Lecture #17:
Validity Rollups

COMS 4995-001:
The Science of Blockchains

URL: https://timroughgarden.org/s25/

Tim Roughgarden

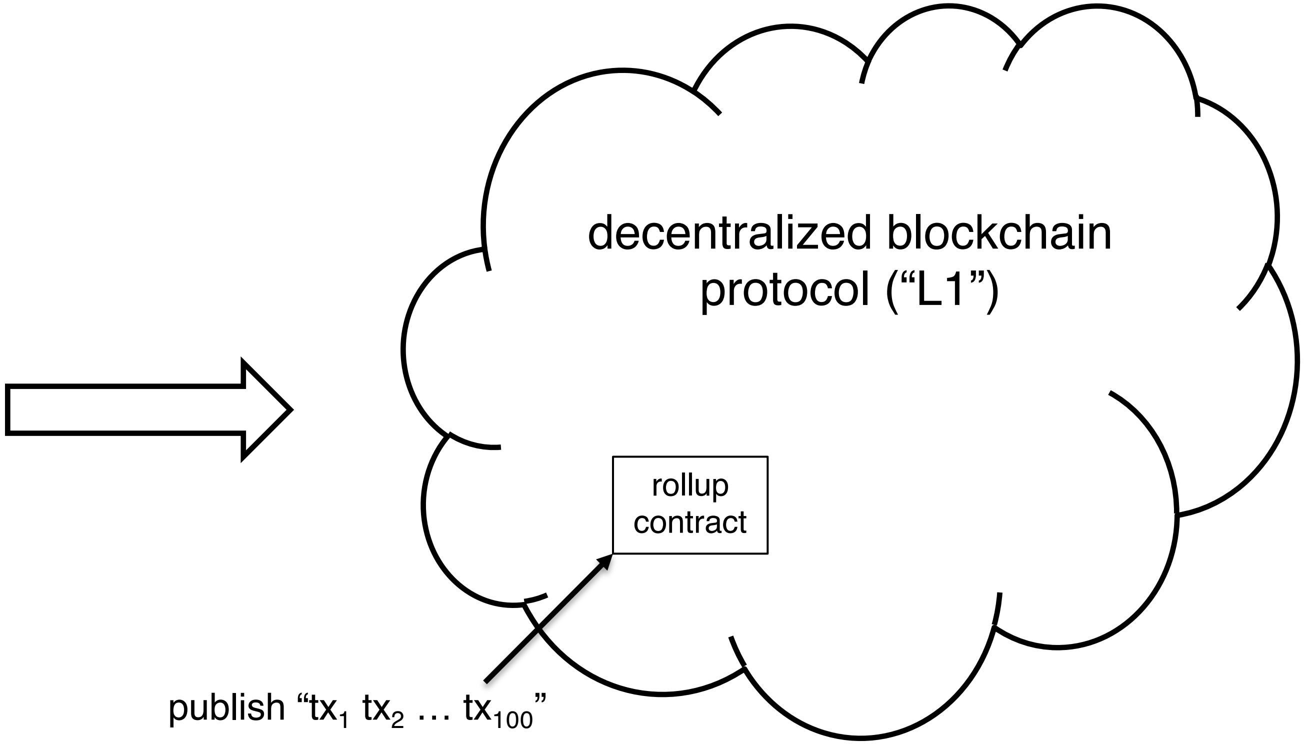
Recall: “Classic” Rollups

“Classic” rollup: a blockchain/virtual machine with its own state

- not necessarily decentralized, subject to crash or Byzantine failure
- performs its own consensus (i.e., tx sequencing) and execution
- associated with smart contract(s) running on the L1
- publishes rollup txs via L1 contract (i.e., uses L1 for data availability)
  - note: anyone can run a rollup full node (i.e., maintain full rollup state)
- periodically publishes commitment to rollup state (e.g. state root) to L1
  - note: any full node can check correctness of commitment
- (hard part) state commitment correctness verified by L1
  - question: how can L1 do this without re-executing rollup txs itself?

L1 ⇔ Rollup Architecture

(possibly centralized) rollup



Goals for Lecture #17

1. Validity rollups. (e.g., Starknet, zkSync)
   – rollup state commitments verified by L1 using “SNARK” proofs
   – cryptographic (rather than cryptoeconomic) security
2. Probabilistic verification.
   – need verification of correct tx execution << actual tx execution
   – example: matrix multiplication (Freivalds’ algorithm)
3. The Fiat-Shamir heuristic.
   – non-manipulable randomness from cryptographic hash functions
   – “flattens” an iterative/interactive computation into a single proof

Drawbacks of Optimistic Rollups

Recall design: rely on watchdogs to catch incorrect state commitments, submit short proof of incorrectness (“fault proof”).

