GKR Protocol. Offline Memory Checking

July 17, 2025

Distributed Lab

zkdl-camp.github.io
github.com/ZKDL-Camp

Introduction

Recap: Multivariate World and Sum-Check

Univariate World:

$$p(X) = q(X) \prod_{u \in \Omega} (X - u)$$

Multivariate World:

$$\sum_{\mathbf{b} \in \{0,1\}^\ell} f(b_1, \ldots, b_\ell) = H$$

Goal: build the set of constraints that boil down to **Sum-Check**:

$$\sum_{(b_1, \ldots, b_\ell) \in \{0,1\}^\ell} f(b_1, \ldots, b_\ell) = H$$

Cost: Quasilinear prover, logarithmic verifier and proof size.

GKR Protocol

Motivation

Goal: Build sumcheck-based version of the circuit arithmetization.

Suppose we are given the **layered** fan-in two arithmetical circuit $\mathsf{C} : \mathbb{F}^n \to \mathbb{F}^m$ of size S (number of gates). The *layered* here means that the circuit C can be decomposed into d layers (note that GKR can be generalized to the unstructured arithmetical circuits as well).

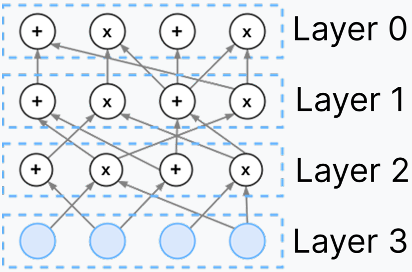


Figure: Layered Circuit Structure of $d = 4$ layers.

Spoilers on Performance

The GKR protocol allows to achieve the following performance:

- The communication consists of $\mathcal{O}(d \cdot \text{polylog}(S))$ field elements.
- The verifier runs in $\mathcal{O}(n + d \cdot \text{polylog}(S))$ time.
- The prover runs in $\mathcal{O}(\text{poly}(S))$ time.
- The soundness error is just $\mathcal{O}(d \log(S)/|\mathbb{F}|)$.

Assumptions:

- Assume we have d rounds in total. Output layer is the 0^{th} layer.
- Each layer consists of S_i gates.
- Assume $S_i = 2^{v_i}$ is the power of two

Concrete Circuit

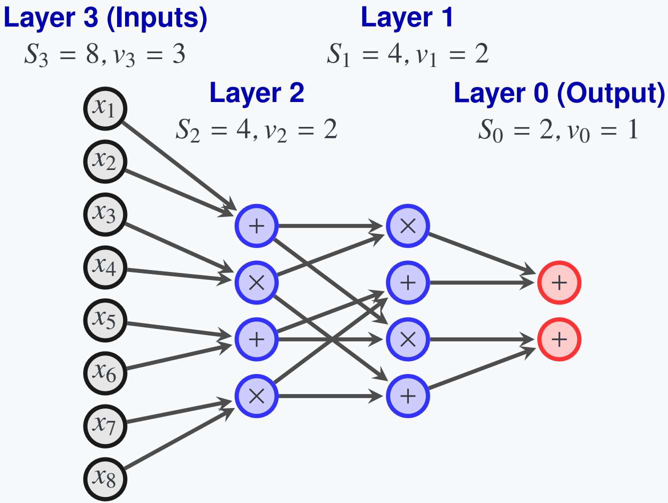
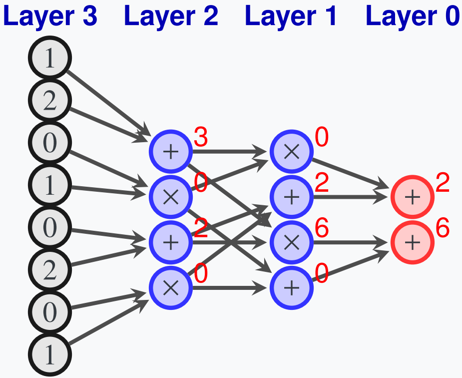


Figure: Example layered arithmetical circuit $C : \mathbb{F}^8 \to \mathbb{F}^2$ with $d = 3$ layers.

