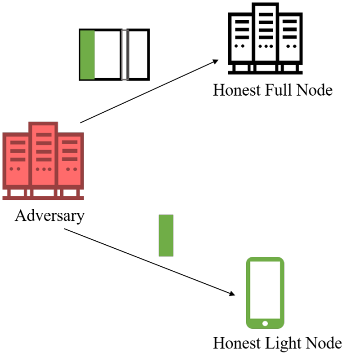


Adversary creates an invalid block

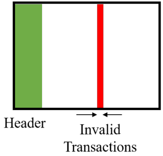


- Adversary: Provides block to Full node but hides invalid portion
  Provides header to Light node
- Honest Nodes: Cannot verify missing transactions

Data Availability (DA) Attack

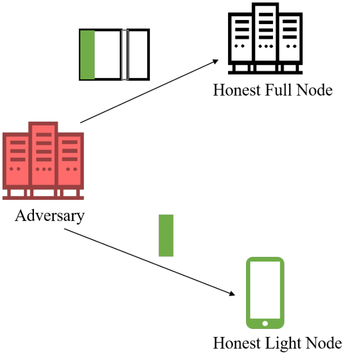


Adversary creates an invalid block

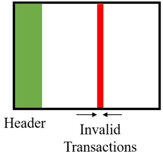


- Adversary: Provides block to Full node but hides invalid portion
  Provides header to Light node
- Honest Nodes: Cannot verify missing transactions → No fraud proof

Data Availability (DA) Attack

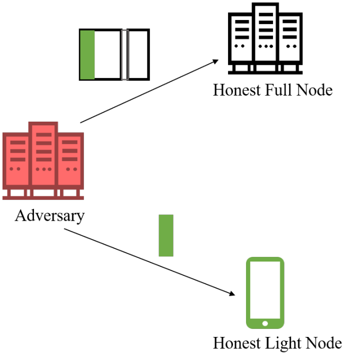


Adversary creates an invalid block

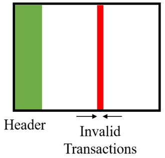


- Adversary: Provides block to Full node but hides invalid portion
  Provides header to Light node
- Honest Nodes: Cannot verify missing transactions → No fraud proof
- Light Nodes: No fraud proof

Data Availability (DA) Attack

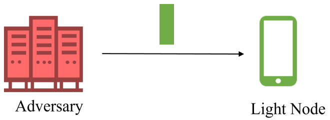


Adversary creates an invalid block

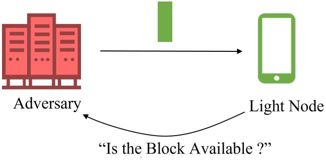


- Adversary: Provides block to Full node but hides invalid portion
  Provides header to Light node
- Honest Nodes: Cannot verify missing transactions → No fraud proof
- Light Nodes: No fraud proof → Accept the header

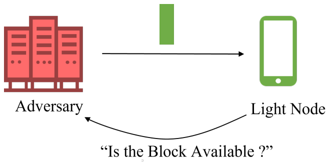
Solution: Light Node Sampling + Merkle Trees



Solution: Light Node Sampling + Merkle Trees

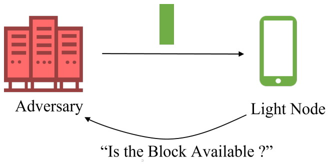


Solution: Light Node Sampling + Merkle Trees

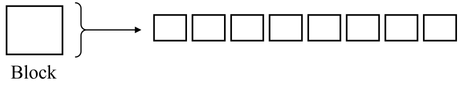


- Request/sample few random chunks of the block

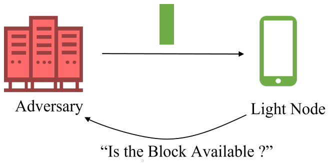
Solution: Light Node Sampling + Merkle Trees



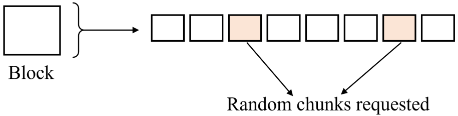
- Request/sample few random chunks of the block



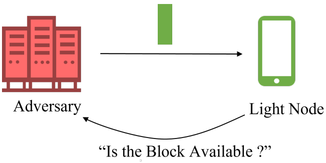
Solution: Light Node Sampling + Merkle Trees



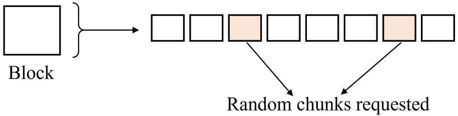
- Request/sample few random chunks of the block



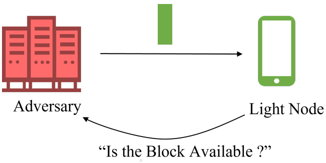
Solution: Light Node Sampling + Merkle Trees



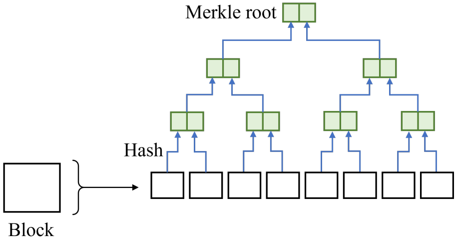
- Request/sample few random chunks of the block
- Use Merkle trees to ensure the integrity of returned chunks



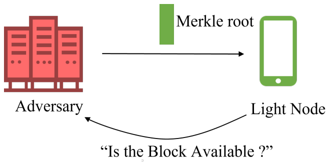
Solution: Light Node Sampling + Merkle Trees



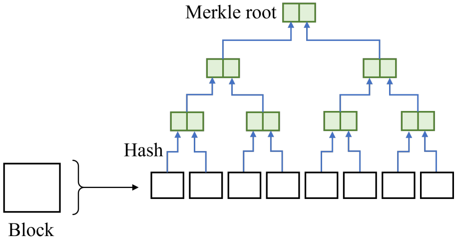
- Request/sample few random chunks of the block
- Use Merkle trees to ensure the integrity of returned chunks



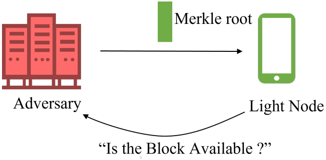
Solution: Light Node Sampling + Merkle Trees



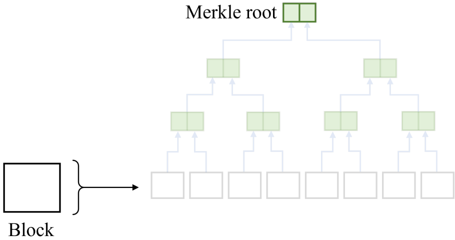
- Request/sample few random chunks of the block
- Use Merkle trees to ensure the integrity of returned chunks



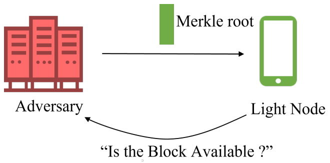
Solution: Light Node Sampling + Merkle Trees



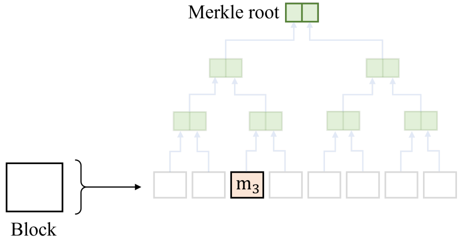
- Request/sample few random chunks of the block
- Use Merkle trees to ensure the integrity of returned chunks



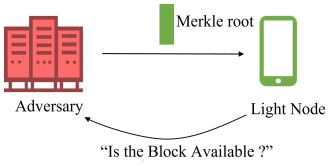
Solution: Light Node Sampling + Merkle Trees



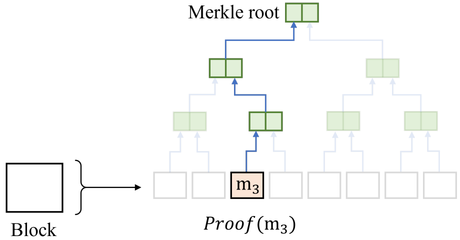
- Request/sample few random chunks of the block
- Use Merkle trees to ensure the integrity of returned chunks



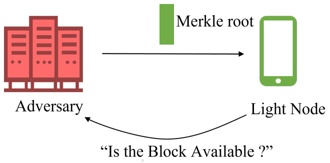
Solution: Light Node Sampling + Merkle Trees



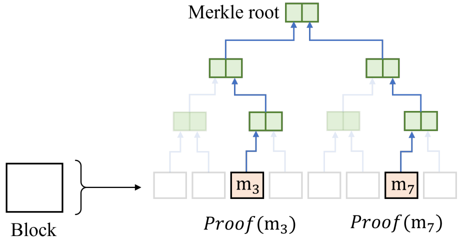
- Request/sample few random chunks of the block
- Use Merkle trees to ensure the integrity of returned chunks



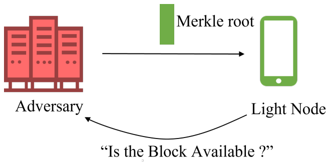
Solution: Light Node Sampling + Merkle Trees



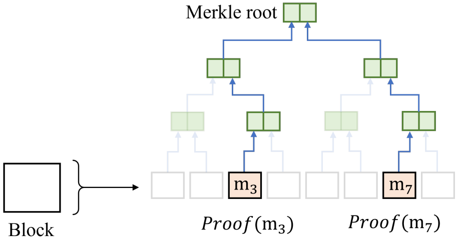
- Request/sample few random chunks of the block
- Use Merkle trees to ensure the integrity of returned chunks



Solution: Light Node Sampling + Merkle Trees

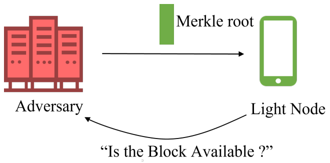


- Request/sample few random chunks of the block
- Use Merkle trees to ensure the integrity of returned chunks

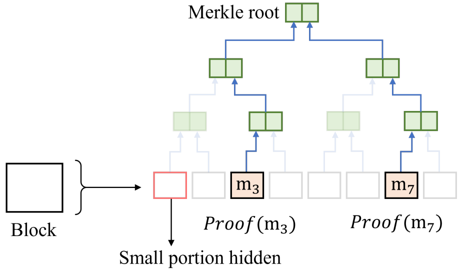


- Adversary can hide a small portion

Solution: Light Node Sampling + Merkle Trees

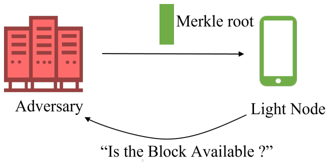


- Request/sample few random chunks of the block
- Use Merkle trees to ensure the integrity of returned chunks

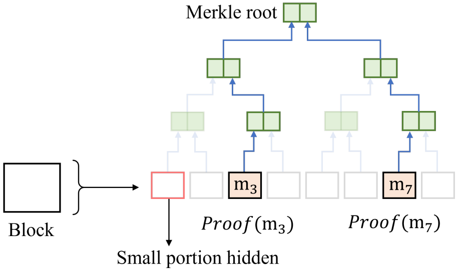


- Adversary can hide a small portion

Solution: Light Node Sampling + Merkle Trees



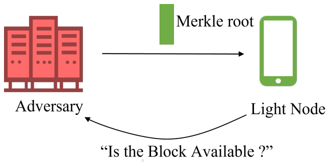
- Request/sample few random chunks of the block
- Use Merkle trees to ensure the integrity of returned chunks



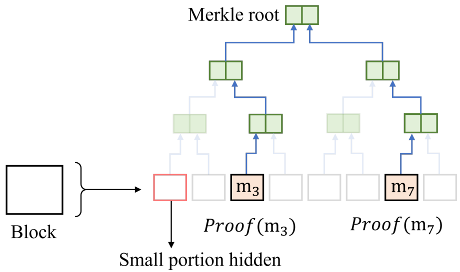
- Adversary can hide a small portion

Probability of failure using 2 random samples:

Solution: Light Node Sampling + Merkle Trees



- Request/sample few random chunks of the block
- Use Merkle trees to ensure the integrity of returned chunks

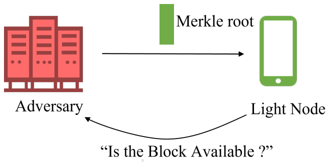


- Adversary can hide a small portion

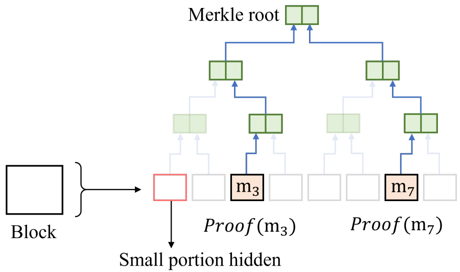
Probability of failure using 2 random samples:

$$\left(1-\frac{1}{8}\right)\left(1-\frac{1}{7}\right) = 0.75$$

Solution: Light Node Sampling + Merkle Trees



- Request/sample few random chunks of the block
- Use Merkle trees to ensure the integrity of returned chunks



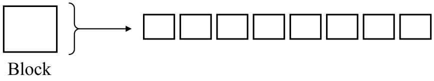
- Adversary can hide a small portion

Probability of failure using 2 random samples:

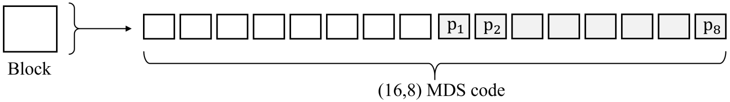
$$\left(1 - \frac{1}{8}\right)\left(1 - \frac{1}{7}\right) = 0.75$$

Erasure coding is used to improve the probability of failure

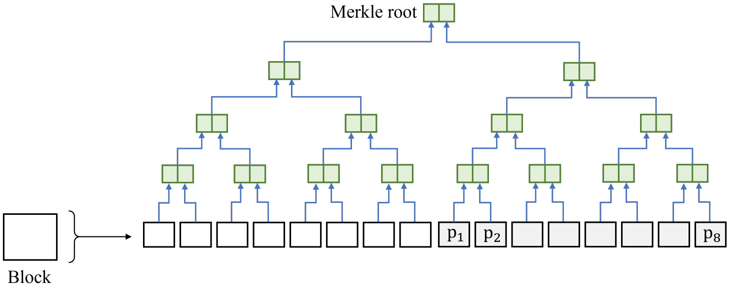
Erasure coding to Improve the Probability of Failure



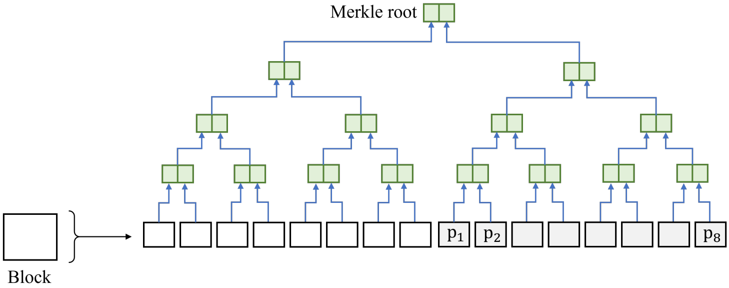
Erasure coding to Improve the Probability of Failure