- L1 performs minimal re-execution necessary to resolve dispute

Drawbacks:

Drawbacks of Optimistic Rollups

Recall design: rely on watchdogs to catch incorrect state commitments, submit short proof of incorrectness ("fault proof").

– L1 performs minimal re-execution necessary to resolve dispute

Drawbacks:

- complex fault-proof logic (warning: SNARKs far more complex)

Drawbacks of Optimistic Rollups

Recall design: rely on watchdogs to catch incorrect state commitments, submit short proof of incorrectness ("fault proof").

– L1 performs minimal re-execution necessary to resolve dispute

Drawbacks:

- complex fault-proof logic (warning: SNARKs far more complex)
- "1 in N" trust assumption for watchdogs

Drawbacks of Optimistic Rollups

Recall design: rely on watchdogs to catch incorrect state commitments, submit short proof of incorrectness (“fault proof”).

– L1 performs minimal re-execution necessary to resolve dispute

Drawbacks:

- complex fault-proof logic (warning: SNARKs far more complex)
- “1 in N” trust assumption for watchdogs
- attacks preventing honest challengers from submitting L1 txs

Drawbacks of Optimistic Rollups

Recall design: rely on watchdogs to catch incorrect state commitments, submit short proof of incorrectness (“fault proof”).

– L1 performs minimal re-execution necessary to resolve dispute

Drawbacks:

- complex fault-proof logic (warning: SNARKs far more complex)
- “1 in N” trust assumption for watchdogs
- attacks preventing honest challengers from submitting L1 txs
- delay (≈ 7 days) before finalization of a state commitment
  - users can proceed on basis of “preconfirmation,” if desired

Validity Rollups

Warning: often called "zk" rollups. (even though not zero-knowledge)

Validity Rollups

Warning: often called “zk” rollups. (even though not zero-knowledge)

Recall: in a “classic” rollup (optimistic or validity), periodically publish rollup txs to L1, along with new state commitment.

Validity Rollups

Warning: often called “zk” rollups. (even though not zero-knowledge)

Recall: in a “classic” rollup (optimistic or validity), periodically publish rollup txs to L1, along with new state commitment.

High-level idea of validity rollups: guilty until proven innocent.

- L1 assumes by default that each state commitment is incorrect

Validity Rollups

Warning: often called “zk” rollups. (even though not zero-knowledge)

Recall: in a “classic” rollup (optimistic or validity), periodically publish rollup txs to L1, along with new state commitment.

High-level idea of validity rollups: guilty until proven innocent.

- L1 assumes by default that each state commitment is incorrect
- rely on “provers” to submit proofs of correctness to L1
  - if nothing else, rollup operator can run its own prover

Validity Rollups

Warning: often called “zk” rollups. (even though not zero-knowledge)

Recall: in a “classic” rollup (optimistic or validity), periodically publish rollup txs to L1, along with new state commitment.

High-level idea of validity rollups: guilty until proven innocent.

- L1 assumes by default that each state commitment is incorrect
- rely on “provers” to submit proofs of correctness to L1
  - if nothing else, rollup operator can run its own prover
- L1 verifies proof of correctness directly
  - state commitment rejected if accompanying proof fails verification

Validity Rollups

Recall: in a “classic” rollup (optimistic or validity), periodically publish rollup txs to L1, along with new state commitment.

High-level idea of validity rollups: guilty until proven innocent.

- L1 assumes by default that each state commitment is incorrect
- rely on “provers” to submit proofs of correctness to L1
- L1 verifies proof of correctness directly

Hard part: verification of correctness proofs should be *much* easier than tx re-execution --- i.e., need “SNARKs.”

Matrix Multiplication

Matrix Multiplication

Let's drill down on the $n = 2$ case. We can describe two 2×2 matrices using eight parameters:

$$\underbrace{\begin{bmatrix} a & b \\ c & d \end{bmatrix}}_{\mathbf{X}} \quad \text{and} \quad \underbrace{\begin{bmatrix} e & f \\ g & h \end{bmatrix}}_{\mathbf{Y}}.$$

In the matrix product $\mathbf{X} \cdot \mathbf{Y}$, the upper-left entry is the dot product of the first row of $\mathbf{X}$ and the first column of $\mathbf{Y}$, or $ae + bg$. In general, for $\mathbf{X}$ and $\mathbf{Y}$ as above,

$$\mathbf{X} \cdot \mathbf{Y} = \begin{bmatrix} ae + bg & af + bh \\ ce + dg & cf + dh \end{bmatrix}. \tag{3.2}$$