Gates Encoding

Gates Encoding. Suppose $W_i : \{0,1\}^{v_i} \to \mathbb{F}$ is structured so that it outputs the value of the i-th layer gate given the gate label. Assume MLE of W_i is $\widetilde{W}_i : \mathbb{F}^{v_i} \to \mathbb{F}$.

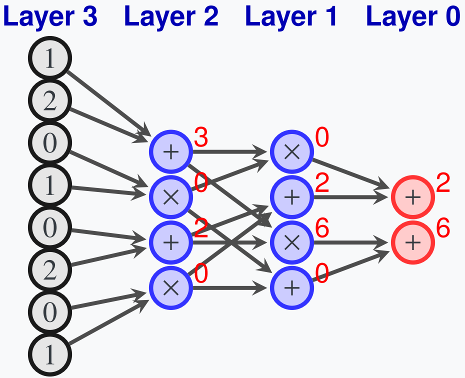


$$W_1(0,0) = 0, \quad W_1(0,1) = 2, \quad W_1(1,0) = 6, \quad W_1(1,1) = 0$$

MLE Extension: $\widetilde{W}_1(X_1, X_2) = 2(1 - X_1)X_2 + 6X_1(1 - X_2)$

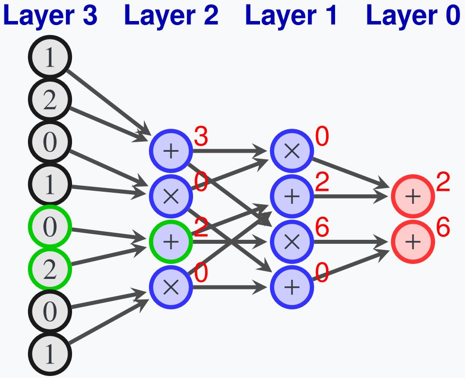
Wiring Encoding

Wiring Predicates. $\text{in}_{1,i}, \text{in}_{2,i} : \{0,1\}^{v_i} \to \{0,1\}^{v_{i+1}}$ indicate which pairs of wiring are connected to the i^{th} layer gate from the layer $i+1$.



Wiring Encoding

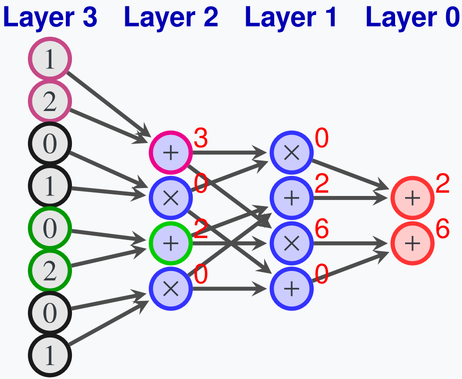
Wiring Predicates. $\mathsf{in}_{1,i}, \mathsf{in}_{2,i} : \{0,1\}^{v_i} \rightarrow \{0,1\}^{v_{i+1}}$ indicate which pairs of wiring are connected to the i^{th} layer gate from the layer $i+1$.



$$\mathsf{in}_{1,2}(1,0) = (1,0,1), \quad \mathsf{in}_{2,2}(1,0) = (1,1,0).$$

Operations Encoding

Operations Encodings. add, mul : $\{0,1\}^{v_i+2v_{i+1}} \to \{0,1\}$:
$\text{add}(a,b,c) = 1 \iff (b,c) = (\text{in}_{1,i}(a), \text{in}_{2,i}(a))$ and a is addition gate



add_2 is non-zero : $((0,0),(0,0,0),(0,0,1))$, $((1,0),(1,0,0),(1,0,1))$.

$\widetilde{\text{add}}_2(\mathbf{X},\mathbf{Y},\mathbf{Z}) = (1-X_1)(1-X_2)(1-Y_1)(1-Y_2)(1-Y_3)(1-Z_1)(1-Z_2)Z_3$

Reducing to Sum-Check

Remark

Note that the operations encodings add_i and mul_i (and thus MLEs $\widetilde{\mathrm{add}}_i$ and $\widetilde{\mathrm{mul}}_i$) do not depend on the solution witness $\{\mathbf{x}^{\langle i \rangle}\}_{i \in [d+1]}$, while the gates encodings W_i do depend.