Erasure coding to Improve the Probability of Failure

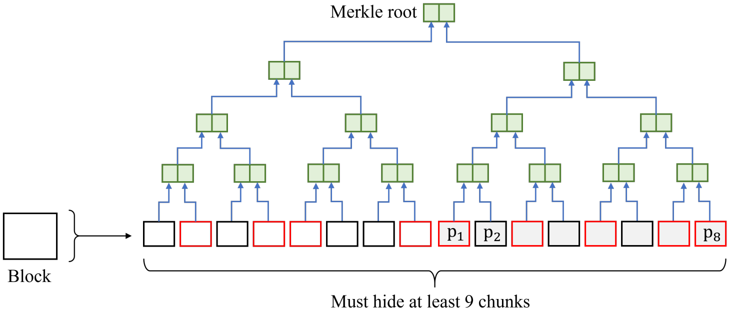


Erasure coding to Improve the Probability of Failure



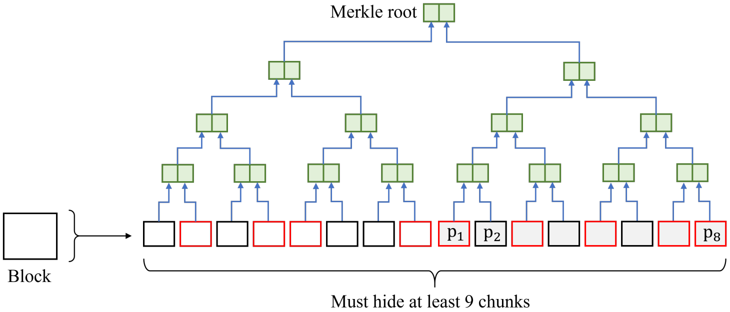
- Adversary must hide more coded chunks

Erasure coding to Improve the Probability of Failure



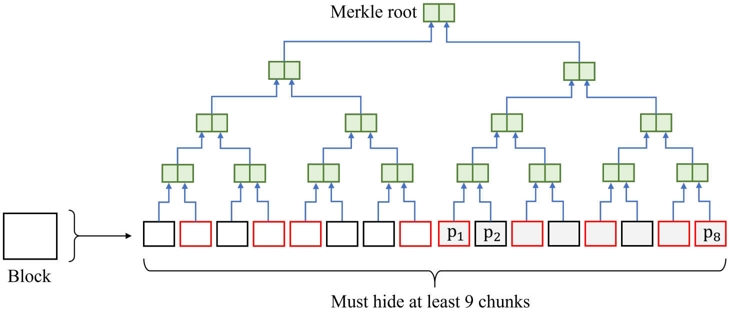
- Adversary must hide more coded chunks

Erasure coding to Improve the Probability of Failure



- Adversary must hide more coded chunks $\rightarrow$ easier for light nodes to catch using random sampling

Erasure coding to Improve the Probability of Failure

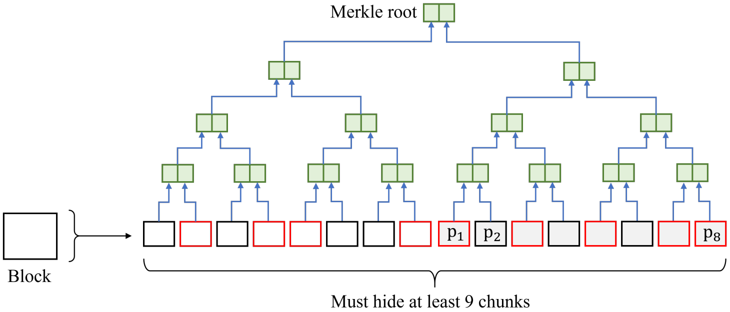


- Adversary must hide more coded chunks $\rightarrow$ easier for light nodes to catch using random sampling

Probability of failure using 2 random samples:

$$\left(1 - \frac{9}{16}\right)\left(1 - \frac{9}{15}\right) = 0.175$$

Erasure coding to Improve the Probability of Failure



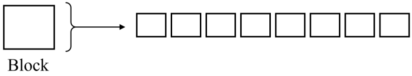
- Adversary must hide more coded chunks $\rightarrow$ easier for light nodes to catch using random sampling

Probability of failure using 2 random samples:

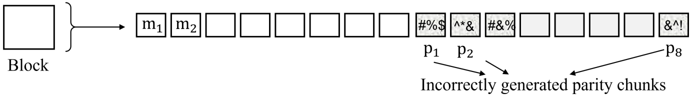
$$\left(1 - \frac{9}{16}\right)\left(1 - \frac{9}{15}\right) = 0.175$$

Adversary can incorrectly encode the block!

Incorrect-Coding (IC) Attack



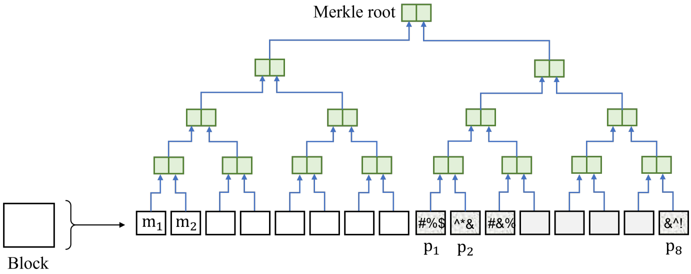
Incorrect-Coding (IC) Attack



Adversary:

- Incorrectly encodes the block

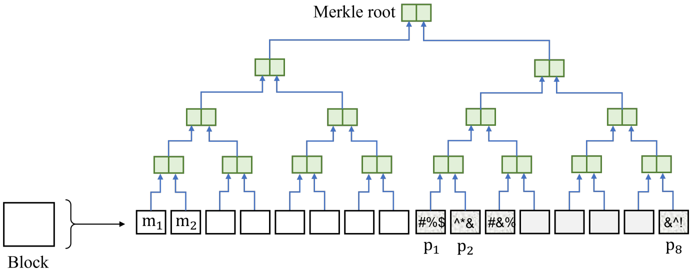
Incorrect-Coding (IC) Attack



Adversary:

- Incorrectly encodes the block

Incorrect-Coding (IC) Attack

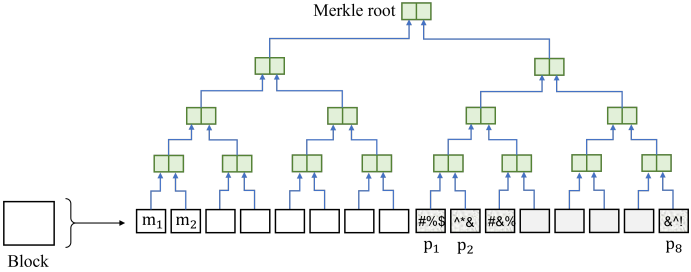


Adversary:

- Incorrectly encodes the block
- Hides less chunks since original block cannot be recovered

Incorrect-Coding (IC) Attack

Consider: $m_1 + m_2 = p_1$ (rule for correct encoding)

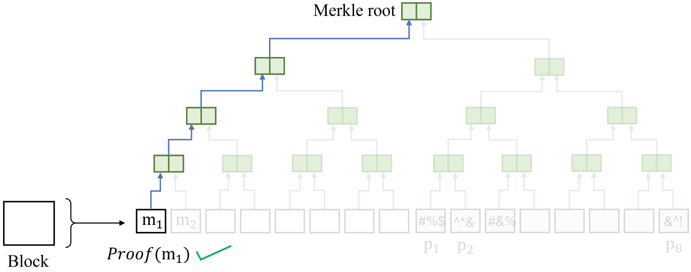


Adversary:

- Incorrectly encodes the block
- Hides less chunks since original block cannot be recovered

Incorrect-Coding (IC) Attack

Consider: $m_1 + m_2 = p_1$ (rule for correct encoding)



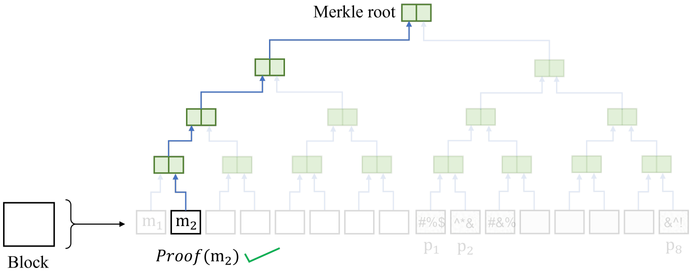
Adversary:

- Incorrectly encodes the block
- Hides less chunks since original block cannot be recovered

Honest Full node:

Incorrect-Coding (IC) Attack

Consider: $m_1 + m_2 = p_1$ (rule for correct encoding)



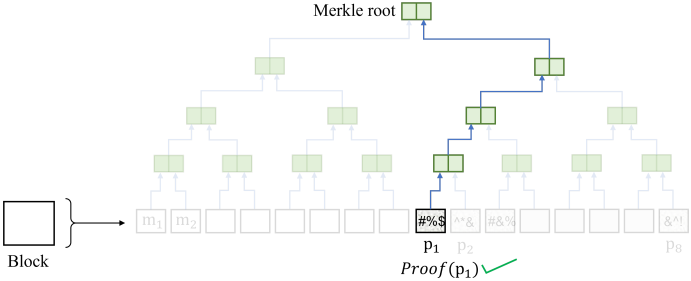
Honest Full node:

Adversary:

- Incorrectly encodes the block
- Hides less chunks since original block cannot be recovered

Incorrect-Coding (IC) Attack

Consider: $m_1 + m_2 = p_1$ (rule for correct encoding)



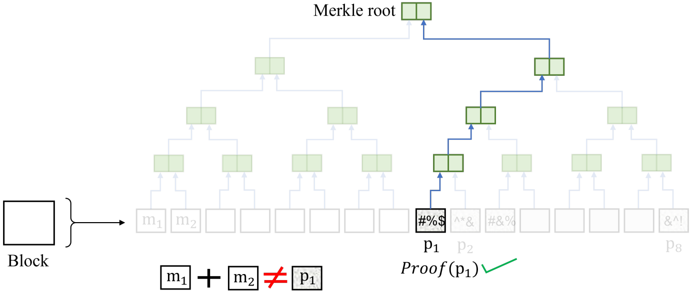
Honest Full node:

Adversary:

- Incorrectly encodes the block
- Hides less chunks since original block cannot be recovered

Incorrect-Coding (IC) Attack

Consider: $\mathrm{m}_1 + \mathrm{m}_2 = \mathrm{p}_1$ (rule for correct encoding)



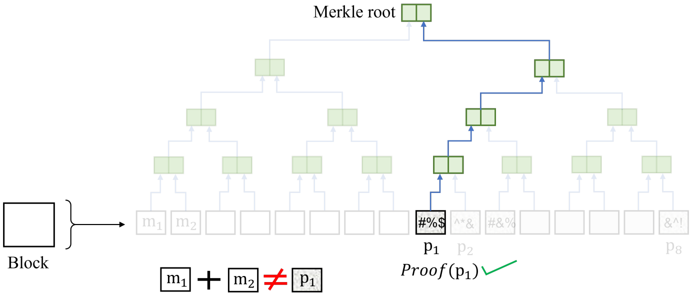
Honest Full node:

Adversary:

- Incorrectly encodes the block
- Hides less chunks since original block cannot be recovered

Incorrect-Coding (IC) Attack

Consider: $m_1 + m_2 = p_1$ (rule for correct encoding)



Adversary:

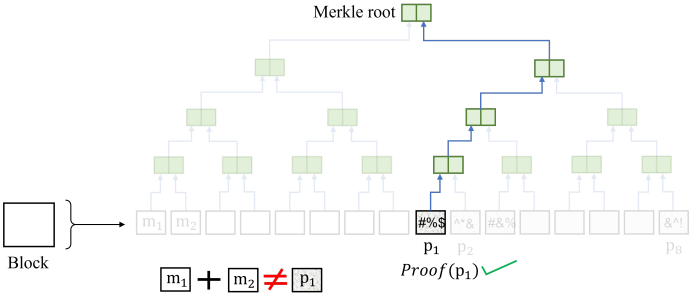
- Incorrectly encodes the block
- Hides less chunks since original block cannot be recovered

Honest Full node:

- IC-proof: m_1, m_2, p_1, $Proof(m_1)$, $Proof(m_2)$, $Proof(p_1)$

Incorrect-Coding (IC) Attack

Consider: $\mathrm{m}_1 + \mathrm{m}_2 = \mathrm{p}_1$ (rule for correct encoding)



Adversary:

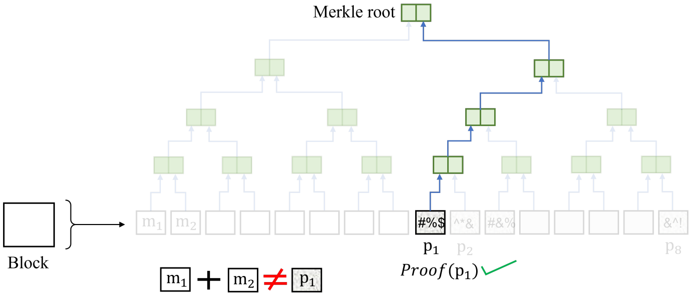
- Incorrectly encodes the block
- Hides less chunks since original block cannot be recovered

Honest Full node:

- IC-proof: m_1, m_2, p_1, $Proof(\mathrm{m}_1)$, $Proof(\mathrm{m}_2)$, $Proof(\mathrm{p}_1)$
- IC-proof size $\propto$ degree of parity check equation

Incorrect-Coding (IC) Attack

Consider: $\mathrm{m}_1 + \mathrm{m}_2 = \mathrm{p}_1$ (rule for correct encoding)



Adversary:

- Incorrectly encodes the block
- Hides less chunks since original block cannot be recovered

Honest Full node:

- IC-proof: m_1, m_2, p_1, $Proof(\mathrm{m}_1)$, $Proof(\mathrm{m}_2)$, $Proof(\mathrm{p}_1)$
- IC-proof size $\propto$ degree of parity check equation

IC-Proof size- 1D-RS: $O(b)$, 2D-RS [Al-Bassam '18] [Santini '22]: $O(\sqrt{b})$

Choice of Code

Important performance metrics for this application:

Choice of Code

Important performance metrics for this application:

1. IC-proof size: must be small in comparison to the block size

Choice of Code

Important performance metrics for this application:

1. IC-proof size: must be small in comparison to the block size
2. Undecodable threshold $\alpha_{\min}$

Choice of Code

Important performance metrics for this application:

1. IC-proof size: must be small in comparison to the block size
2. Undecodable threshold $\alpha_{\min}$
    - minimum number of coded symbols the adversary must hide to prevent decoding

Choice of Code

Important performance metrics for this application:

1. IC-proof size: must be small in comparison to the block size
2. Undecodable threshold $\alpha_{\min}$
   - minimum number of coded symbols the adversary must hide to prevent decoding
   - Probability of failure $P_f(s) = \left(1 - \frac{\alpha_{\min}}{N}\right)^s$ [per light node]

Choice of Code

Important performance metrics for this application:

1. IC-proof size: must be small in comparison to the block size
2. Undecodable threshold $\alpha_{\min}$
   - minimum number of coded symbols the adversary must hide to prevent decoding
   - Probability of failure $P_f(s) = \left(1 - \frac{\alpha_{\min}}{N}\right)^s$ [per light node]
3. Complexity of computing $\alpha_{\min}$

Choice of Code

Important performance metrics for this application:

1. IC-proof size: must be small in comparison to the block size
2. Undecodable threshold $\alpha_{\min}$
   - minimum number of coded symbols the adversary must hide to prevent decoding
   - Probability of failure $P_f(s) = \left(1 - \frac{\alpha_{\min}}{N}\right)^s$ [per light node]
3. Complexity of computing $\alpha_{\min}$
   - Important at large code length N

Choice of Code

Important performance metrics for this application:

1. IC-proof size: must be small in comparison to the block size
2. Undecodable threshold $\alpha_{\min}$
   - minimum number of coded symbols the adversary must hide to prevent decoding
   - Probability of failure $P_f(s) = \left(1 - \frac{\alpha_{\min}}{N}\right)^s$ [per light node]
3. Complexity of computing $\alpha_{\min}$
   - Important at large code length N
4. Decoding complexity

Choice of Code

Important performance metrics for this application:

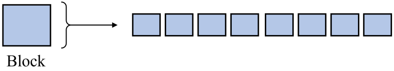
1. IC-proof size: must be small in comparison to the block size
2. Undecodable threshold $\alpha_{\min}$
   - minimum number of coded symbols the adversary must hide to prevent decoding
   - Probability of failure $P_f(s) = \left(1 - \frac{\alpha_{\min}}{N}\right)^s$ [per light node]
3. Complexity of computing $\alpha_{\min}$
   - Important at large code length N
4. Decoding complexity

Our work: A novel construction of Merkle trees using polar codes that performs well on all the above metrics for large transaction block sizes.