Matrix Multiplication

Let's drill down on the $n = 2$ case. We can describe two 2×2 matrices using eight parameters:

$$\underbrace{\begin{bmatrix} a & b \\ c & d \end{bmatrix}}_{\mathbf{X}} \text{ and } \underbrace{\begin{bmatrix} e & f \\ g & h \end{bmatrix}}_{\mathbf{Y}}.$$

In the matrix product $\mathbf{X} \cdot \mathbf{Y}$, the upper-left entry is the dot product of the first row of $\mathbf{X}$ and the first column of $\mathbf{Y}$, or $ae + bg$. In general, for $\mathbf{X}$ and $\mathbf{Y}$ as above,

$$\mathbf{X} \cdot \mathbf{Y} = \begin{bmatrix} ae + bg & af + bh \\ ce + dg & cf + dh \end{bmatrix}. \tag{3.2}$$

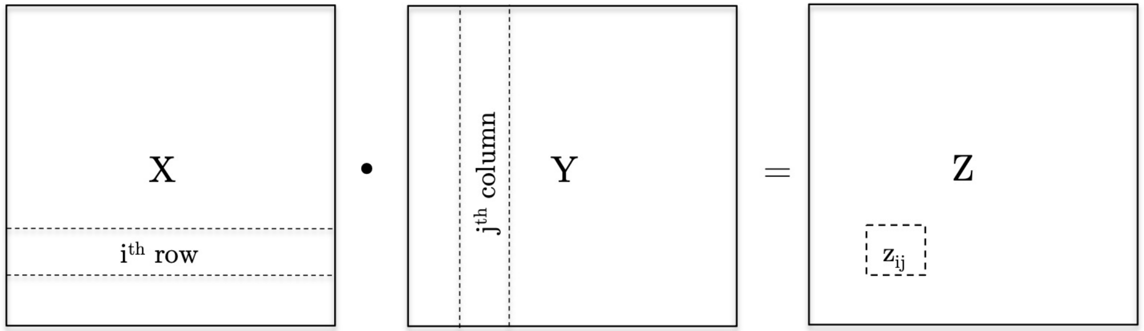


Figure 3.2: The (i, j) entry of the matrix product $\mathbf{X} \cdot \mathbf{Y}$ is the dot product of the ith row of $\mathbf{X}$ and the jth column of $\mathbf{Y}$.

Matrix Multiplication (con'd)

Note: can compute the product of two $n \times n$ matrices in $O(n^3)$ time.

- n^2 dot products, $O(n)$ time for each

Straightforward Matrix Multiplication

Input: $n \times n$ integer matrices $\mathbf{X}$ and $\mathbf{Y}$.
Output: $\mathbf{Z} = \mathbf{X} \cdot \mathbf{Y}$.

---

for $i := 1$ to n **do** // loop over rows of $\mathbf{X}$
  for $j := 1$ to n **do** // loop over columns of $\mathbf{Y}$
    $\mathbf{Z}[i][j] := 0$
    for $k := 1$ to n **do** // compute dot product
      $\mathbf{Z}[i][j] := \mathbf{Z}[i][j] + \mathbf{X}[i][k] \cdot \mathbf{Y}[k][j]$
return $\mathbf{Z}$

Verifying Matrix Multiplication

Problem: matrix multiplication verification.

Input: three $n \times n$ matrices A, B, and C.

– C is allegedly the product of A and B

Verifying Matrix Multiplication

Problem: matrix multiplication verification.

Input: three $n \times n$ matrices A, B, and C.

– C is allegedly the product of A and B

Output: “yes” if $C = A \cdot B$ and “no” otherwise.

Verifying Matrix Multiplication

Problem: matrix multiplication verification.

Input: three $n \times n$ matrices A, B, and C.

- C is allegedly the product of A and B

Output: "yes" if $C = A \cdot B$ and "no" otherwise.

Obvious algorithm: compute A · B from scratch, compare result to C.

- running time = $O(n^3)$

Verifying Matrix Multiplication

Problem: matrix multiplication verification.

Input: three $n \times n$ matrices A, B, and C.

- C is allegedly the product of A and B

Output: "yes" if $C = A \cdot B$ and "no" otherwise.

Obvious algorithm: compute A · B from scratch, compare result to C.

- running time = $O(n^3)$
- or $O(n^{2.37})$ with the asymptotically best known (but hopelessly impractical) matrix multiplication algorithm

Freivalds' Algorithm

Input: three $n \times n$ matrices A, B, and C.