Idea: Prover sends the claimed value of $\widetilde{W}_0$ (say, $D : \{0,1\}^{v_0} \to \mathbb{F}$), then reduce the claim to the next around with $\widetilde{W}_1$ (in general, prove the reducing from $\widetilde{W}_i$ to $\widetilde{W}_{i+1}$).

Sum-Check Protocol Applied

Lemma

The following statement holds:

$$\widetilde{W}_i(\mathbf{z}) = \sum_{\boldsymbol{b},\boldsymbol{c}\in\{0,1\}^{v_{i+1}}} \Big[\widetilde{\mathsf{add}}_i(\mathbf{z},\boldsymbol{a},\boldsymbol{b})(\widetilde{W}_{i+1}(\boldsymbol{b}) + \widetilde{W}_{i+1}(\boldsymbol{c})) + \widetilde{\mathsf{mul}}_i(\mathbf{z},\boldsymbol{b},\boldsymbol{c})\widetilde{W}_{i+1}(\boldsymbol{b})\widetilde{W}_{i+1}(\boldsymbol{c})\Big].$$

Why Lemma works?

Both sides are multilinear polynomials, thus it suffices to check the equality only over the boolean hypercube $\mathbf{z} \in \{0,1\}^{v_i}$.

Fix $\mathbf{z} = \mathbf{z}_0 \in \{0,1\}^{v_i}$. Without loss of generality, assume $\mathbf{z}_0$ is the addition gate. This way, we reduced the check to:

$$\widetilde{W}_i(\mathbf{z}_0) = \sum_{\boldsymbol{b},\boldsymbol{c}\in\{0,1\}^{v_{i+1}}} \widetilde{\mathsf{add}}_i(\mathbf{z},\boldsymbol{a},\boldsymbol{b})(\widetilde{W}_{i+1}(\boldsymbol{b}) + \widetilde{W}_{i+1}(\boldsymbol{c}))$$

According to $\widetilde{\mathsf{add}}$ definition, the only term that is not zero in the sum is for $(\boldsymbol{b},\boldsymbol{c}) = (\mathsf{in}_{1,i}(\mathbf{z}_0), \mathsf{in}_{2,i}(\mathbf{z}_0))$. Therefore, our sum is:

$$\widetilde{W}_i(\mathbf{z}_0) = \widetilde{W}_{i+1}(\mathsf{in}_{1,i}(\mathbf{z}_0)) + \widetilde{W}_{i+1}(\mathsf{in}_{2,i}(\mathbf{z}_0))$$

Key Procedure

Apply the Sum-Check protocol on the function

$$f_i(\boldsymbol{b},\boldsymbol{c};\mathbf{r}_i) = \widetilde{\mathsf{add}}_i(\mathbf{r}_i,\boldsymbol{b},\boldsymbol{c})(\widetilde{W}_{i+1}(\boldsymbol{b}) + \widetilde{W}_{i+1}(\boldsymbol{c})) + \widetilde{\mathsf{mul}}_i(\mathbf{r}_i,\boldsymbol{b},\boldsymbol{c})\widetilde{W}_{i+1}(\boldsymbol{b})\widetilde{W}_{i+1}(\boldsymbol{c}).$$

Caveat with Sum-Check

Note that the verifier $\mathcal{V}$ does not know $\widetilde{W}_{i+1}$.

In fact, he does not need to until the last round, where he needs to call an oracle access $\mathcal{O}^{f_i}$ at $(\boldsymbol{b}^*, \boldsymbol{c}^*) \leftarrow\$ \mathbb{F}^{2v_{i+1}}$.

This requires evaluating:

- $\widetilde{\mathsf{add}}_i(\mathbf{r}_i, \boldsymbol{b}^*, \boldsymbol{c}^*)$ — can be done by $\mathcal{V}$.
- $\widetilde{\mathsf{mul}}_i(\mathbf{r}_i, \boldsymbol{b}^*, \boldsymbol{c}^*)$ — can be done by $\mathcal{V}$.
- $\widetilde{W}_{i+1}(\boldsymbol{b}^*)$ and $\widetilde{W}_{i+1}(\boldsymbol{c}^*)$ — $\mathcal{V}$ needs $\mathcal{P}$'s assistance.