Coded Merkle Tree (CMT) [Yu '19]

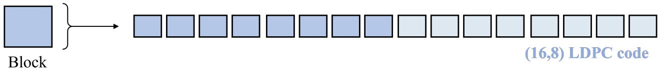
- Uses Low-Density Parity-Check (LDPC) code to encode each layer of the Merkle Tree

Coded Merkle Tree (CMT) [Yu '19]



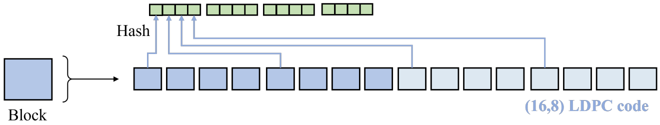
- Uses Low-Density Parity-Check (LDPC) code to encode each layer of the Merkle Tree

Coded Merkle Tree (CMT) [Yu '19]



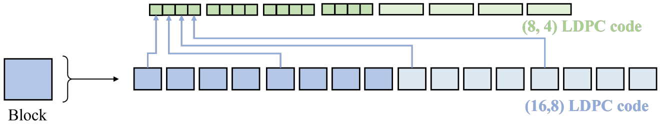
- Uses Low-Density Parity-Check (LDPC) code to encode each layer of the Merkle Tree

Coded Merkle Tree (CMT) [Yu '19]



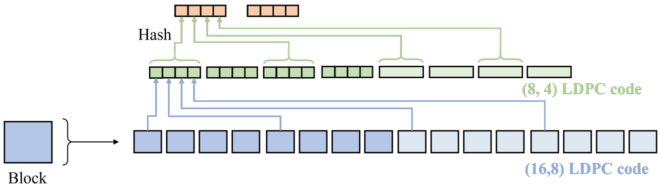
- Uses Low-Density Parity-Check (LDPC) code to encode each layer of the Merkle Tree

Coded Merkle Tree (CMT) [Yu '19]



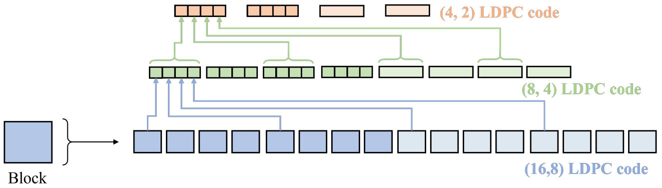
- Uses Low-Density Parity-Check (LDPC) code to encode each layer of the Merkle Tree

Coded Merkle Tree (CMT) [Yu '19]



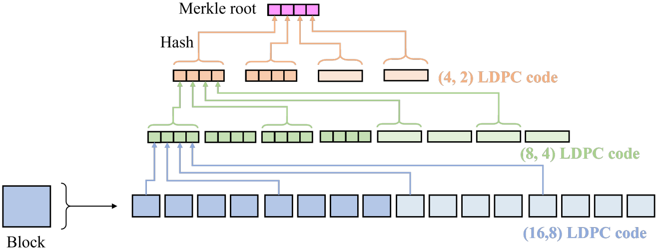
- Uses Low-Density Parity-Check (LDPC) code to encode each layer of the Merkle Tree

Coded Merkle Tree (CMT) [Yu '19]



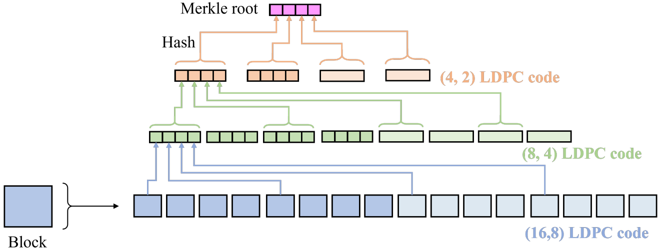
- Uses Low-Density Parity-Check (LDPC) code to encode each layer of the Merkle Tree

Coded Merkle Tree (CMT) [Yu '19]



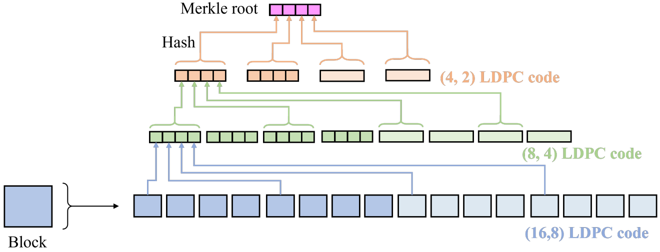
- Uses Low-Density Parity-Check (LDPC) code to encode each layer of the Merkle Tree

Coded Merkle Tree (CMT) [Yu '19]



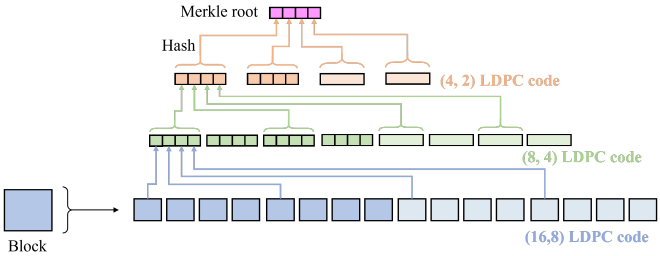
- Uses Low-Density Parity-Check (LDPC) code to encode each layer of the Merkle Tree → Detects DA attacks on any layer of CMT

Coded Merkle Tree (CMT) [Yu '19]



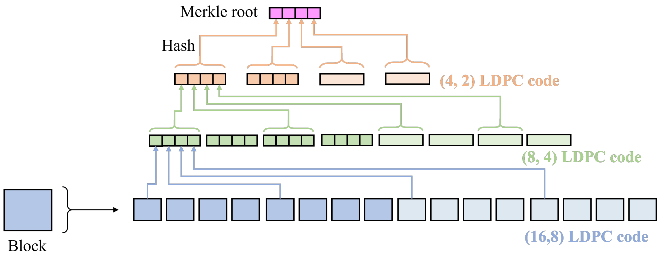
- Uses Low-Density Parity-Check (LDPC) code to encode each layer of the Merkle Tree → Detects DA attacks on any layer of CMT
  Performance:
  1. IC-proof size: small due to sparse parity check equations

Coded Merkle Tree (CMT) [Yu '19]



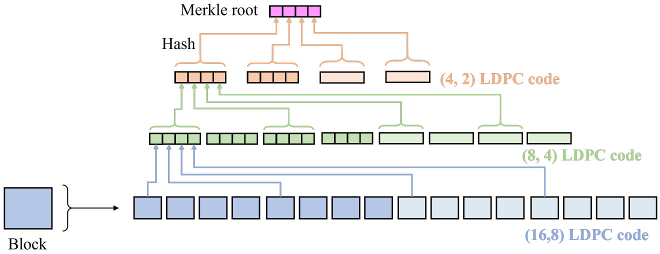
- Uses Low-Density Parity-Check (LDPC) code to encode each layer of the Merkle Tree → Detects DA attacks on any layer of CMT
  Performance:
  1. IC-proof size: small due to sparse parity check equations
  2. Decoding complexity: linear in code length using a peeling decoder

Coded Merkle Tree (CMT) [Yu '19]



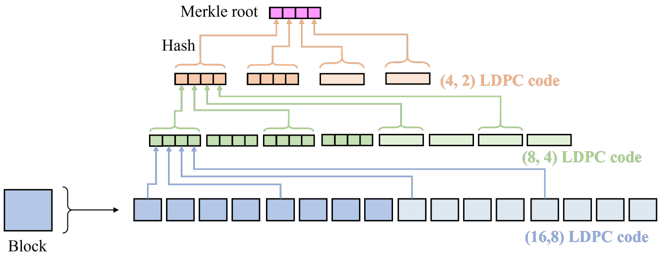
- Uses Low-Density Parity-Check (LDPC) code to encode each layer of the Merkle Tree → Detects DA attacks on any layer of CMT
  Performance:
  1. IC-proof size: small due to sparse parity check equations
  2. Decoding complexity: linear in code length using a peeling decoder
  3. What about undecodable threshold $\alpha_{\min}$ and complexity of computing $\alpha_{\min}$?

Coded Merkle Tree (CMT) [Yu '19]



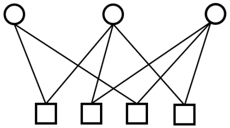
Challenge with LDPC codes: Stopping sets

Coded Merkle Tree (CMT) [Yu '19]

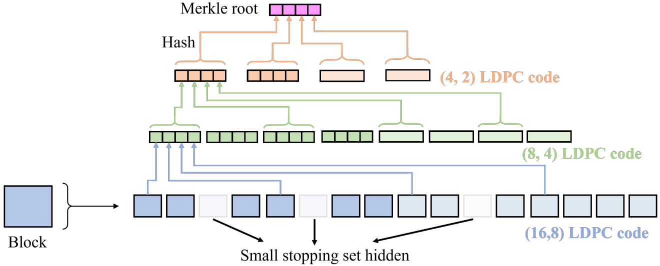


Challenge with LDPC codes: Stopping sets

- Substructure in the Tanner Graph

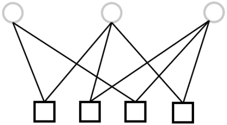


Coded Merkle Tree (CMT) [Yu '19]

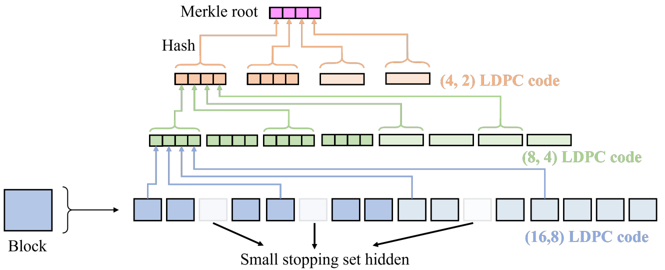


Challenge with LDPC codes: Stopping sets

- Substructure in the Tanner Graph
- If hidden, prevents peeling decoder from decoding the block

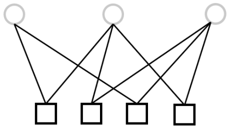


Coded Merkle Tree (CMT) [Yu '19]

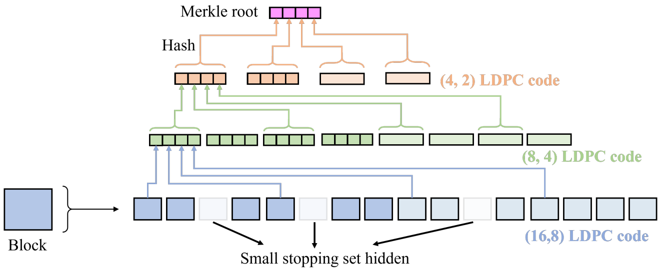


Challenge with LDPC codes: Stopping sets

- Substructure in the Tanner Graph
- If hidden, prevents peeling decoder from decoding the block
- Undecodable threshold $\alpha_{\min}$ = size of smallest stopping set

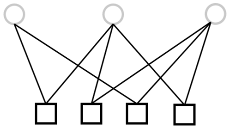


Coded Merkle Tree (CMT) [Yu '19]

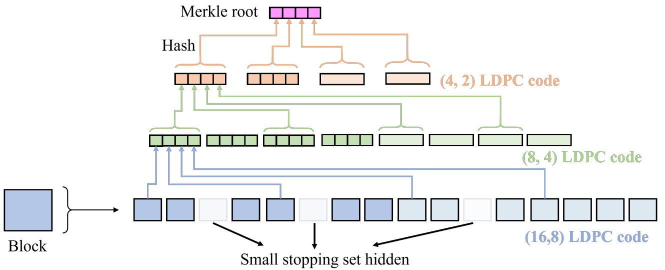


Challenge with LDPC codes: Stopping sets

- Substructure in the Tanner Graph
- If hidden, prevents peeling decoder from decoding the block
- Undecodable threshold $\alpha_{\min}$ = size of smallest stopping set → NP-hard to compute [Krishnan '07]

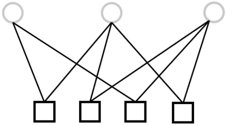


Coded Merkle Tree (CMT) [Yu '19]



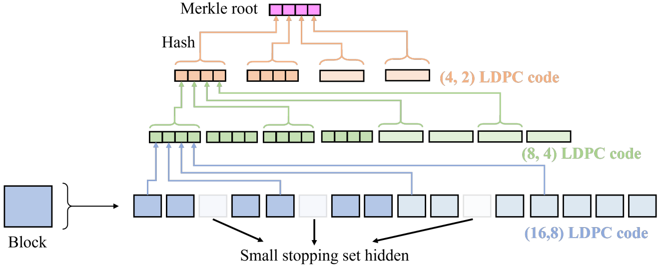
Challenge with LDPC codes: Stopping sets

- Substructure in the Tanner Graph
- If hidden, prevents peeling decoder from decoding the block
- Undecodable threshold $\alpha_{\min}$ = size of smallest stopping set → NP-hard to compute [Krishnan '07]



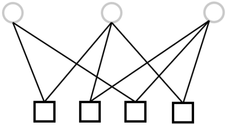
Merkle tree construction using polar codes allows for an efficient method to compute $\alpha_{\min}$

Coded Merkle Tree (CMT) [Yu '19]



Challenge with LDPC codes: Stopping sets

- Substructure in the Tanner Graph
- If hidden, prevents peeling decoder from decoding the block
- Undecodable threshold $\alpha_{\min}$ = size of smallest stopping set → NP-hard to compute [Krishnan '07]



Merkle tree construction using polar codes allows for an efficient method to compute $\alpha_{\min}$ while having small IC-proof size and decoding complexity.