Freivalds' Algorithm ('77):

Freivalds' Algorithm

Input: three $n \times n$ matrices A, B, and C.

Freivalds' Algorithm ('77):

- for i = 1,2,…,t: [t = # of trials, parameter of our choosing]

Freivalds’ Algorithm

Input: three $n \times n$ matrices A, B, and C.

Freivalds’ Algorithm (’77):

- for i = 1,2,…,t: [t = # of trials, parameter of our choosing]
  - choose $x_i \in \{0,1\}^n$ uniformly at random [2^n choices, each equally likely]

Freivalds' Algorithm

Input: three $n \times n$ matrices A, B, and C.

Freivalds' Algorithm ('77):

- for i = 1,2,…,t: [t = # of trials, parameter of our choosing]
  - choose $x_i \in \{0,1\}^n$ uniformly at random [2^n choices, each equally likely]
  - compute $y_i \coloneqq C \cdot x_i$ [matrix-vector product]

Freivalds' Algorithm

Input: three $n \times n$ matrices A, B, and C.

Freivalds' Algorithm ('77):

- for i = 1,2,…,t: [t = # of trials, parameter of our choosing]
  - choose $x_i \in \{0,1\}^n$ uniformly at random [2^n choices, each equally likely]
  - compute $y_i \coloneqq C \cdot x_i$ [matrix-vector product]
  - compute $z_i \coloneqq A \cdot (B \cdot x_i)$ [two matrix-vector products]

Freivalds' Algorithm

Input: three $n \times n$ matrices A, B, and C.

Freivalds' Algorithm ('77):

- for i = 1,2,…,t: [t = # of trials, parameter of our choosing]
  - choose $x_i \in \{0,1\}^n$ uniformly at random [2^n choices, each equally likely]
  - compute $y_i \coloneqq C \cdot x_i$ [matrix-vector product]
  - compute $z_i \coloneqq A \cdot (B \cdot x_i)$ [two matrix-vector products]
  - if $y_i \neq z_i$, return "no"

Freivalds' Algorithm

Input: three $n \times n$ matrices A, B, and C.

Freivalds' Algorithm ('77):

- for i = 1,2,…,t: [t = # of trials, parameter of our choosing]
  - choose $x_i \in \{0,1\}^n$ uniformly at random [2^n choices, each equally likely]
  - compute $y_i \coloneqq C \cdot x_i$ [matrix-vector product]
  - compute $z_i \coloneqq A \cdot (B \cdot x_i)$ [two matrix-vector products]
  - if $y_i \neq z_i$, return "no"
- return "yes"

Freivalds' Algorithm: Example

- for i = 1,2,…,t:
  - choose $x_i \in \{0,1\}^n$ uniformly at random, compute $y_i \coloneqq C \cdot x_i$ and $z_i \coloneqq A \cdot (B \cdot x_i)$
  - if $y_i \neq z_i$, return "no"
- return "yes"

Example: $\mathrm{A} = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}$, $\mathrm{B} = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}$, $\mathrm{C} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}$

Freivalds' Algorithm: Example

- for i = 1,2,…,t:
  - choose $x_i \in \{0,1\}^n$ uniformly at random, compute $y_i \coloneqq C \cdot x_i$ and $z_i \coloneqq A \cdot (B \cdot x_i)$
  - if $y_i \neq z_i$, return "no"
- return "yes"

Example: $\mathrm{A} = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}$, $\mathrm{B} = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}$, $\mathrm{C} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}$

- for all $x \in \{(0,0), (1,0), (0,1), (1,1)\}$: $A \cdot (B \cdot x) = (0,0)$

Freivalds' Algorithm: Example

- for i = 1,2,…,t:
  - choose $x_i \in \{0,1\}^n$ uniformly at random, compute $y_i \coloneqq C \cdot x_i$ and $z_i \coloneqq A \cdot (B \cdot x_i)$
  - if $y_i \neq z_i$, return "no"
- return "yes"

Example: $\mathrm{A} = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}$, $\mathrm{B} = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}$, $\mathrm{C} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}$

- for all $x \in \{(0,0),(1,0),(0,1),(1,1)\}$: $A \cdot (B \cdot x) = (0,0)$
- for $x \in \{(0,0),(1,0)\}$: $C \cdot x = (0,0)$
- for $x \in \{(0,1),(1,1)\}$: $C \cdot x = (1,0)$

Freivalds' Algorithm: Example

- for i = 1,2,…,t:
  - choose $x_i \in \{0,1\}^n$ uniformly at random, compute $y_i \coloneqq C \cdot x_i$ and $z_i \coloneqq A \cdot (B \cdot x_i)$
  - if $y_i \neq z_i$, return "no"
- return "yes"