$\mathcal{P}$ sends two values $z_b = \widetilde{W}_{i+1}(\boldsymbol{b}^*)$ and $z_c = \widetilde{W}_{i+1}(\boldsymbol{c}^*)$. If we had only one value to check, we could use the standard Sum-Check reduction, but here we have two randomnesses!

Line Restriction Trick

Proposition

Let $\ell : \mathbb{F} \to \mathbb{F}^{v_{i+1}}$ be the line such that $\ell(0) = \boldsymbol{b}^*$ and $\ell(1) = \boldsymbol{c}^*$. Then, the prover $\mathcal{P}$ sends the univariate polynomial $q(X)$ claimed to be equal to $\widetilde{W}_{i+1} \circ \ell$ — the restriction of $\widetilde{W}_{i+1}$ to the line ℓ. $\mathcal{V}$ checks whether indeed $\ell(0) = z_b$ and $\ell(1) = z_c$, then chooses a random point $\mathbf{r}^* \xleftarrow{R} \mathbb{F}^{v_{i+1}}$ and checks whether $\widetilde{W}_{i+1}(\ell(\mathbf{r}^*)) = q(\mathbf{r}^*)$.

This way, the interaction ends with new claim about next(previous) layer $\widetilde{W}_{i+1}(\mathbf{r}_{i+1})$ with $\mathbf{r}_{i+1} = \ell(\mathbf{r}^*)$.

In the last round, $\mathcal{V}$ computes $\widetilde{W}_d(\mathbf{r}_d)$ on his own.

Protocol Summary

- $\mathcal{P}$ sends function $D : \{0,1\}^{v_0} \to \mathbb{F}$, claimed to equal W_0.
- $\mathcal{V}$ picks random $\mathbf{r}_0 \leftarrow\!\$\ \mathbb{F}^{v_0}$ and lets $m_0 \leftarrow \widetilde{D}(\mathbf{r}_0)$.
- For each round $i \in [d]$ do the following:
  - Define the $2v_{i+1}$-variate polynomial:

    $$f_i(\boldsymbol{b}, \boldsymbol{c}; \mathbf{r}_i) = \widetilde{\text{add}}_i(\mathbf{r}_i, \boldsymbol{b}, \boldsymbol{c})(\widetilde{W}_{i+1}(\boldsymbol{b}) + \widetilde{W}_{i+1}(\boldsymbol{c})) + \widetilde{\text{mul}}_i(\mathbf{r}_i, \boldsymbol{b}, \boldsymbol{c})\widetilde{W}_{i+1}(\boldsymbol{b})\widetilde{W}_{i+1}(\boldsymbol{c}).$$

  - $\mathcal{P}$ claims $\sum_{\boldsymbol{b},\boldsymbol{c} \in \{0,1\}^{v_{i+1}}} f_i(\boldsymbol{b}, \boldsymbol{c}; \mathbf{r}_i) = m_i$.
  - $\mathcal{P}$ and $\mathcal{V}$ interact using Sum-Check protocol until the last round when $\mathcal{V}$ needs to evalutate f_i at $\boldsymbol{b}^*, \boldsymbol{c}^* \leftarrow\!\$\ \mathbb{F}^{v_{i+1}}$.
  - $\mathcal{P}$ and $\mathcal{V}$ compute the line $\ell : \mathbb{F} \to \mathbb{F}^{v_{i+1}}$ s.t. $\ell(0) = \boldsymbol{b}^*$ and $\ell(1) = \boldsymbol{c}^*$.
  - $\mathcal{P}$ sends q claimed to equal $\widetilde{W}_{i+1} \circ \ell$.
  - $\mathcal{V}$ validates the last round of sum-check using $\ell(0)$ and $\ell(1)$, then chooses $\mathbf{r}^* \leftarrow\!\$\ \mathbb{F}^{v_{i+1}}$ and sets $\mathbf{r}_{i+1} \leftarrow \ell(\mathbf{r}^*)$ and $m_{i+1} \leftarrow q(\mathbf{r}_{i+1})$.
  - The check reduces to verifying $\widetilde{W}_{i+1}(\mathbf{r}_{i+1}) = m_{i+1}$.
- $\mathcal{V}$ directly checks whether $m_d = \widetilde{W}_d(\mathbf{r}_d)$.