Polar Coded Merkle Tree (PCMT)

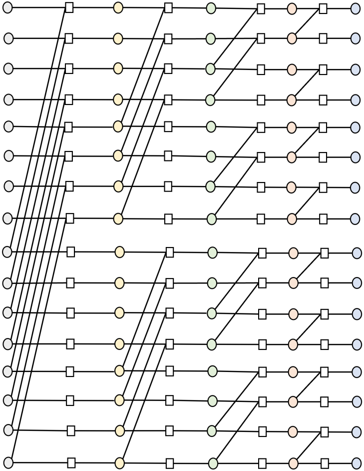
Polar codes

- Dense parity check matrices [Goela '10]

Polar Coded Merkle Tree (PCMT)

Polar codes

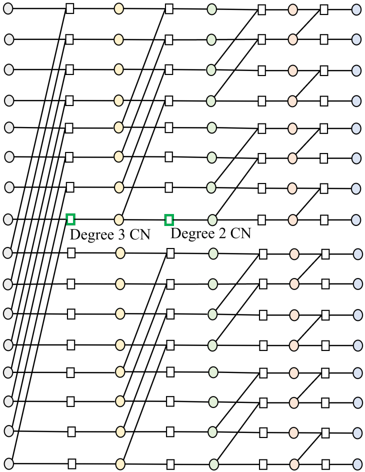
- Dense parity check matrices [Goela '10]
- Sparse encoding graph



Polar Coded Merkle Tree (PCMT)

Polar codes

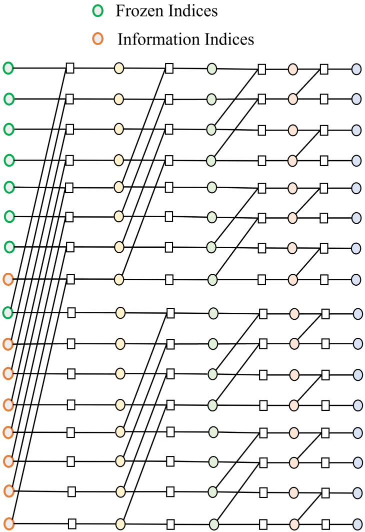
- Dense parity check matrices [Goela '10]
- Sparse encoding graph



Polar Coded Merkle Tree (PCMT)

Polar codes

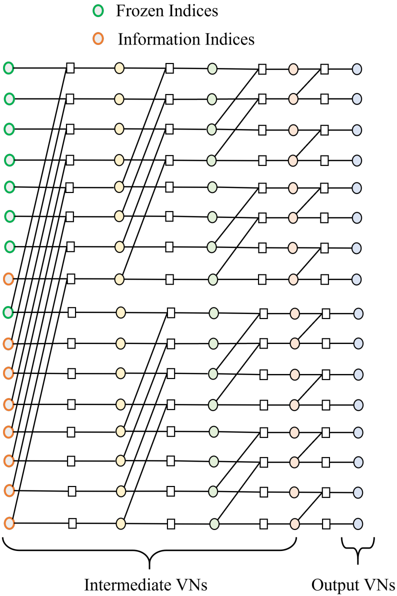
- Dense parity check matrices [Goela '10]
- Sparse encoding graph



Polar Coded Merkle Tree (PCMT)

Polar codes

- Dense parity check matrices [Goela '10]
- Sparse encoding graph
- Intermediate VNs in addition to output VNs

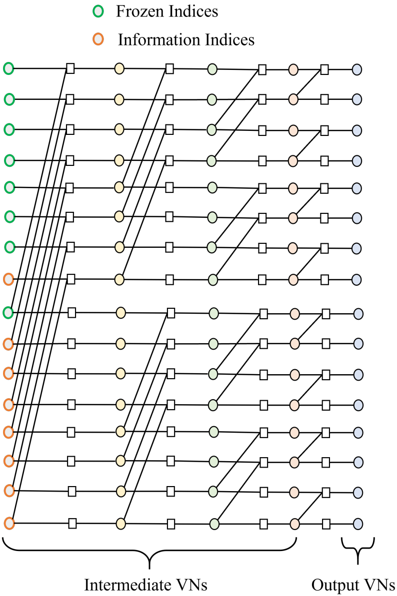


Polar Coded Merkle Tree (PCMT)

Polar codes

- Dense parity check matrices [Goela '10]
- Sparse encoding graph
- Intermediate VNs in addition to output VNs

PCMT

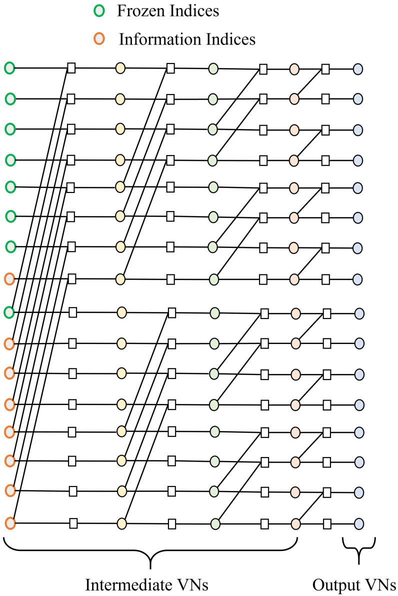


Polar Coded Merkle Tree (PCMT)

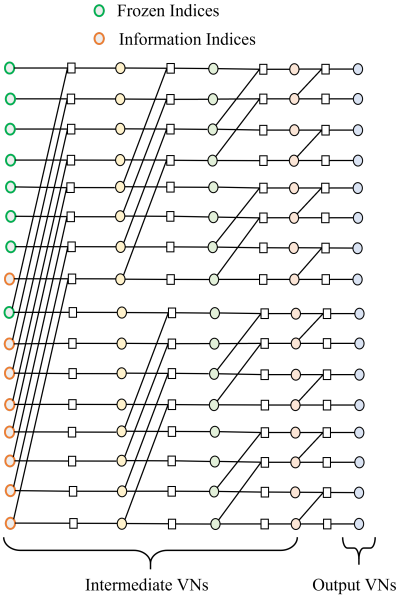
Polar codes

- Dense parity check matrices [Goela '10]
- Sparse encoding graph
- Intermediate VNs in addition to output VNs

PCMT
- store the hashes of the intermediate VNs



Polar Coded Merkle Tree (PCMT)

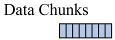


Polar codes

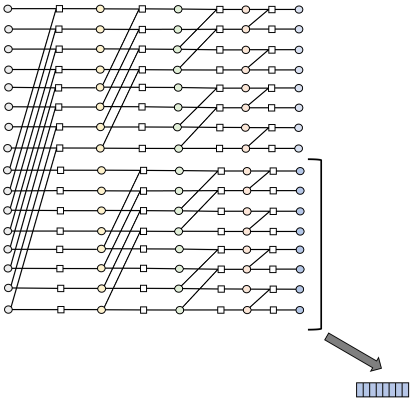
- Dense parity check matrices [Goela '10]
- Sparse encoding graph
- Intermediate VNs in addition to output VNs

PCMT
- store the hashes of the intermediate VNs
- use these hashes to build small IC-proofs for the degree 2 and degree 3 CNs