Example: $\mathrm{A} = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}$, $\mathrm{B} = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}$, $\mathrm{C} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}$

- for all $x \in \{(0,0), (1,0), (0,1), (1,1)\}$: $A \cdot (B \cdot x) = (0,0)$
- for $x \in \{(0,0), (1,0)\}$: $C \cdot x = (0,0)$
- for $x \in \{(0,1), (1,1)\}$: $C \cdot x = (1,0)$
  - upshot: $C \cdot x \neq A \cdot (B \cdot x)$ with 50% probability (over choice of x)

Freivalds' Algorithm: Running Time

- for i = 1,2,…,t:
  - choose $x_i \in \{0,1\}^n$ uniformly at random, compute $y_i := C \cdot x_i$ and $z_i := A \cdot (B \cdot x_i)$
  - if $y_i \neq z_i$, return "no"
- return "yes"

Running time analysis:

Freivalds' Algorithm: Running Time

- for i = 1,2,…,t:
  - choose $x_i \in \{0,1\}^n$ uniformly at random, compute $y_i \coloneqq C \cdot x_i$ and $z_i \coloneqq A \cdot (B \cdot x_i)$
  - if $y_i \neq z_i$, return "no"
- return "yes"

Running time analysis:

- t iterations

Freivalds' Algorithm: Running Time

- for i = 1,2,…,t:
  - choose $x_i \in \{0,1\}^n$ uniformly at random, compute $y_i \coloneqq C \cdot x_i$ and $z_i \coloneqq A \cdot (B \cdot x_i)$
  - if $y_i \neq z_i$, return "no"
- return "yes"

Running time analysis:

- t iterations
- three matrix-vector products per iteration ($O(n^2)$ time each)

Freivalds' Algorithm: Running Time

- for i = 1,2,…,t:
  - choose $x_i \in \{0,1\}^n$ uniformly at random, compute $y_i \coloneqq C \cdot x_i$ and $z_i \coloneqq A \cdot (B \cdot x_i)$
  - if $y_i \neq z_i$, return "no"
- return "yes"

Running time analysis:

- t iterations
- three matrix-vector products per iteration ($O(n^2)$ time each)
- overall running time = $O(t \cdot n^2)$

Freivalds' Algorithm: Correctness

- for i = 1,2,…,t:
  - choose $x_i \in \{0,1\}^n$ uniformly at random, compute $y_i \coloneqq C \cdot x_i$ and $z_i \coloneqq A \cdot (B \cdot x_i)$
  - if $y_i \neq z_i$, return "no"
- return "yes"

Correctness [case 1]: suppose $C = A \cdot B$.

Freivalds' Algorithm: Correctness

- for i = 1,2,…,t:
  - choose $x_i \in \{0,1\}^n$ uniformly at random, compute $y_i \coloneqq C \cdot x_i$ and $z_i \coloneqq A \cdot (B \cdot x_i)$
  - if $y_i \neq z_i$, return "no"
- return "yes"

Correctness [case 1]: suppose $C = A \cdot B$.

- for every $x \in \{0,1\}^n$, $C \cdot x = A \cdot (B \cdot x)$

Freivalds' Algorithm: Correctness

- for i = 1,2,…,t:
  - choose $x_i \in \{0,1\}^n$ uniformly at random, compute $y_i \coloneqq C \cdot x_i$ and $z_i \coloneqq A \cdot (B \cdot x_i)$
  - if $y_i \neq z_i$, return "no"
- return "yes"

Correctness [case 1]: suppose $C = A \cdot B$.

- for every $x \in \{0,1\}^n$, $C \cdot x = A \cdot (B \cdot x)$
- algorithm guaranteed to (correctly) return "yes"
  - i.e., no false negatives

Freivalds' Algorithm: Correctness

- for i = 1,2,…,t:
  - choose $x_i \in \{0,1\}^n$ uniformly at random, compute $y_i \coloneqq C \cdot x_i$ and $z_i \coloneqq A \cdot (B \cdot x_i)$
  - if $y_i \neq z_i$, return "no"
- return "yes"

Correctness [case 2]: suppose $C \neq A \cdot B$.

Freivalds' Algorithm: Correctness

- for i = 1,2,…,t:
  - choose $x_i \in \{0,1\}^n$ uniformly at random, compute $y_i \coloneqq C \cdot x_i$ and $z_i \coloneqq A \cdot (B \cdot x_i)$
  - if $y_i \neq z_i$, return "no"
- return "yes"

Correctness [case 2]: suppose $C \neq A \cdot B$.