Grand Product Check

Grand Product Relation

$$\mathcal{R}_{GP} = \{(p \in \mathbb{F}, \mathbf{v} \in \mathbb{F}^m) : p = \prod_{i=0}^{m} v_i\}$$

Assume that m is a power of 2.

Let $\widetilde{v}$ be an MLE of $\mathbf{v}$, by viewing $\mathbf{v}$ as a function mapping $\{0, 1\}^{\log m} \to \mathbb{F}$.

Main Lemma

Lemma

A scalar p and a vector $\mathbf{v}$ satisfies the relation $\mathcal{R}_{GP}$ if and only if there exists a multilinear polynomial f in $\log m + 1$ variables such that $f(1, \ldots, 1, 0) = p$ and $\forall x \in \{0, 1\}^{\log m}$ the following hold:

$$f(0, x) = v(x)$$
$$f(1, x) = f(x, 0) \cdot f(x, 1)$$

Such polynomial f has the following construction:

- $f(1, \ldots, 1) = 0$
- *For all $\ell \in [\log m]$ and $x \in \{0, 1\}^{\log m - \ell}$:*

$$f(1^{\ell}, 0, x) = \prod_{y \in \{0,1\}^{\ell}} v(x, y)$$

Example

Let $m = 4$ ($\log m = 2$), $\mathbf{v} = \{1, 2, 3, 4\}$, consequently $p = 1 \times 2 \times 3 \times 4 = 24$, then:

$$v(x_1, x_2) = 1 + 2x_1 + x_2$$
$$v(x_1, x_2): \quad v(0,0) = 1,\ v(0,1) = 2,\ v(1,0) = 3,\ v(1,1) = 4.$$

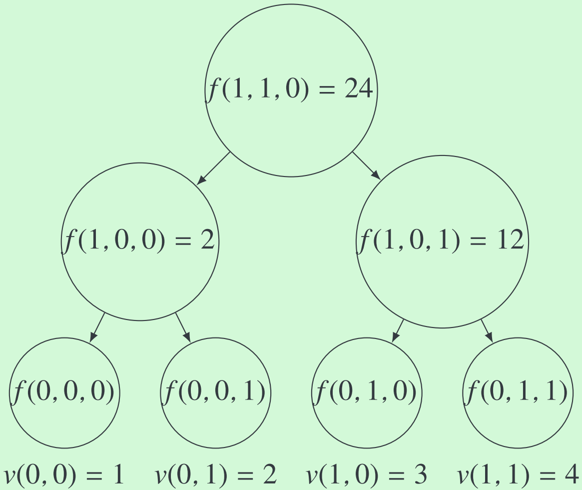
Now, we define f as follows:

$$f(0,0,0) = 1,\ f(0,0,1) = 2,\ f(0,1,0) = 3,\ f(0,1,1) = 4,$$

and:

$$f(1,0,0) = f(0,0,0) \times f(0,0,1) = 1 \times 2 = 2,$$
$$f(1,0,1) = f(0,1,0) \times f(0,1,1) = 3 \times 4 = 12,$$
$$f(1,1,0) = f(1,0,0) \times f(1,0,1) = 2 \times 12 = 24 = p,$$
$$f(1,1,1) = 0.$$

Example



Where Is Sum-Check?