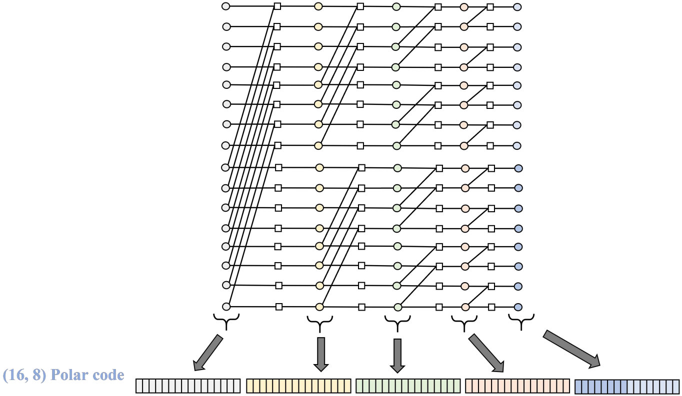
PCMT Construction



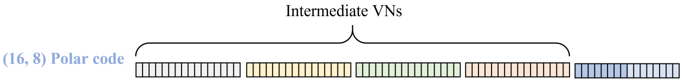
PCMT Construction



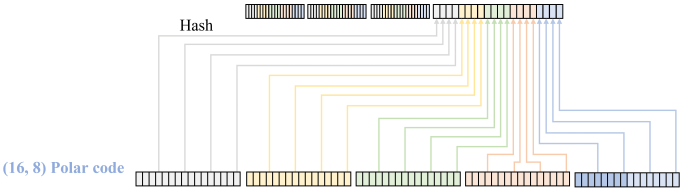
PCMT Construction



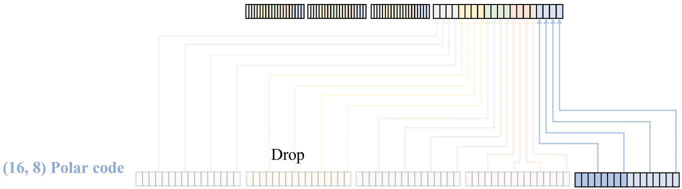
PCMT Construction



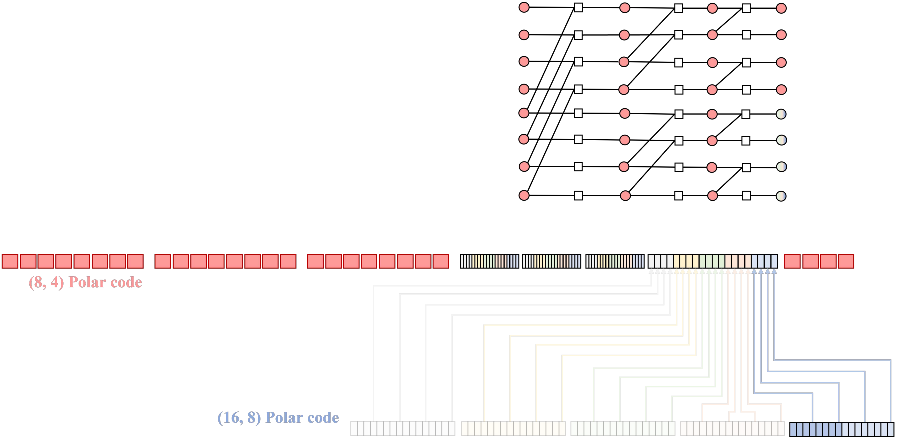
PCMT Construction



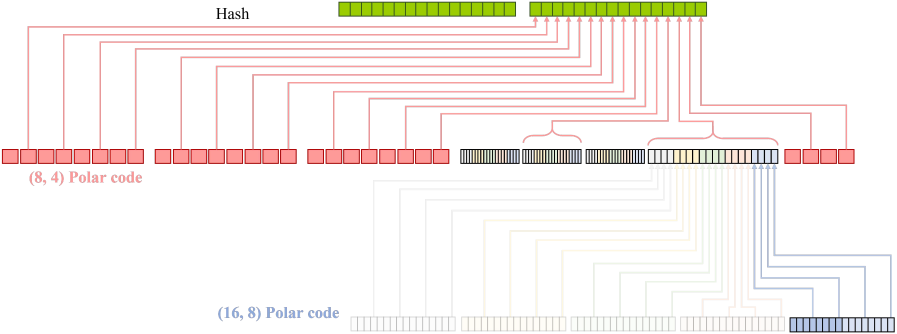
PCMT Construction



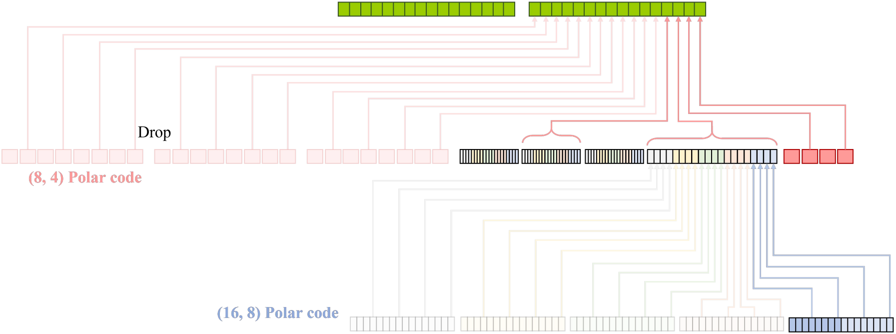
PCMT Construction



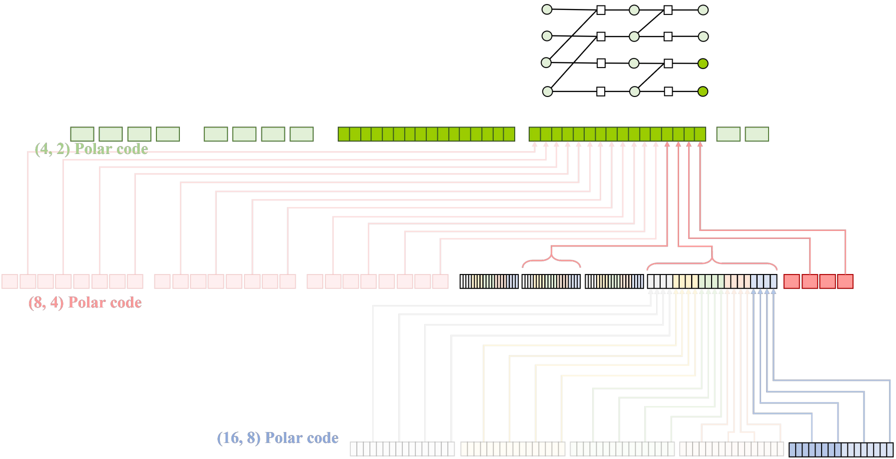
PCMT Construction



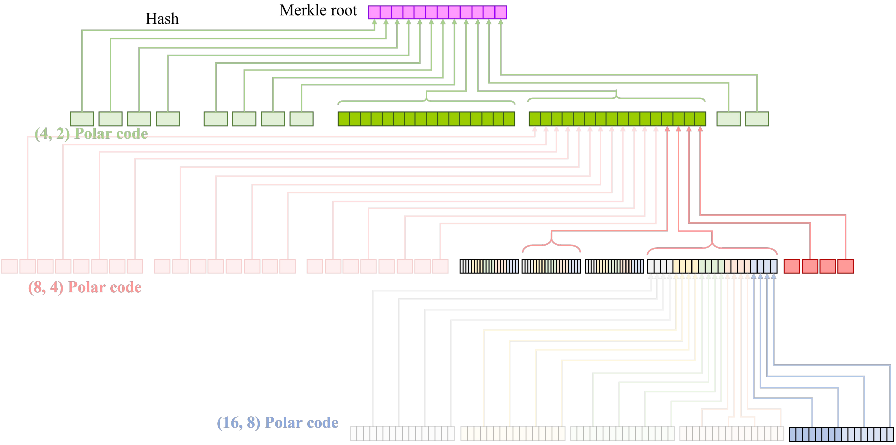
PCMT Construction



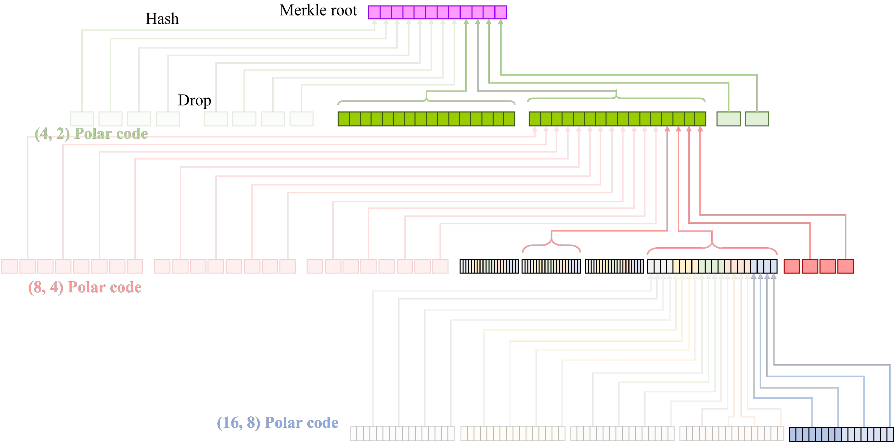
PCMT Construction



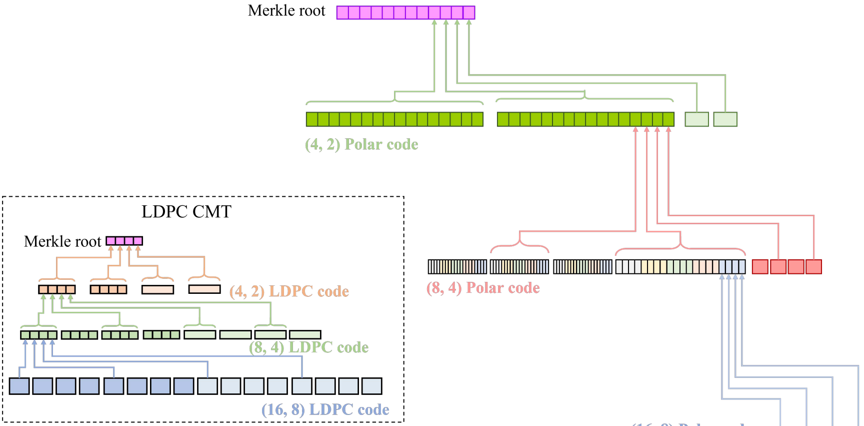
PCMT Construction



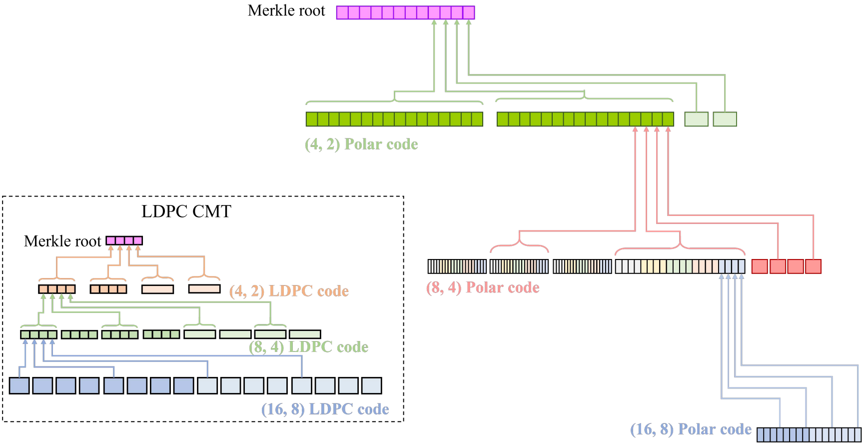
PCMT Construction



PCMT Construction

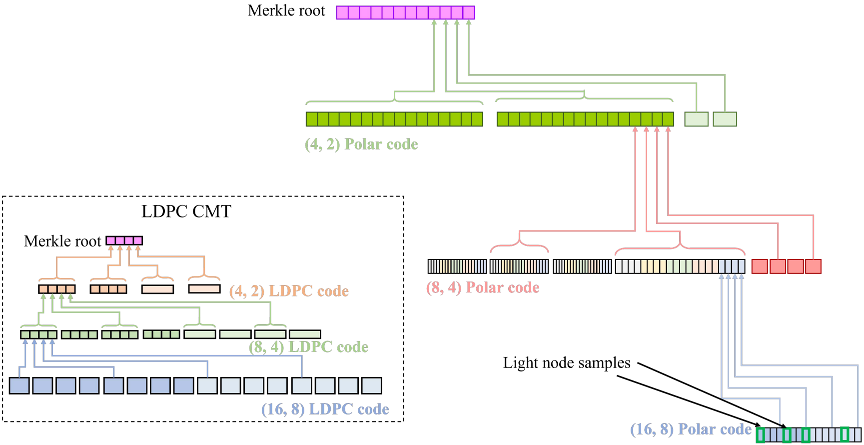


PCMT Construction



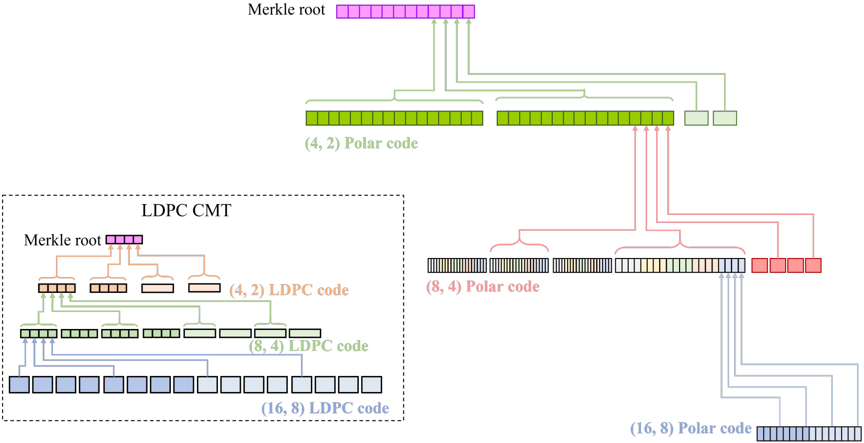
- Dropped VNs can be decoded back using a peeling decoder

PCMT Construction



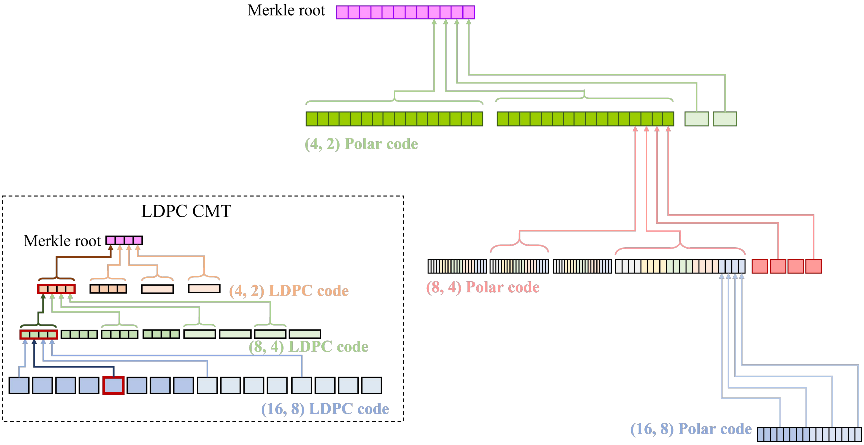
- Dropped VNs can be decoded back using a peeling decoder
- Light nodes sample the non-dropped VNs

PCMT: Merkle Proofs



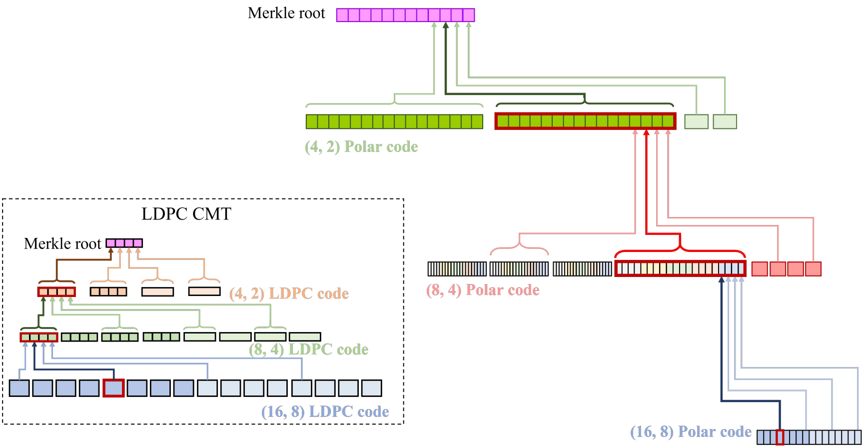
- Both dropped and non-dropped VNs have merkle proofs

PCMT: Merkle Proofs



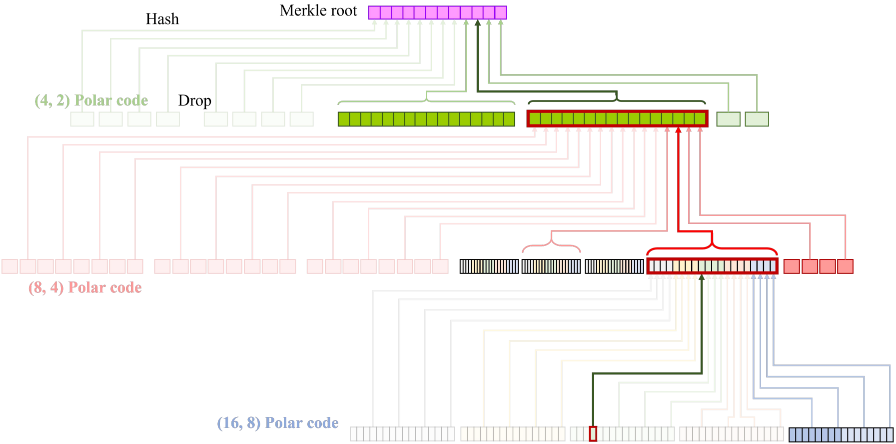
- Both dropped and non-dropped VNs have merkle proofs

PCMT: Merkle Proofs



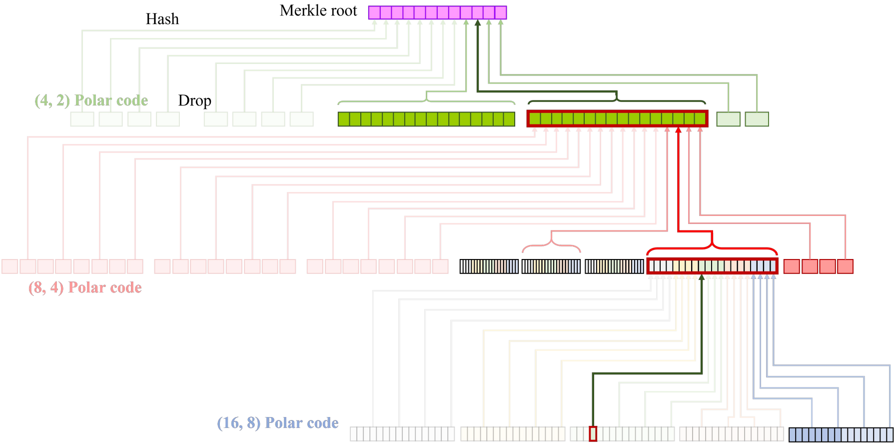
- Both dropped and non-dropped VNs have merkle proofs

PCMT: Merkle Proofs



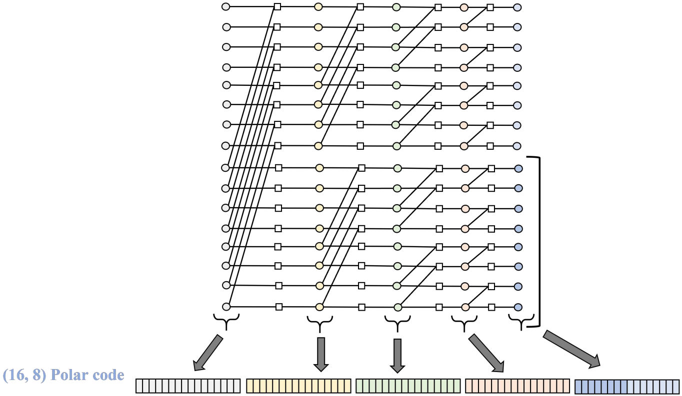
- Both dropped and non-dropped VNs have merkle proofs

PCMT: Merkle Proofs

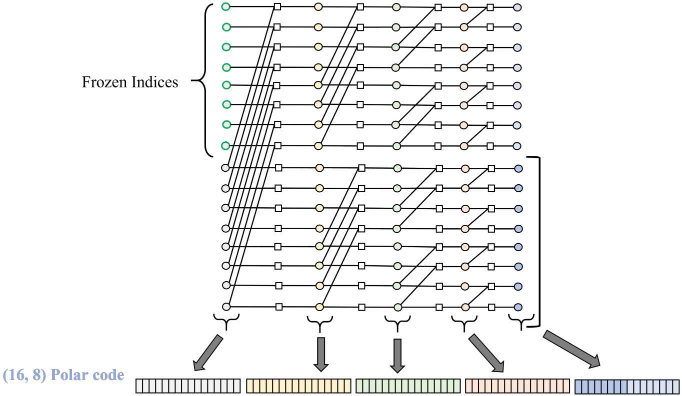


- Both dropped and non-dropped VNs have merkle proofs
- Used for integrity checks and in IC-proofs similar to LDPC CMT

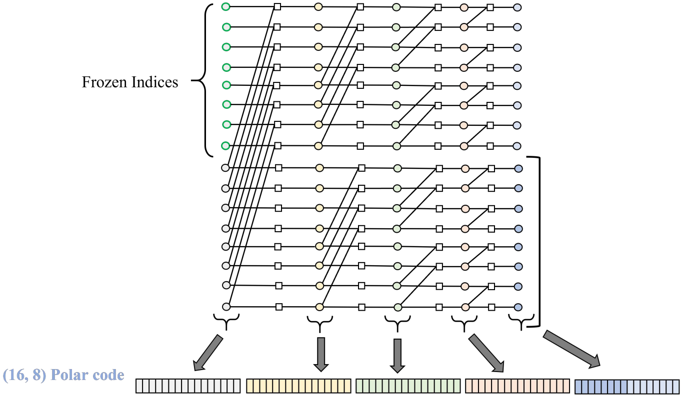
Frozen Index Selection for PCMT



Frozen Index Selection for PCMT

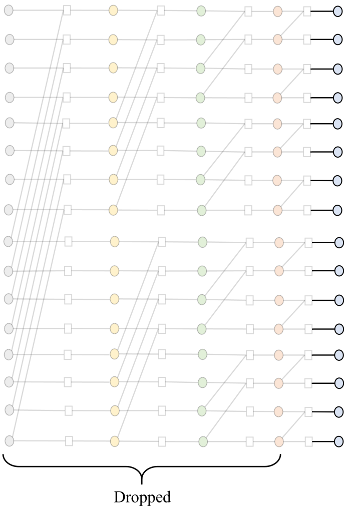


Frozen Index Selection for PCMT

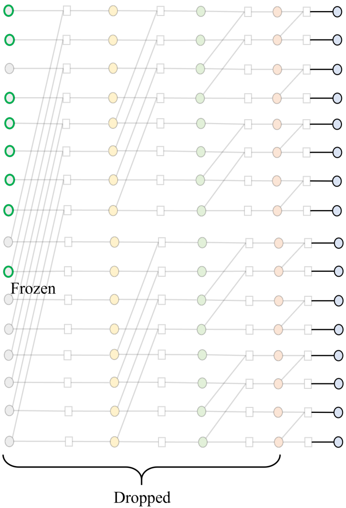


- Not the best choice for frozen indices

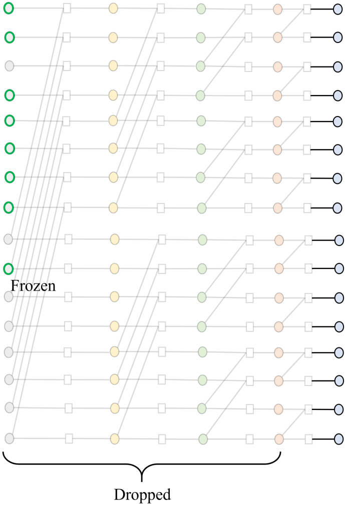
Frozen Index Selection for PCMT



Frozen Index Selection for PCMT



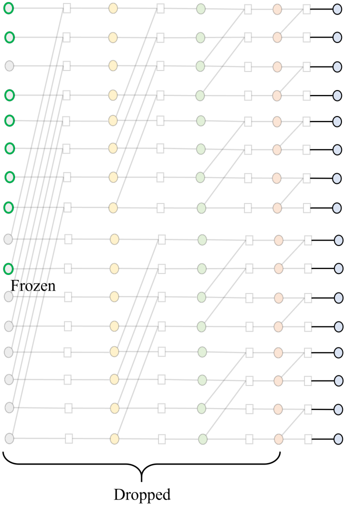
Frozen Index Selection for PCMT



Adversary:

- Cannot hide frozen VNs

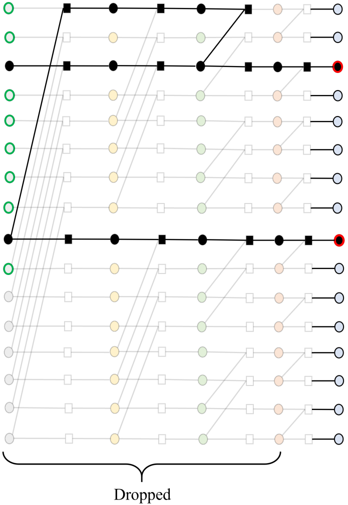
Frozen Index Selection for PCMT



Adversary:

- Cannot hide frozen VNs
- Must hide non-dropped VNs such that a stopping set becomes unavailable

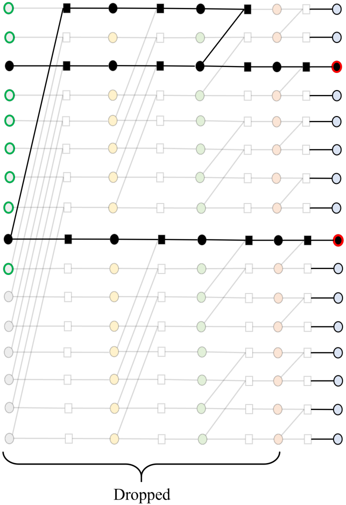
Frozen Index Selection for PCMT



Adversary:

- Cannot hide frozen VNs
- Must hide non-dropped VNs such that a stopping set becomes unavailable

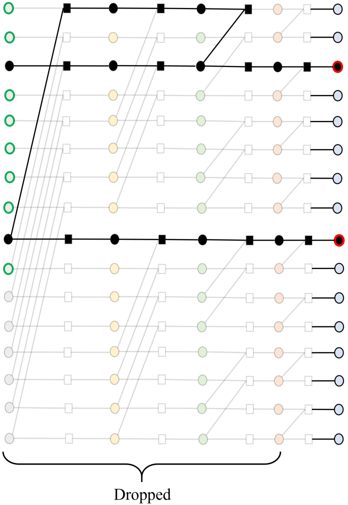
Frozen Index Selection for PCMT



Adversary:

- Cannot hide frozen VNs
- Must hide non-dropped VNs such that a stopping set becomes unavailable
- Hide the leaf set of a stopping set with no frozen VNs

Frozen Index Selection for PCMT

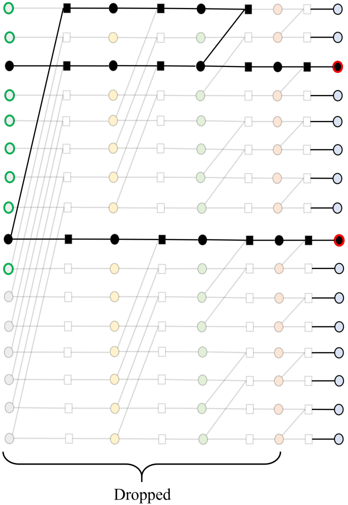


Adversary:

- Cannot hide frozen VNs
- Must hide non-dropped VNs such that a stopping set becomes unavailable
- Hide the leaf set of a stopping set with no frozen VNs

Undecodable threshold $\alpha_{\min}$

Frozen Index Selection for PCMT

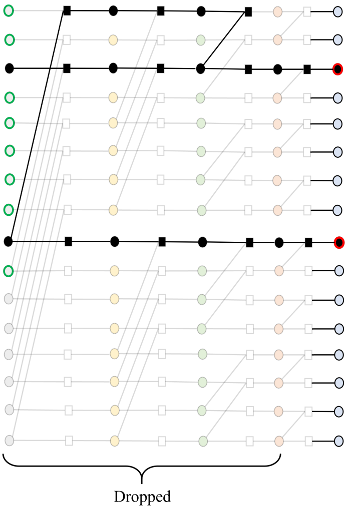


Adversary:

- Cannot hide frozen VNs
- Must hide non-dropped VNs such that a stopping set becomes unavailable
- Hide the leaf set of a stopping set with no frozen VNs

Undecodable threshold $\alpha_{\min}$ $=$ smallest leaf set size of all stopping sets with no frozen VNs

Frozen Index Selection for PCMT

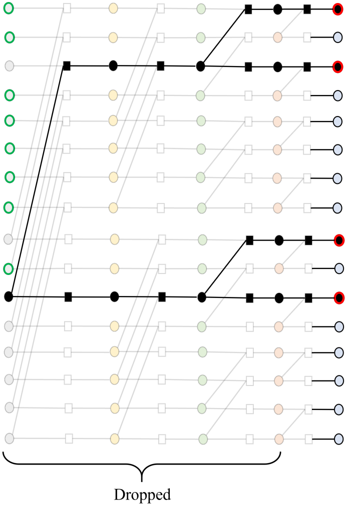


Adversary:

- Cannot hide frozen VNs
- Must hide non-dropped VNs such that a stopping set becomes unavailable
- Hide the leaf set of a stopping set with no frozen VNs

Undecodable threshold $\alpha_{\min}$
= smallest leaf set size of all stopping sets with no frozen VNs
= smallest leaf set size of all stopping trees with no frozen VNs [Eslami '13]

Frozen Index Selection for PCMT

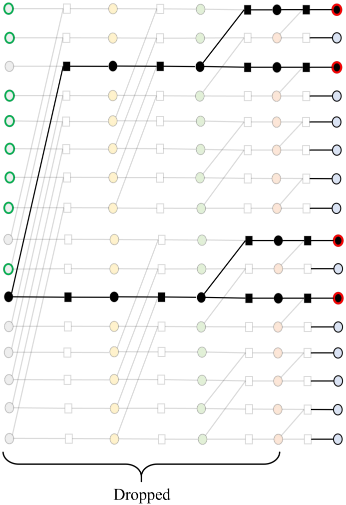


Adversary:

- Cannot hide frozen VNs
- Must hide non-dropped VNs such that a stopping set becomes unavailable
- Hide the leaf set of a stopping set with no frozen VNs

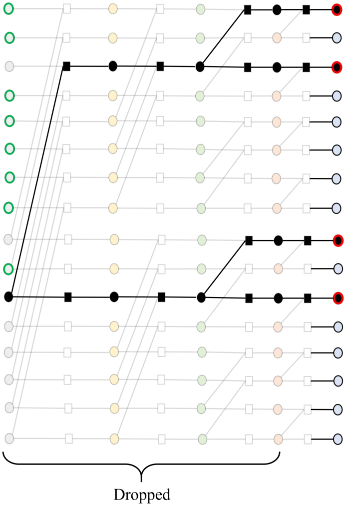
Undecodable threshold $\alpha_{\min}$
= smallest leaf set size of all stopping sets with no frozen VNs
= smallest leaf set size of all stopping trees with no frozen VNs [Eslami '13]

Frozen Index Selection for PCMT



Stopping Trees:

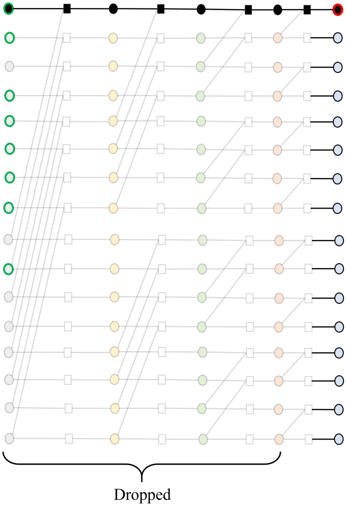
Frozen Index Selection for PCMT



Stopping Trees:

- Every VN in the leftmost column is associated with a unique stopping tree

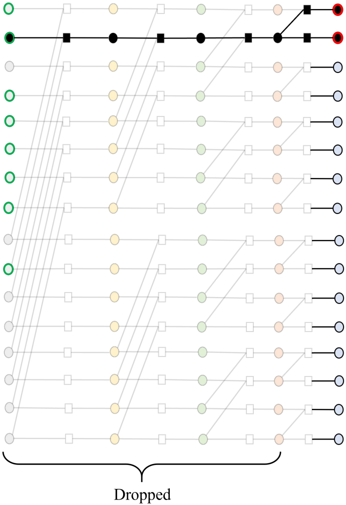
Frozen Index Selection for PCMT



Stopping Trees:

- Every VN in the leftmost column is associated with a unique stopping tree

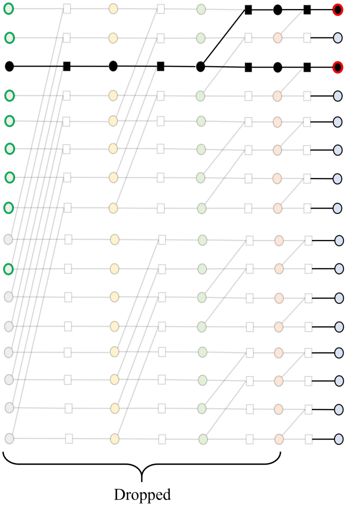
Frozen Index Selection for PCMT



Stopping Trees:

- Every VN in the leftmost column is associated with a unique stopping tree

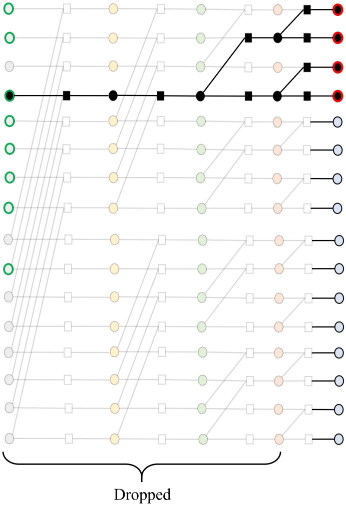
Frozen Index Selection for PCMT



Stopping Trees:

- Every VN in the leftmost column is associated with a unique stopping tree

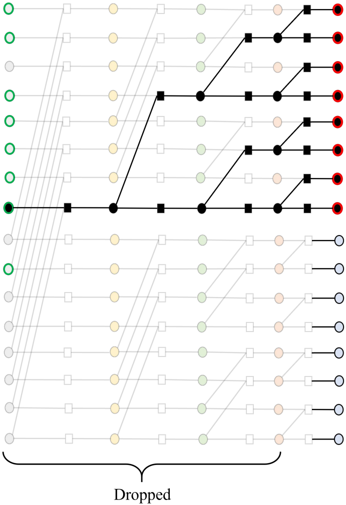
Frozen Index Selection for PCMT



Stopping Trees:

- Every VN in the leftmost column is associated with a unique stopping tree

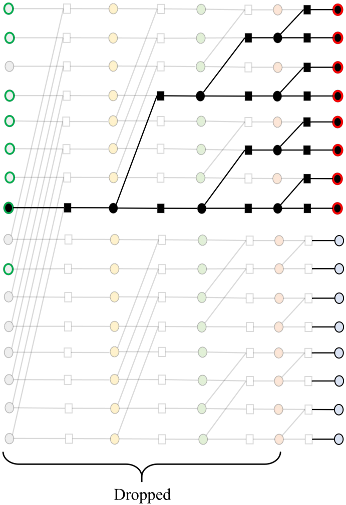
Frozen Index Selection for PCMT



Stopping Trees:

- Every VN in the leftmost column is associated with a unique stopping tree

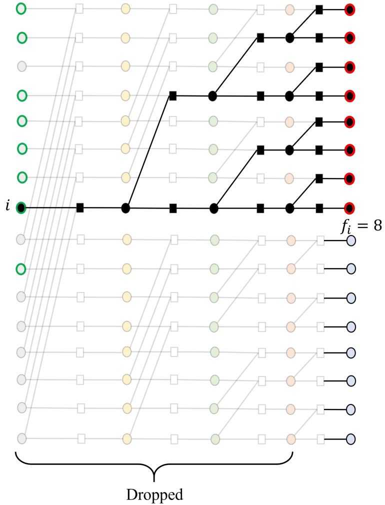
Frozen Index Selection for PCMT



Stopping Trees:

- Every VN in the leftmost column is associated with a unique stopping tree
- f_i = leaf set size of stopping tree associated with ith VN

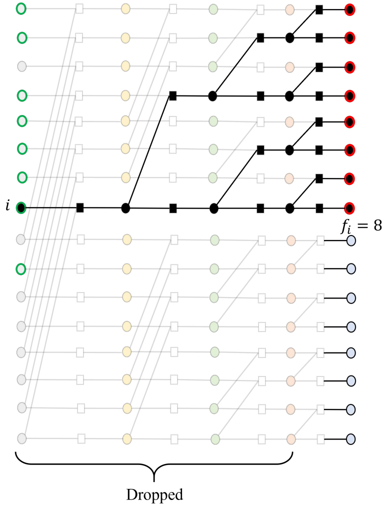
Frozen Index Selection for PCMT



Stopping Trees:

- Every VN in the leftmost column is associated with a unique stopping tree
- f_i = leaf set size of stopping tree associated with ith VN

Frozen Index Selection for PCMT

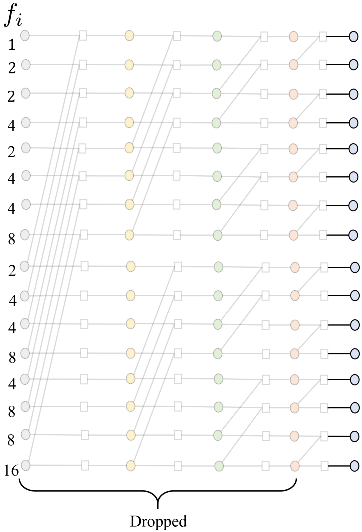


Stopping Trees:

- Every VN in the leftmost column is associated with a unique stopping tree
- f_i = leaf set size of stopping tree associated with ith VN

$$\alpha_{\min} = \min_{i \text{ not frozen}} f_i$$

Frozen Index Selection for PCMT

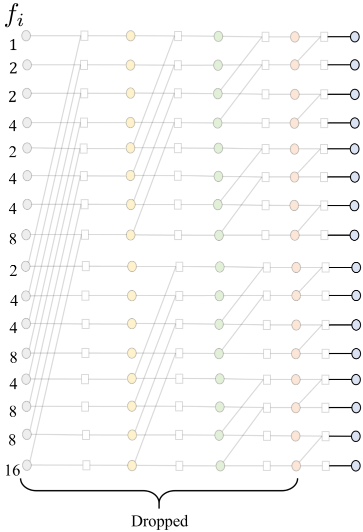


Stopping Trees:

- Every VN in the leftmost column is associated with a unique stopping tree
- f_i = leaf set size of stopping tree associated with ith VN

$$\alpha_{\min} = \min_{i \text{ not frozen}} f_i$$

Frozen Index Selection for PCMT



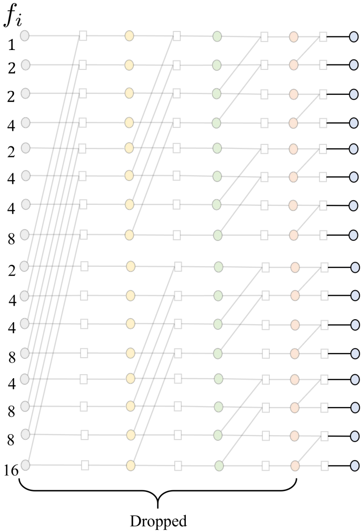
Stopping Trees:

- Every VN in the leftmost column is associated with a unique stopping tree
- f_i = leaf set size of stopping tree associated with ith VN

$$\alpha_{\min} = \min_{i \text{ not frozen}} f_i$$

Naive selection method

Frozen Index Selection for PCMT



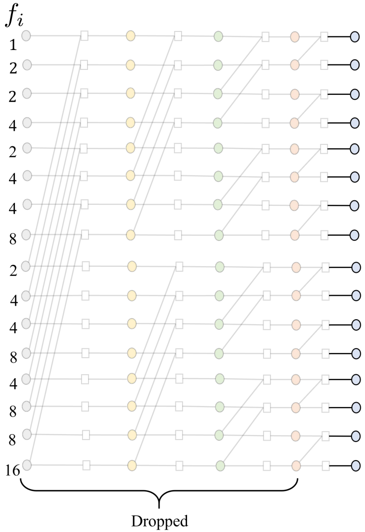
Stopping Trees:

- Every VN in the leftmost column is associated with a unique stopping tree
- f_i = leaf set size of stopping tree associated with ith VN

$$\alpha_{\min} = \min_{i \text{ not frozen}} f_i$$

Naive selection method
- For (N, k) polar code, select the indices with $N - k$ smallest leaf set sizes as frozen set.

