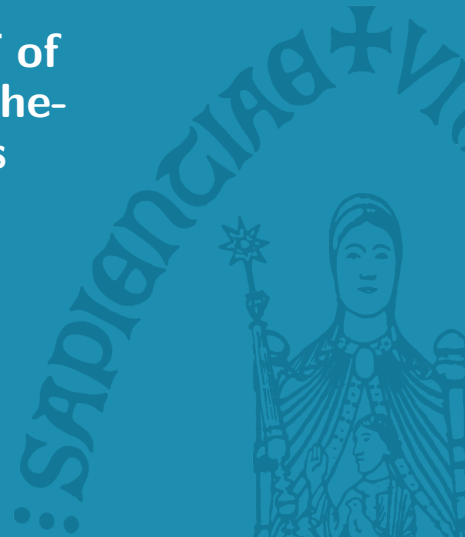
KU LEUVEN

Efficient Non-Interactive Proof of RAM Programs with MPC-in-the-Head Zero-Knowledge Systems

TPMPC 2022, Aarhus

Cyprien de Saint Guilhem Emmanuela Orsini
Titouan Tanguy Michiel Verbauwhede
imec-COSIC, KU Leuven



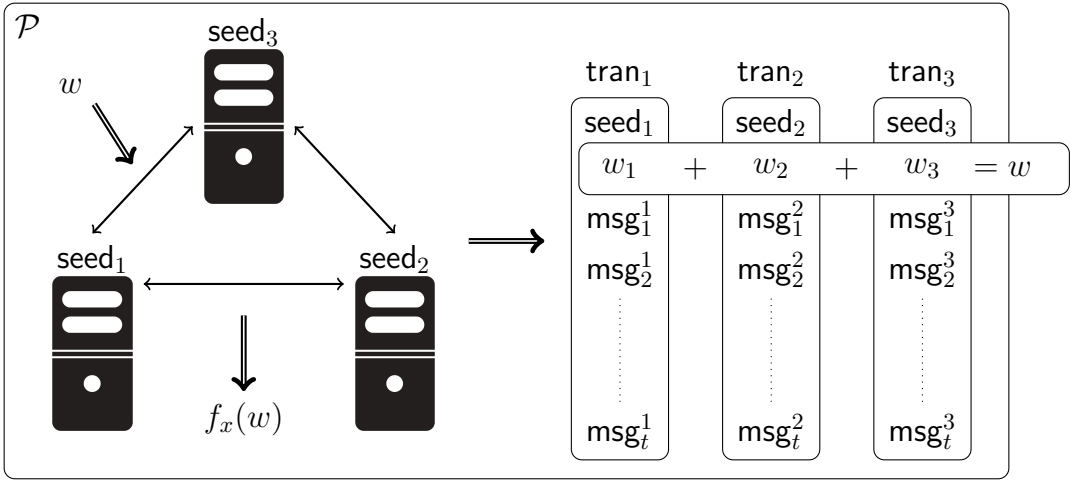
0 Outline

1. MPC-in-the-Head
2. Constant-Overhead from RAM Access Verification
3. Public-coin Verification of RAM Access

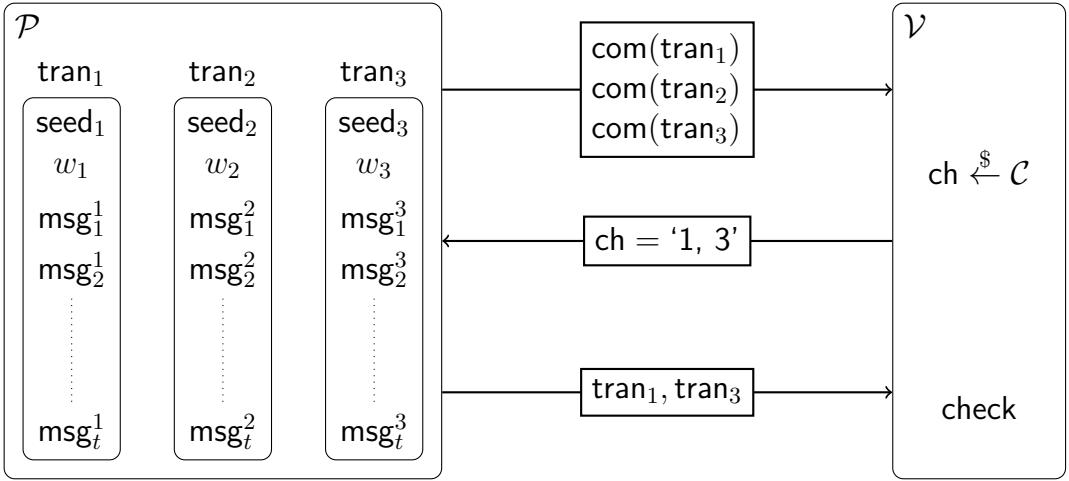
1 Table of Contents

1. MPC-in-the-Head
2. Constant-Overhead from RAM Access Verification
3. Public-coin Verification of RAM Access

1 MPC in the head?



1 ZKPoK from MPC in the head



1 MPC-in-the-Head from Computation

Original idea: $\mathcal{P}$ simulates a protocol that *computes* $C(x, w)$.