Zero-Check

$$\forall x \in \{0,1\}^{\log m}: f(1,x) = f(x,0) \cdot f(x,1)$$

$$\forall x \in \{0,1\}^{\log m}: f(1,x) - f(x,0) \cdot f(x,1) = 0$$

One can use the sum-check protocol to prove the evaluation of g that is referred to a MLE of $f(1,x) - f(x,0) \cdot f(x,1)$:

$$g(t) = \sum_{x \in \{0,1\}^{\log m}} \widetilde{\mathsf{eq}}(t,x) \cdot (f(1,x) - f(x,0) \cdot f(x,1))$$

By the Schwartz–Zippel lemma, for random $\tau \in \mathbb{F}^{\log m}$, $g(\tau) = 0$ if and only if $g = 0$, except for a soundness error of $\frac{\log m}{|\mathbb{F}|}$.

Thus, to prove the existence of f and hence the grand product relationship, it suffices to prove, for some verifier selected random $\tau, \gamma \in \mathbb{F}^\ell$, that:

$$0 = \sum_{x \in \{0,1\}^{\log m}} \widetilde{\mathrm{eq}}(x, \tau) \cdot (f(1, x) - f(x, 0) \cdot f(x, 1)) \tag{1}$$

$$f(0, \gamma) = \widetilde{v}(\gamma) \tag{2}$$

$$f(1, \ldots, 1, 0) = p \tag{3}$$

Algorithm 1: Grand Product Check

$\mathcal{P}$: Compute polynomials $v \in \mathbb{F}^{\log m}[x], f \in \mathbb{F}^{\log m+1}[x]$ such that
$$p = \prod_{x \in \{0,1\}^{\log m}} v(x) \quad \text{and} \quad f, v \text{ satisfy } (1), (2), (3).$$
$\mathcal{P}$: $C_f \leftarrow \textsc{Commit}(f)$; $C_v \leftarrow \textsc{Commit}(v)$; send C_f, C_v to $\mathcal{V}$.
$\mathcal{V}$: Choose random $\tau, \gamma \in \mathbb{F}^{\log m}$ and send them to $\mathcal{P}$.
$\mathcal{P}$: Compute $g(x) = \widetilde{\text{eq}}(x, \tau)\left(f(1, x) - f(x, 0) f(x, 1)\right)$.
$\mathcal{P}$ & $\mathcal{V}$: Run $\textsc{SumCheckProtocol}(0, g, C_f)$
$\mathcal{V}$: $a \leftarrow \textsc{Query}(C_f, (0, \gamma))$, $v(\gamma) \leftarrow \textsc{Query}(C_v, \gamma)$.
if $a \neq v(\gamma)$ **then**
  $\mathcal{V}$ **rejects**.
end
$\mathcal{V}$: $r \leftarrow \textsc{Query}(C_f, (1, \ldots, 1, 0))$.
if $r \neq p$ **then**
  $\mathcal{V}$ **rejects**.
end

Randomized Permutation Check

The goal is to verify, with high probability, whether two sequences of tuples are permutations of each other, without performing a full sort or pairwise comparison.

$$A = \{(1, 2, 3),\ (4, 0, 6)\}, \quad B = \{(4, 0, 6),\ (1, 2, 3)\}.$$

The naive approach would be to sort both sequences and then compare them.

Reed-Solomon Fingerprinting

Definition (Reed-Solomon Fingerprinting)

Let $\mathbf{a} \in \mathbb{F}^n$, then for a random $\gamma \in \mathbb{F}$, the Reed-Solomon fingerprinting of $\mathbf{a}$ is defined as:

$$h_\gamma(\mathbf{a}) = \sum_{i \in n} a_i \cdot \gamma^i.$$

$h_\gamma(\mathbf{a})$ uniquely identifies the sequence $\mathbf{a}$ with high probability, i.e., let $\mathbf{b} \in F^n$ and $\mathbf{a} \neq \mathbf{b}$, then, according to the Schwartz-Zippel lemma:

$$\Pr[h_\gamma(\mathbf{a}) = h_\gamma(\mathbf{b})] \leq \frac{n}{|\mathbb{F}|}.$$

Reed-Solomon Fingerprinting

Example

Consider all operations in $\mathbb{F}_7$, and set $n = 3$, $\gamma = 3$. Let

$$\mathbf{a} = (1, 2, 3), \quad \mathbf{b} = (4, 0, 6).$$

Then

$$h_\gamma(1, 2, 3) = 1 \cdot 3^0 + 2 \cdot 3^1 + 3 \cdot 3^2 = 1 + 6 + 6 = 13 \equiv 6,$$
$$h_\gamma(4, 0, 6) = 4 \cdot 1 + 0 \cdot 3 + 6 \cdot 2 = 4 + 0 + 12 = 16 \equiv 2.$$