- claim: every iteration i, ≥ 50% chance that $C \cdot x_i \neq A \cdot (B \cdot x_i)$

Freivalds' Algorithm: Correctness

- for i = 1,2,…,t:
  - choose $x_i \in \{0,1\}^n$ uniformly at random, compute $y_i \coloneqq C \cdot x_i$ and $z_i \coloneqq A \cdot (B \cdot x_i)$
  - if $y_i \neq z_i$, return "no"
- return "yes"

Correctness [case 2]: suppose $C \neq A \cdot B$.

- claim: every iteration i, ≥ 50% chance that $C \cdot x_i \neq A \cdot (B \cdot x_i)$
- thus: ≤ 2^{-t} probability that $C \cdot x_i = A \cdot (B \cdot x_i)$ for all i=1,2,…,t

Freivalds' Algorithm: Correctness

- for i = 1,2,…,t:
  - choose $x_i \in \{0,1\}^n$ uniformly at random, compute $y_i \coloneqq C \cdot x_i$ and $z_i \coloneqq A \cdot (B \cdot x_i)$
  - if $y_i \neq z_i$, return "no"
- return "yes"

Correctness [case 2]: suppose $C \neq A \cdot B$.

- claim: every iteration i, ≥ 50% chance that $C \cdot x_i \neq A \cdot (B \cdot x_i)$
- thus: ≤ 2^{-t} probability that $C \cdot x_i = A \cdot (B \cdot x_i)$ for all i=1,2,…,t
- ➔ algorithm (correctly) returns "no" except with ≤ 2^{-t} probability
  - i.e., false positive probability ≤ 2^{-t}

Freivalds' Algorithm: Correctness

Claim: if $C \neq A \cdot B$ and $x \in \{0,1\}^n$ chosen uniformly at random, then $C \cdot x \neq A \cdot (B \cdot x)$ with probability ≥ ½.

Proof of claim:

Freivalds' Algorithm: Correctness

Claim: if $C \neq A \cdot B$ and $x \in \{0,1\}^n$ chosen uniformly at random, then $C \cdot x \neq A \cdot (B \cdot x)$ with probability ≥ ½.

Proof of claim:

- $C \cdot x \neq A \cdot (B \cdot x) \Leftrightarrow M \cdot x \neq 0$, where $M := C - A \cdot B$

Freivalds' Algorithm: Correctness

Claim: if $C \neq A \cdot B$ and $x \in \{0,1\}^n$ chosen uniformly at random, then $C \cdot x \neq A \cdot (B \cdot x)$ with probability ≥ ½.

Proof of claim:

- $C \cdot x \neq A \cdot (B \cdot x) \Leftrightarrow M \cdot x \neq 0$, where $M := C - A \cdot B$
- let j be index of non-zero column of M (exists because $C \neq A \cdot B$)

Freivalds' Algorithm: Correctness

Claim: if $C \neq A \cdot B$ and $x \in \{0,1\}^n$ chosen uniformly at random, then $C \cdot x \neq A \cdot (B \cdot x)$ with probability ≥ ½.

Proof of claim:

- $C \cdot x \neq A \cdot (B \cdot x) \Leftrightarrow M \cdot x \neq 0$, where $M := C - A \cdot B$
- let j be index of non-zero column of M (exists because $C \neq A \cdot B$)
- note: if x, x' differ only in jth coordinate, $M \cdot x \neq M \cdot x'$
  - $M \cdot (x - x') = \pm$ jth column of M

Freivalds' Algorithm: Correctness

Claim: if $C \neq A \cdot B$ and $x \in \{0,1\}^n$ chosen uniformly at random, then $C \cdot x \neq A \cdot (B \cdot x)$ with probability ≥ ½.

Proof of claim:

- $C \cdot x \neq A \cdot (B \cdot x) \Leftrightarrow M \cdot x \neq 0$, where $M := C - A \cdot B$
- let j be index of non-zero column of M (exists because $C \neq A \cdot B$)
- note: if x, x' differ only in jth coordinate, $M \cdot x \neq M \cdot x'$
- ➔ either $M \cdot x \neq 0$ or $M \cdot x' \neq 0$ (or both)

Freivalds' Algorithm: Correctness

Claim: if $C \neq A \cdot B$ and $x \in \{0,1\}^n$ chosen uniformly at random, then $C \cdot x \neq A \cdot (B \cdot x)$ with probability ≥ ½.