Frozen Index Selection for PCMT



Stopping Trees:

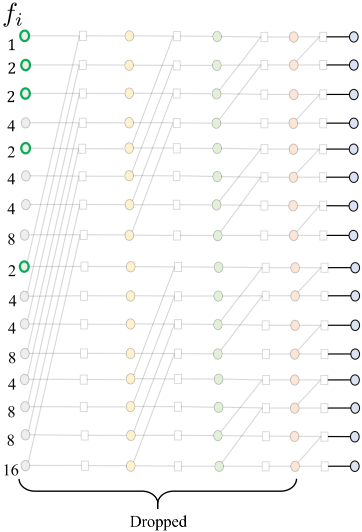
- Every VN in the leftmost column is associated with a unique stopping tree
- f_i = leaf set size of stopping tree associated with ith VN

$$\alpha_{\min} = \min_{i \text{ not frozen}} f_i$$

> Naive selection method
> - For (N, k) polar code, select the indices with $N - k$ smallest leaf set sizes as frozen set.

E.g. $(16, 8)$ polar code:

Frozen Index Selection for PCMT



Stopping Trees:

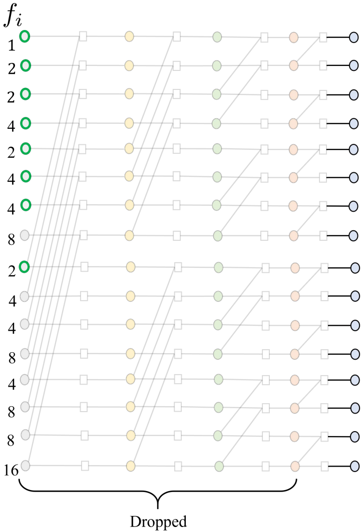
- Every VN in the leftmost column is associated with a unique stopping tree
- f_i = leaf set size of stopping tree associated with ith VN

$$\alpha_{\min} = \min_{i \text{ not frozen}} f_i$$

> Naive selection method
> - For (N, k) polar code, select the indices with $N - k$ smallest leaf set sizes as frozen set.

E.g. $(16, 8)$ polar code:

Frozen Index Selection for PCMT



Stopping Trees:

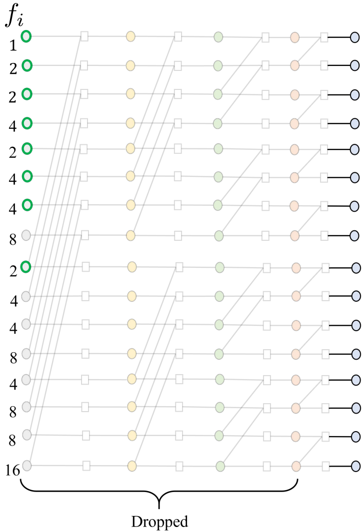
- Every VN in the leftmost column is associated with a unique stopping tree
- f_i = leaf set size of stopping tree associated with ith VN

$$\alpha_{\min} = \min_{i \text{ not frozen}} f_i$$

Naive selection method
- For (N, k) polar code, select the indices with $N - k$ smallest leaf set sizes as frozen set.

E.g. $(16, 8)$ polar code:

Frozen Index Selection for PCMT



Stopping Trees:

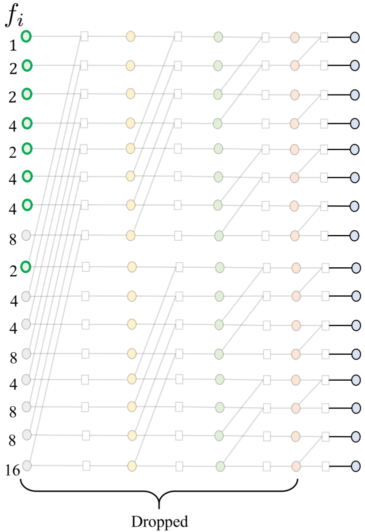
- Every VN in the leftmost column is associated with a unique stopping tree
- f_i = leaf set size of stopping tree associated with ith VN

$$\alpha_{\min} = \min_{i \text{ not frozen}} f_i$$

Naive selection method
- For (N, k) polar code, select the indices with $N - k$ smallest leaf set sizes as frozen set.

E.g. $(16, 8)$ polar code: $\alpha_{\min} = 4$.

Frozen Index Selection for PCMT



Stopping Trees:

- Every VN in the leftmost column is associated with a unique stopping tree
- f_i = leaf set size of stopping tree associated with ith VN

$$\alpha_{\min} = \min_{i \text{ not frozen}} f_i$$

Naive selection method
- For (N, k) polar code, select the indices with $N - k$ smallest leaf set sizes as frozen set.

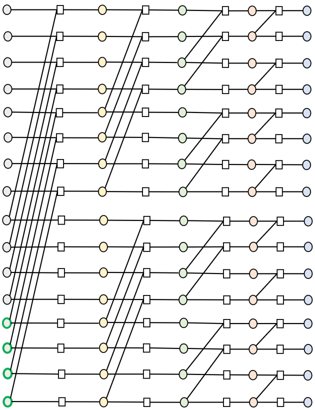
E.g. $(16, 8)$ polar code: $\alpha_{\min} = 4$.

Can we do better ?

Sampling Efficient Freezing (SEF) Algorithm

Lemma

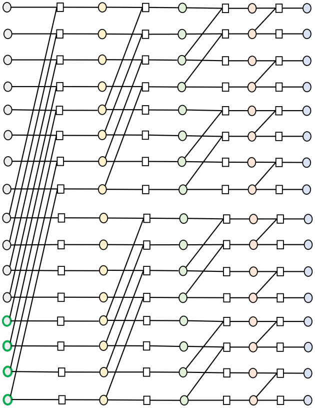
If we freeze last μ indices from the bottom of the encoding graph,



Sampling Efficient Freezing (SEF) Algorithm

Lemma

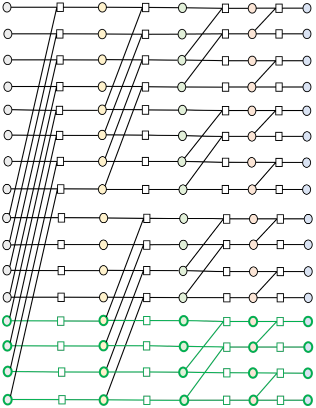
If we freeze last μ indices from the bottom of the encoding graph, then a stopping set with no frozen VNs cannot have a VN from the last μ rows of the encoding graph.



Sampling Efficient Freezing (SEF) Algorithm

Lemma

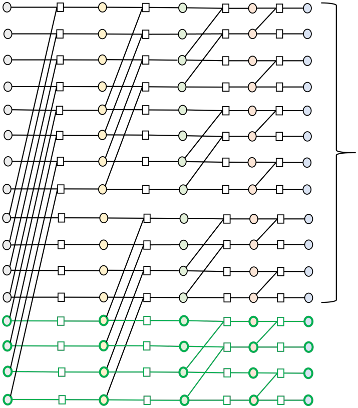
If we freeze last μ indices from the bottom of the encoding graph, then a stopping set with no frozen VNs cannot have a VN from the last μ rows of the encoding graph.



Sampling Efficient Freezing (SEF) Algorithm

Lemma

If we freeze last μ indices from the bottom of the encoding graph, then a stopping set with no frozen VNs cannot have a VN from the last μ rows of the encoding graph.

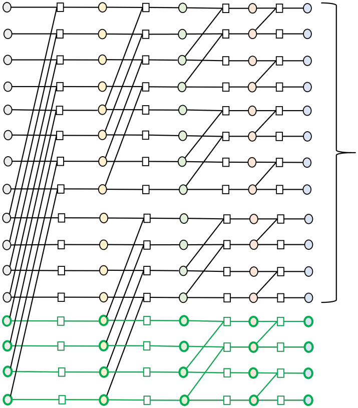


- Light nodes do not need to sample VNs from the last μ rows

Sampling Efficient Freezing (SEF) Algorithm

Lemma

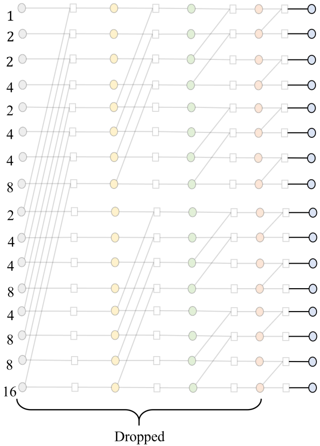
If we freeze last μ indices from the bottom of the encoding graph, then a stopping set with no frozen VNs cannot have a VN from the last μ rows of the encoding graph.



- Light nodes do not need to sample VNs from the last μ rows
  -Improves the effective undecodable threshold

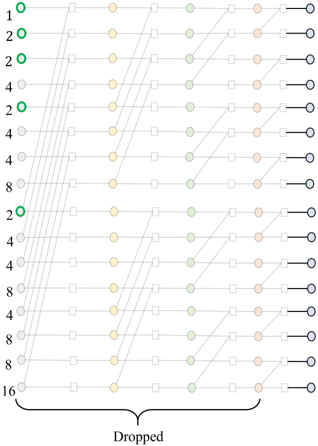
Sampling Efficient Freezing (SEF) Algorithm

Design of (16, 8) Polar code:



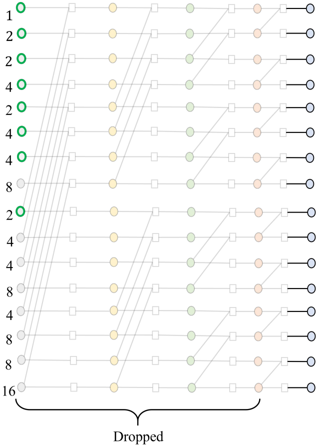
Sampling Efficient Freezing (SEF) Algorithm

Design of (16, 8) Polar code:



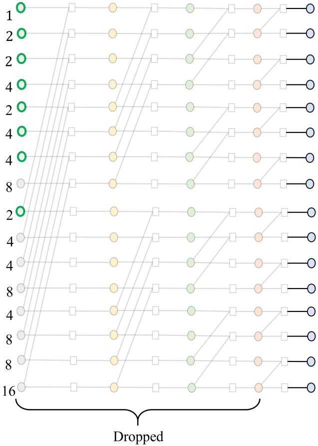
Sampling Efficient Freezing (SEF) Algorithm

Design of (16, 8) Polar code:



Sampling Efficient Freezing (SEF) Algorithm

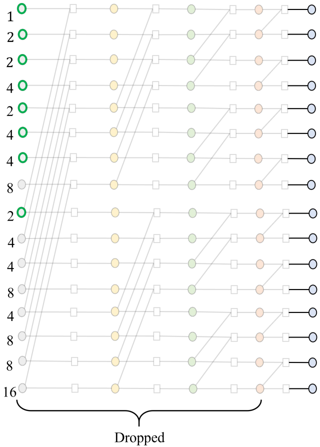
Design of (16, 8) Polar code:



- $\alpha_{\min} = 4$

Sampling Efficient Freezing (SEF) Algorithm

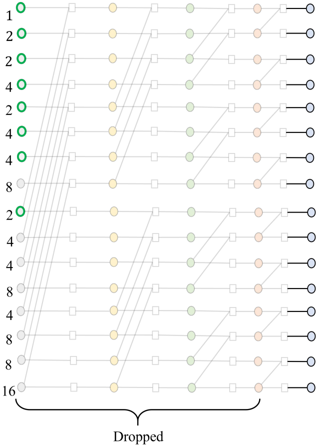
Design of (16, 8) Polar code:

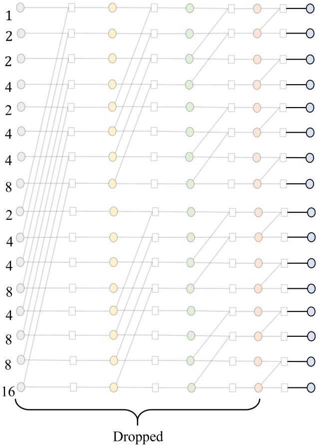


- $\alpha_{\min} = 4$
- $P_f(s) = (1 - \frac{4}{16})^s$

Sampling Efficient Freezing (SEF) Algorithm

Design of (16, 8) Polar code:

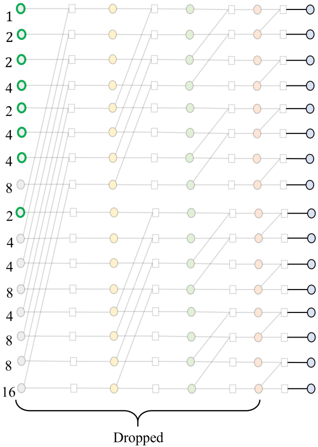


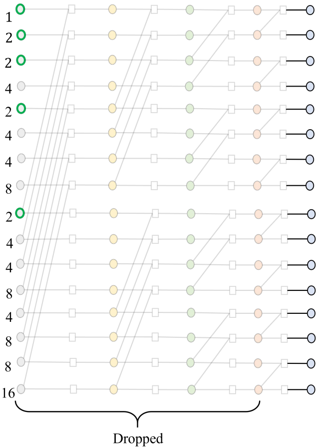


- $\alpha_{\min} = 4$
- $P_f(s) = (1 - \frac{4}{16})^s$

Sampling Efficient Freezing (SEF) Algorithm

Design of (16, 8) Polar code:

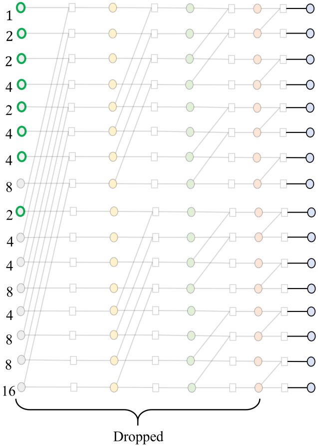


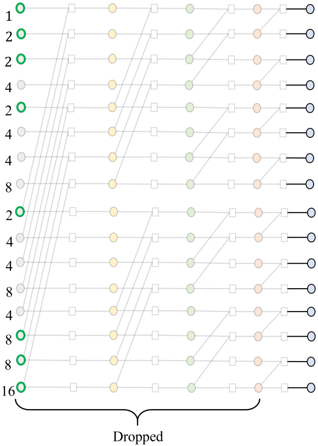


- $\alpha_{\min} = 4$
- $P_f(s) = (1 - \frac{4}{16})^s$

Sampling Efficient Freezing (SEF) Algorithm

Design of (16, 8) Polar code:

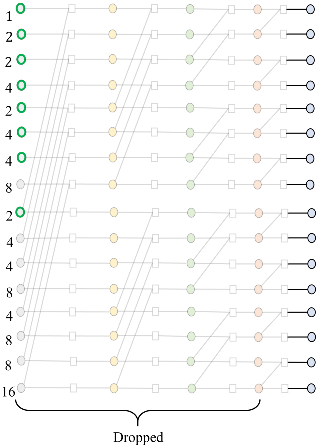




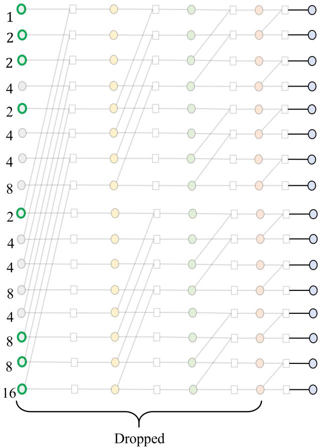
- $\alpha_{\min} = 4$
- $P_f(s) = (1 - \frac{4}{16})^s$