Randomized Permutation Check

Definition (Randomized Permutation Check)

Let A and B be two multisets of tuples in $\mathbb{F}^n$. Define

$$\mathcal{H}_{\tau,\gamma}(X) = \prod_{x \in X} (h_\gamma(x) - \tau).$$

Then comparing $\mathcal{H}_{\tau,\gamma}(A)$ and $\mathcal{H}_{\tau,\gamma}(B)$ yields a randomized test for whether A and B are permutations of one another. Concretely:

- (Completeness) If $A = B$ (as multisets), then

$$\mathcal{H}_{\tau,\gamma}(A) = \mathcal{H}_{\tau,\gamma}(B)$$

  with probability 1 over uniform $\tau, \gamma \in \mathbb{F}$.

- (Soundness) If $A \neq B$, then

$$\Pr\left[\mathcal{H}_{\tau,\gamma}(A) = \mathcal{H}_{\tau,\gamma}(B)\right] \le \frac{\max(|A|, |B|)}{|\mathbb{F}|}.$$

Example

Consider all operations in $\mathbb{F}_7$, set $n = 3$, $\tau = 5$, and $\gamma = 3$.
Let $A = \{(1, 2, 3),\ (4, 0, 6)\}$, $\quad B = \{(4, 0, 6),\ (1, 2, 3)\}$.

First compute the Reed–Solomon fingerprints modulo 7:

$$h_\gamma(1, 2, 3) = 1 \cdot 3^0 + 2 \cdot 3^1 + 3 \cdot 3^2 = 1 + 6 + 6 = 13 \equiv 6,$$
$$h_\gamma(4, 0, 6) = 4 \cdot 1 + 0 \cdot 3 + 6 \cdot 2 = 4 + 0 + 12 = 16 \equiv 2.$$

Now form the shifted products:

$$\mathcal{H}_{\tau,\gamma}(A) = (6 - 5)(2 - 5) = 1 \cdot (-3) \equiv 4,$$
$$\mathcal{H}_{\tau,\gamma}(B) = (2 - 5)(6 - 5) = (-3) \cdot 1 \equiv 4,$$

so the test *accepts* A vs. B (they are indeed permutations).

Now consider a non-permutation $B' = \{(1, 2, 3),\ (2, 1, 3)\}$.

$$h_\gamma(2, 1, 3) = 2 \cdot 1 + 1 \cdot 3 + 3 \cdot 2 = 2 + 3 + 6 = 11 \equiv 4.$$
$$\mathcal{H}_{\tau,\gamma}(B') = (6 - 5)(4 - 5) = 1 \cdot (-1) \equiv 6 \neq 4,$$

so the test *rejects* A vs. B'.

Offline Memory Checking

Motivation

Consider Alice, who stores two values on Bob's dedicated server at addresses 0 and 1. Initially, Bob's memory contains

$$M = \{(0, 100),\ (1, 200)\}.$$

Alice then performs the following operations in sequence:

1. writes 150 at address 0. Bob updates his memory to $\{(0, 150), (1, 200)\}$.
2. read from address 0 and obtains the reply 150.
3. reads from address 1 and (honestly) obtains 200.

However, Bob can cheat on the very last step by returning a wrong value:

$$(1, 200) \longrightarrow (1, 300).$$

Without keeping an auditable record of all reads, Alice cannot later prove that all replies came from correct memory contents.

Memory Model

Each **memory cell** can be described as a tuple $(\text{addr}, \text{val}, \text{counter})$.

- addr is the address of the memory cell;
- val is the value stored at that address;
- counter is a counter that is incremented each time the value at that address is written to.

The protocol utilizes four sets of tuples:

- **init** – contains the initial memory state;
- **write** – contains memory cels that represent write operations;
- **read** – contains memory cels that represent read operations;
- **final** – contains the final memory state, where all counters are set to the last value.