Proof of claim:

- $C \cdot x \neq A \cdot (B \cdot x) \Leftrightarrow M \cdot x \neq 0$, where $M := C - A \cdot B$
- let j be index of non-zero column of M (exists because $C \neq A \cdot B$)
- note: if x, x' differ only in jth coordinate, $M \cdot x \neq M \cdot x'$
- ➔ either $M \cdot x \neq 0$ or $M \cdot x' \neq 0$ (or both)
- ➔ number of x's with $M \cdot x = 0$ ≤ number of x's with $M \cdot x \neq 0$

Freivalds' Algorithm: Correctness

Claim: if $C \neq A \cdot B$ and $x \in \{0,1\}^n$ chosen uniformly at random, then $C \cdot x \neq A \cdot (B \cdot x)$ with probability ≥ ½.

Proof of claim:

- $C \cdot x \neq A \cdot (B \cdot x) \Leftrightarrow M \cdot x \neq 0$, where $M := C - A \cdot B$
- let j be index of non-zero column of M (exists because $C \neq A \cdot B$)
- note: if x, x' differ only in jth coordinate, $M \cdot x \neq M \cdot x'$
- ➔ either $M \cdot x \neq 0$ or $M \cdot x' \neq 0$ (or both)
- ➔ number of x's with $M \cdot x = 0$ ≤ number of x's with $M \cdot x \neq 0$
- ➔ $M \cdot x \neq 0$ with probability ≥ ½ over choice of $x \in \{0,1\}^n$

Freivalds' Algorithm: Report Card

- running time = $O(t \cdot n^2)$ [t = number of trials]

Freivalds' Algorithm: Report Card

- running time = $O(t \cdot n^2)$ [t = number of trials]
- "completeness" = 1
  - i.e., 0% false negative probability on "yes" inputs

Freivalds' Algorithm: Report Card

- running time = $O(t \cdot n^2)$ [t = number of trials]
- "completeness" = 1
  - i.e., 0% false negative probability on "yes" inputs
- "soundness" = $\leq 2^{-t}$
  - i.e., $\leq 2^{-t}$ false positive probability on "no" inputs

Freivalds' Algorithm: Report Card

- running time = $O(t \cdot n^2)$ [t = number of trials]
- "completeness" = 1
  - i.e., 0% false negative probability on "yes" inputs
- "soundness" = ≤ 2^{-t}
  - i.e., ≤ 2^{-t} false positive probability on "no" inputs

Upshot: can verify correctness of matrix multiplication in $O(n^2)$ time with arbitrarily small constant error.

- cf., "recompute from scratch" algorithm that takes $O(n^3)$ (or $O(n^{2.37})$) time

From Algorithms to Proofs

Question: how can a layer-one blockchain protocol verify that $C = A \cdot B$ without recomputing $A \cdot B$ from scratch?

From Algorithms to Proofs

Question: how can a layer-one blockchain protocol verify that $C = A \cdot B$ without recomputing $A \cdot B$ from scratch?

Idea: post C along with $x_1, x_2, \ldots, x_t$. [assume L1 knows A and B]

- L1 accepts answer ⇔ $C \cdot x_i = A \cdot (B \cdot x_i)$ for all i=1,2,…,t
  - L1 only has to carry out matrix-vector products, not matrix multiplication

From Algorithms to Proofs

Question: how can a layer-one blockchain protocol verify that $C = A \cdot B$ without recomputing $A \cdot B$ from scratch?

Idea: post C along with $x_1, x_2, \ldots, x_t$. [assume L1 knows A and B]

- L1 accepts answer ⬄ $C \cdot x_i = A \cdot (B \cdot x_i)$ for all i=1,2,…,t
  - L1 only has to carry out matrix-vector products, not matrix multiplication

Problem: could post $C \neq A \cdot B$ along with $x_1 = x_2 = \cdots = x_n = \vec{0}$.

- adversarially chosen x_i's can trick L1 into accepting incorrect answer

From Algorithms to Proofs

Question: how can a layer-one blockchain protocol verify that $C = A \cdot B$ without recomputing $A \cdot B$ from scratch?

Idea: post C along with $x_1, x_2, \ldots, x_t$. [assume L1 knows A and B]

- L1 accepts answer ⇔ $C \cdot x_i = A \cdot (B \cdot x_i)$ for all i=1,2,…,t
  - L1 only has to carry out matrix-vector products, not matrix multiplication

Problem: could post $C \neq A \cdot B$ along with $x_1 = x_2 = \cdots = x_n = \vec{0}$.