Sampling Efficient Freezing (SEF) Algorithm

Design of (16, 8) Polar code:



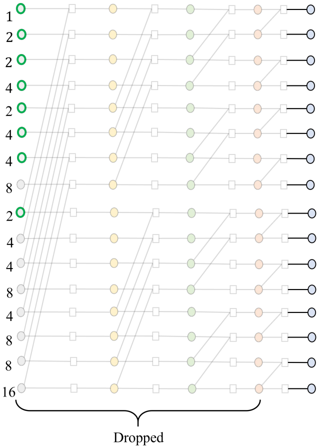
- $\alpha_{\min} = 4$
- $P_f(s) = (1 - \frac{4}{16})^s$



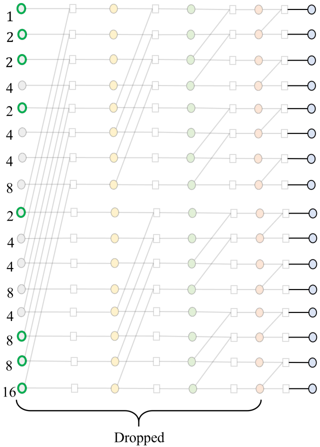
- $\alpha_{\min} = 4$,

Sampling Efficient Freezing (SEF) Algorithm

Design of (16, 8) Polar code:



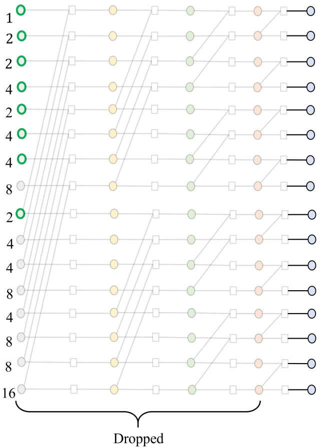
- $\alpha_{\min} = 4$
- $P_f(s) = (1 - \frac{4}{16})^s$



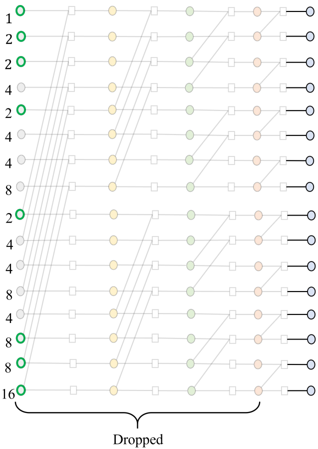
- $\alpha_{\min} = 4, \; P_f(s) = (1 - \frac{4}{13})^s$

Sampling Efficient Freezing (SEF) Algorithm

Design of (16, 8) Polar code:



- $\alpha_{\min} = 4$
- $P_f(s) = (1 - \frac{4}{16})^s$



- $\alpha_{\min} = 4,\ P_f(s) = (1 - \frac{4}{13})^s$
- $\alpha_{\min}^{\text{effective}} = \frac{4*16}{13} = 4.923$

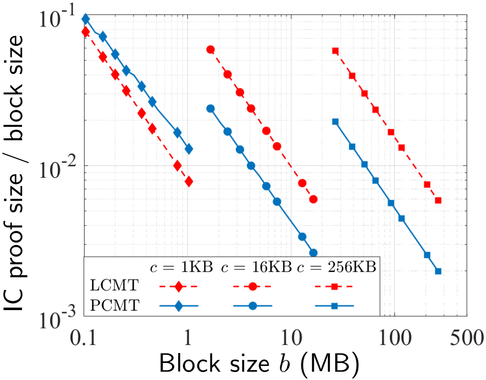
Simulation Results: IC-proof size

Parameters: Rate $R = 0.5$, Code length N, Data chunk size c,
Block size $b = cRN$, Hash size $= 32B$

Simulation Results: IC-proof size

Parameters: Rate $R = 0.5$, Code length N, Data chunk size c,
Block size $b = cRN$, Hash size $= 32B$

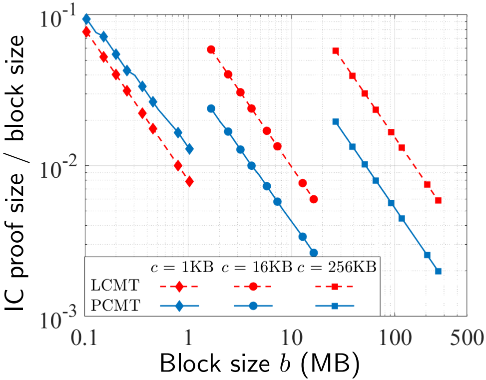
LCMT: LDPC CMT



Simulation Results: IC-proof size

Parameters: Rate $R = 0.5$, Code length N, Data chunk size c, Block size $b = cRN$, Hash size $= 32B$

LCMT: LDPC CMT



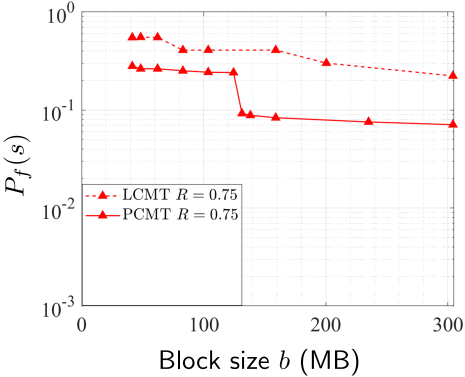
- For large block sizes, IC-proof size of PCMT is lower than LCMT

Simulation Results: Probability of failure

Code length N, Data chunk size $c = 256KB$, Hash size $32B$, Block size $b = cRN$, number of samples s such that total sample download is $\frac{b}{5}$

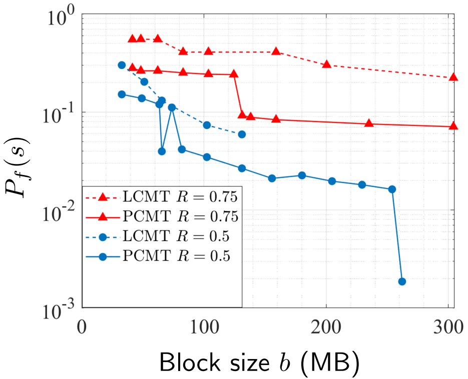
Simulation Results: Probability of failure

Code length N, Data chunk size $c = 256KB$, Hash size $32B$, Block size $b = cRN$, number of samples s such that total sample download is $\frac{b}{5}$



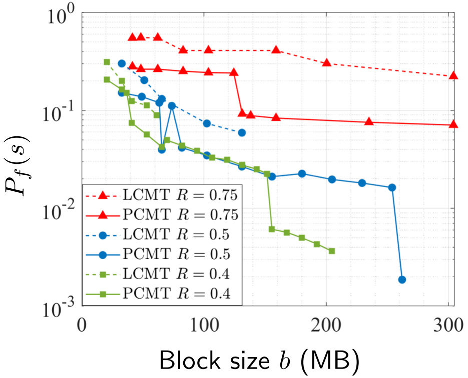
Simulation Results: Probability of failure

Code length N, Data chunk size $c = 256KB$, Hash size $32B$, Block size $b = cRN$, number of samples s such that total sample download is $\frac{b}{5}$



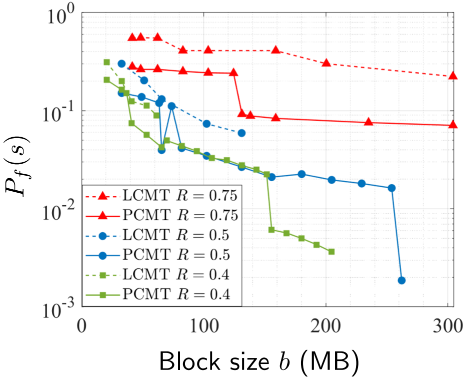
Simulation Results: Probability of failure

Code length N, Data chunk size $c = 256KB$, Hash size $32B$, Block size $b = cRN$, number of samples s such that total sample download is $\frac{b}{5}$



Simulation Results: Probability of failure

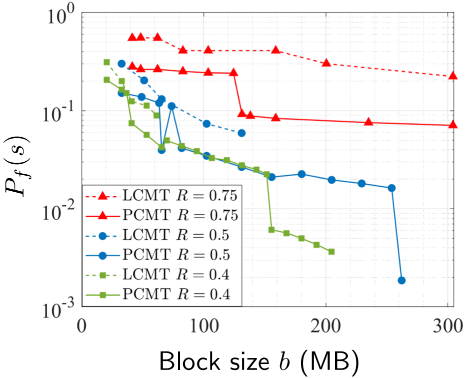
Code length N, Data chunk size $c = 256KB$, Hash size $32B$, Block size $b = cRN$, number of samples s such that total sample download is $\frac{b}{5}$



- For large block sizes (100-300MB), $P_f(s)$ for PCMT is lower than LCMT

Simulation Results: Probability of failure

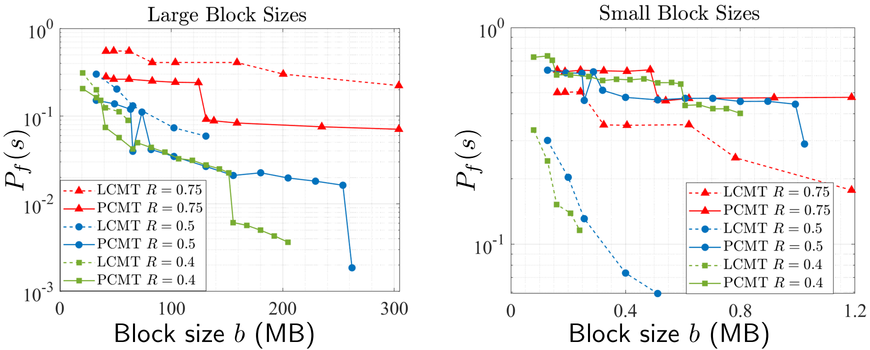
Code length N, Data chunk size $c = 256KB$, Hash size $32B$, Block size $b = cRN$, number of samples s such that total sample download is $\frac{b}{5}$



- For large block sizes (100-300MB), $P_f(s)$ for PCMT is lower than LCMT
  - Note: For small block sizes, $P_f(s)$ for PCMT gets worse than LCMT

Simulation Results: Probability of failure

Code length N, Data chunk size $c = 256KB$, Hash size $32B$, Block size $b = cRN$, number of samples s such that total sample download is $\frac{b}{5}$



- For large block sizes (100-300MB), $P_f(s)$ for PCMT is lower than LCMT
  - Note: For small block sizes, $P_f(s)$ for PCMT gets worse than LCMT

Conclusion and Future Work

Conclusion:

Conclusion and Future Work

Conclusion:

- Provided a novel construction of a Merkle Tree using Polar codes called PCMT

Conclusion and Future Work

Conclusion:

- Provided a novel construction of a Merkle Tree using Polar codes called PCMT
- A specialized polar code construction for PCMT called SEF algorithm

Conclusion and Future Work

Conclusion:

- Provided a novel construction of a Merkle Tree using Polar codes called PCMT
- A specialized polar code construction for PCMT called SEF algorithm
- PCMT with SEF polar codes perform well in detecting DA attacks and offers a new trade-off in various metrics compared to prior literature

Conclusion and Future Work

Conclusion:

- Provided a novel construction of a Merkle Tree using Polar codes called PCMT
- A specialized polar code construction for PCMT called SEF algorithm
- PCMT with SEF polar codes perform well in detecting DA attacks and offers a new trade-off in various metrics compared to prior literature

Future Work:

Conclusion and Future Work

Conclusion:

- Provided a novel construction of a Merkle Tree using Polar codes called PCMT
- A specialized polar code construction for PCMT called SEF algorithm
- PCMT with SEF polar codes perform well in detecting DA attacks and offers a new trade-off in various metrics compared to prior literature

Future Work:

- Improve the PCMT construction to make it not store hashes of all VNs of the encoding graph

Conclusion and Future Work

Conclusion:

- Provided a novel construction of a Merkle Tree using Polar codes called PCMT
- A specialized polar code construction for PCMT called SEF algorithm
- PCMT with SEF polar codes perform well in detecting DA attacks and offers a new trade-off in various metrics compared to prior literature

Future Work:

- Improve the PCMT construction to make it not store hashes of all VNs of the encoding graph
- Extend the PCMT construction to other encoding trellises

References

- (Al-Bassam '18) M. Al-Bassam, et al., "Fraud and Data Availability Proofs: Maximising Light Client Security and Scaling Blockchains with Dishonest Majorities," *arXiv preprint arXiv:1809.09044*, 2018.
- (Yu '19) M. Yu, et al., "Coded Merkle Tree: Solving Data Availability Attacks in Blockchains," *International Conference on Financial Cryptography and Data Security*, Springer, Cham, 2020.
- (Krishnan '07) K. M. Krishnan, and P. Shankar, "Computing the stopping distance of a Tanner graph is NP-hard," *IEEE Transactions on Information Theory*, vol. 53, no. 6, pp. 2278-2280, Jun. 2007.
- (Goela '10) N. Goela, S. B. Korada, and M. Gastpar, "On LP decoding of polar codes," *IEEE Information Theory Workshop*, pp. 1-5, Aug. 2010.
- (Eslami '13) A. Eslami, H. Pishro-Nik, "On finite-length performance of polar codes: stopping sets, error floor, and concatenated design," *IEEE Transactions on Communications*, vol. 61, no. 3, pp. 919-929, Feb. 2013
- (Santini '22) P. Santini, G. Rafaiani, M. Battaglioni, F. Chiaraluce, M. Baldi, "Optimization of a Reed-Solomon code-based protocol against blockchain data availability attacks", *arXiv preprint arXiv:2201.08261*, 2022.

Simulation Results

$\mathcal{T}_1$: small block size
$\mathcal{T}_2$: large block size

	2D-RS [Al-Bassam '18] [Santini '22]		LCMT		PCMT	
	$\mathcal{T}_1$	$\mathcal{T}_2$	$\mathcal{T}_1$	$\mathcal{T}_2$	$\mathcal{T}_1$	$\mathcal{T}_2$
Root size (KB)	2.05	5.82	0.26	0.51	1.02	2.56
IC proof size (MB)	5.80	16.40	1.54	1.54	0.53	0.54
Undecodable threshold $\alpha_{\min}$	Analytical expression		NP-hard		Analytical expression	
Decoding complexity	$O(N^{1.5})$		$O(N)$		$O(N\lceil\log N\rceil)$	

Simulation Results

$\mathcal{T}_1$: small block size
$\mathcal{T}_2$: large block size

	2D-RS [Al-Bassam '18] [Santini '22]		LCMT		PCMT	
	$\mathcal{T}_1$	$\mathcal{T}_2$	$\mathcal{T}_1$	$\mathcal{T}_2$	$\mathcal{T}_1$	$\mathcal{T}_2$
Root size (KB)	2.05	5.82	0.26	0.51	1.02	2.56
IC proof size (MB)	5.80	16.40	1.54	1.54	0.53	0.54
Undecodable threshold $\alpha_{\min}$	Analytical expression		NP-hard		Analytical expression	
Decoding complexity	$O(N^{1.5})$		$O(N)$		$O(N\lceil\log N\rceil)$	

- PCMT offers a new trade-off in the metrics of importance compared to LCMT and 2D-RS codes that were used in prior literature