Memory Operations

- **Init:** load initial state; all counters = 0;
- **Read:**
  - Query untrusted memory at addr → (val,counter);
  - Append (addr,val,counter) to **reads**;
  - Append (addr,val,counter+1) to **writes**.
- **Write:**
  - Query untrusted memory at addr → (val,counter);
  - Append (addr,val,counter) to **reads**;
  - Append (addr,**newval**,counter+1) to **writes**.

Consistency Check

After all reads and writes are done, the **final** set is populated with the final memory state.

Lemma

One can check the consistency of the memory operations by verifying that:

$$\mathbf{read} \cup \mathbf{final} = \mathbf{write} \cup \mathbf{init}.$$

Equivalently, via randomized permutation check:

$$\mathcal{H}_{\tau,\gamma}(\mathbf{read}) \cdot \mathcal{H}_{\tau,\gamma}(\mathbf{final}) = \mathcal{H}_{\tau,\gamma}(\mathbf{write}) \cdot \mathcal{H}_{\tau,\gamma}(\mathbf{init}).$$

Example

Suppose the initial memory is:

$$\textbf{init} = \{(0,2,0),\ (1,5,0),\ (2,7,0),\ (3,9,0)\},$$

while $\textbf{read} = \varnothing$ and $\textbf{write} = \varnothing$.

step	operation	$\Delta\textbf{read}_{\text{step}}$	$\Delta\textbf{write}_{\text{step}}$
1	read(1) $\to$ $(5,0)$	$(1,5,0)$	-
2	write((1,6))	-	$(1,6,1)$
3	read(2) $\to$ $(7,0)$	$(2,7,0)$	-
4	write((2,7))	-	$(2,7,1)$

$$\textbf{read} = \{(1,5,0),\ (2,7,0)\}, \quad \textbf{write} = \{(1,6,1),\ (2,7,1)\}$$
$$\textbf{final} = \{(0,2,0),\ (1,6,1),\ (2,7,1),\ (3,9,0)\}$$

One can clearly see that $\textbf{read} \cup \textbf{final} = \textbf{write} \cup \textbf{init}$.

$$\{(\mathbf{1},\mathbf{5},\mathbf{0}),(\mathbf{2},\mathbf{7},\mathbf{0})\} \cup \{(0,2,0),(\mathbf{1},\mathbf{6},\mathbf{1}),(\mathbf{2},\mathbf{7},\mathbf{1}),(3,9,0)\} =$$
$$= \{(\mathbf{1},\mathbf{6},\mathbf{1}),(\mathbf{2},\mathbf{7},\mathbf{1})\} \cup \{(0,2,0),(\mathbf{1},\mathbf{5},\mathbf{0}),(\mathbf{2},\mathbf{7},\mathbf{0}),(3,9,0)\}$$

Example

Suppose the initial memory is:

$$\mathbf{init} = \{(0,2,0),\ (1,5,0),\ (2,7,0),\ (3,9,0)\},$$

while $\mathbf{read} = \varnothing$ and $\mathbf{write} = \varnothing$.

step	operation	$\Delta\mathbf{read}_{\text{step}}$	$\Delta\mathbf{write}_{\text{step}}$
1	read(1) $\to$ $(a,0)$	$(1,a,0)$	-
2	write((1,6))	-	$(1,6,1)$

$$\mathbf{read} = \{(1,a,0)\}, \quad \mathbf{write} = \{(1,6,1)\},$$
$$\mathbf{final} = \{(0,2,0),\ (1,6,1),\ (2,7,1),\ (3,9,0)\}$$

The verifier checks $\mathbf{read} \cup \mathbf{final} = \mathbf{write} \cup \mathbf{init}$:

$$\{(1,a,0)\} \cup \{(0,2,0),(1,6,1),(2,7,0),(3,9,0)\} \neq$$
$$\neq \{(1,6,1)\} \cup \{(0,2,0),(1,5,0),(2,7,0),(3,9,0)\}$$

and *rejects*.

Thank you for your attention

♥

zkdl-camp.github.io
github.com/ZKDL-Camp