- adversarially chosen x_i's can trick L1 into accepting incorrect answer
- need to somehow ensure that the x_i's are (as good as) random

The Fiat-Shamir Heuristic

Idea: post C along with $x_1, x_2, \ldots, x_t$. [assume L1 knows A and B]

- L1 accepts answer ⇔ $C \cdot x_i = A \bullet (B \cdot x_i)$ for all i=1,2,…,t
- need to somehow ensure that the x_i's are (as good as) random

The Fiat-Shamir Heuristic

Idea: post C along with $x_1, x_2, \ldots, x_t$. [assume L1 knows A and B]

– L1 accepts answer ⬄ $C \cdot x_i = A \bullet (B \cdot x_i)$ for all i=1,2,…,t
– need to somehow ensure that the x_i's are (as good as) random

Fiat-Shamir heuristic: derive the x_i's from the outputs of a cryptographic hash function h (e.g., h = SHA-256).

The Fiat-Shamir Heuristic

Idea: post C along with $x_1, x_2, \ldots, x_t$. [assume L1 knows A and B]

- L1 accepts answer ⇔ $C \cdot x_i = A \bullet (B \cdot x_i)$ for all i=1,2,…,t
- need to somehow ensure that the x_i's are (as good as) random

Fiat-Shamir heuristic: derive the x_i's from the outputs of a cryptographic hash function h (e.g., h = SHA-256).

- e.g., if n=256, set $x_i = h(C||i)$ for each i=1,2,…,t
  - interpret output of hash function as a 0-1 vector
  - for larger n, apply this idea to each "chunk" of 256 coordinates

The Fiat-Shamir Heuristic

Idea: post C along with $x_1,x_2,\ldots,x_t$. [assume L1 knows A and B]

- L1 accepts answer ⇔ $C \cdot x_i = A \cdot (B \cdot x_i)$ for all i=1,2,…,t
- need to somehow ensure that the x_i's are (as good as) random

Fiat-Shamir heuristic: derive the x_i's from the outputs of a cryptographic hash function h (e.g., h = SHA-256).

- e.g., if n=256, set $x_i = h(C||i)$ for each i=1,2,…,t (e.g., t=128)
- assuming h acts like a random function, would need ≈ 2^t attempts to find a matrix $C \neq A \cdot B$ with $C \cdot x_i = A \cdot (B \cdot x_i)$ for all i=1,2,…,t
  - "computational soundness" (i.e., infeasible to produce false proof)

The Fiat-Shamir Heuristic

Idea: post C along with $x_1, x_2, \ldots, x_t$. [assume L1 knows A and B]

- L1 accepts answer ⇔ $C \cdot x_i = A \bullet (B \cdot x_i)$ for all i=1,2,…,t
- need to somehow ensure that the x_i's are (as good as) random

Fiat-Shamir heuristic: derive the x_i's from the outputs of a cryptographic hash function h (e.g., h = SHA-256).

- e.g., if n=256, set $x_i = h(C||i)$ for each i=1,2,…,t (e.g., t=128)
- assuming h acts like a random function, would need ≈ 2^t attempts to find a matrix $C \neq A \bullet B$ with $C \cdot x_i = A \bullet (B \cdot x_i)$ for all i=1,2,…,t
  - "computational soundness" (i.e., infeasible to produce false proof)
  - question: what goes wrong if instead set $x_i = h(i)$ for all i?

Recap

- Freivalds' algorithm + Fiat-Shamir heuristic ➔ proofs of matrix multiplication such that:
  - size = $O(n^2)$ [size of the answer]
  - proof verification = $O(n^2)$ time [thinking of t as constant]
  - perfect completeness [correct answers always successfully verify]
  - computational soundness [computationally infeasible to find false proofs]

Recap

- Freivalds' algorithm + Fiat-Shamir heuristic ➔ proofs of matrix multiplication such that:
  - size = $O(n^2)$ [size of the answer]
  - proof verification = $O(n^2)$ time [thinking of t as constant]
  - perfect completeness [correct answers always successfully verify]
  - computational soundness [computationally infeasible to find false proofs]
- for validity rollups, need:
  - verification of arbitrary computation (not just matrix multiplication)

Recap

- Freivalds' algorithm + Fiat-Shamir heuristic ➔ proofs of matrix multiplication such that:
  - size = $O(n^2)$ [size of the answer]
  - proof verification = $O(n^2)$ time [thinking of t as constant]
  - perfect completeness [correct answers always successfully verify]
  - computational soundness [computationally infeasible to find false proofs]
- for validity rollups, need:
  - verification of arbitrary computation (not just matrix multiplication)
  - short proofs that can be posted to an L1 (along with state commitment)
    - i.e., "SNARKs" (see next lecture)