IKOS (2007)	Theoretical
ZKBoo (2016), ZKB++ (2017, Picnic v1)	Boolean
KKW (2018, Picnic v2, Picnic v3 in 2020)	Boolean
BBQ (2019, "Picnic with AES")	Arithmetic (SPDZ-style)

1 MPC-in-the-Head from Verification

ZKP advantage

$\mathcal{P}$ can already compute $C(x, w)$, MPC protocol just has to *verify* $C(x, w)$.

		Verification
BN (2020)	Arithmetic	randomness
TurboIKOS (2021)	Arithmetic	randomness
Banquet (2021)	Arithmetic	multiplication
Limbo (2021)	Arithmetic	multiplication

ZK proof of AES in 2.7ms.

Banquet: 6.34ms, Picnic3: 5.33ms for LowMC (90% smaller)

What next?

What next?

RAM

2 Table of Contents

1. MPC-in-the-Head
2. Constant-Overhead from RAM Access Verification
3. Public-coin Verification of RAM Access

2 Constant Overhead: Checking RAM Accesses

Franzese et al. (CCS 2021)

Similar to KKW $\longrightarrow$ BN improvement: transition from *computing* to *checking*. (Because computing permutation network is $n \log n$.)

2 Constant Overhead: Checking RAM Accesses

Franzese et al. (CCS 2021)

Similar to KKW $\longrightarrow$ BN improvement: transition from *computing* to *checking*. (Because computing permutation network is $n \log n$.)

Input $\xleftarrow{\mathcal{P}}$ $\boxed{M_1 \quad \cdots \quad M_N}$

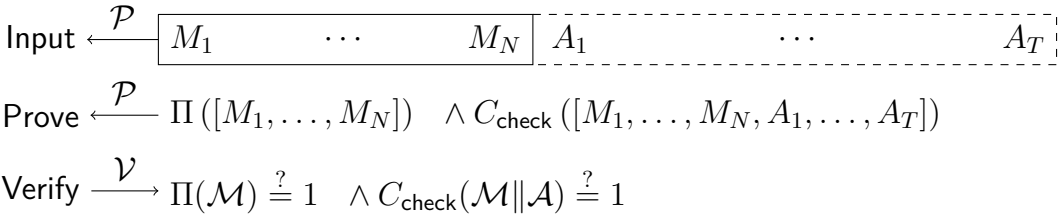
Prove $\xleftarrow{\mathcal{P}}$ $\Pi\left([M_1, \ldots, M_N]\right)$

Verify $\xrightarrow{\mathcal{V}}$ $\Pi(\mathcal{M}) \stackrel{?}{=} 1$

2 Constant Overhead: Checking RAM Accesses

Franzese et al. (CCS 2021)

Similar to KKW $\longrightarrow$ BN improvement: transition from *computing* to *checking*. (Because computing permutation network is $n \log n$.)



2 Constant Overhead: Checking RAM Accesses

Franzese et al. (CCS 2021)

Let $\mathcal{L} = \mathcal{M} \| \mathcal{A}$ be a list of $N + T$ tuples $(\ell_i, t_i, \mathsf{op}_i, d_i)$.

Need to check that $\mathcal{L}$ is *consistent*:

1. Every ℓ_i is first written to, before being read;
2. Each Read operation returns the value last written to the address.

2 Constant Overhead: Checking RAM Accesses

Franzese et al. (CCS 2021)

Let $\mathcal{L} = \mathcal{M} \| \mathcal{A}$ be a list of $N + T$ tuples $(\ell_i, t_i, \mathsf{op}_i, d_i)$.

Need to check that $\mathcal{L}$ is *consistent*:

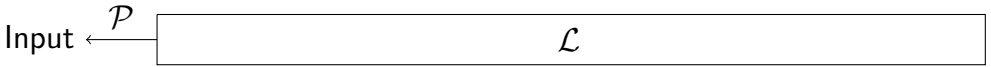
1. Every ℓ_i is first written to, before being read;
2. Each Read operation returns the value last written to the address.

What does C_{check} do?

1. Accepts $\mathcal{L}'$ as input: permutation of $\mathcal{L}$ sorted by ℓ and by t;
2. Checks that $\mathcal{L}'$ is a permutation of $\mathcal{L}$;
3. Checks that ordering in $\mathcal{L}'$ is correct;
4. Checks consistency requirements.

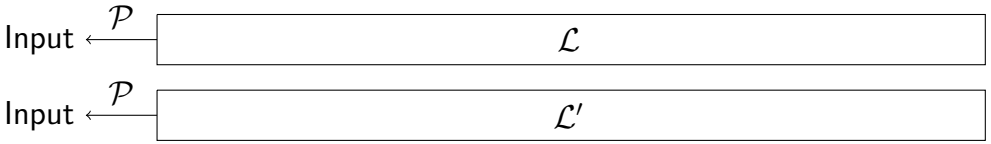
2 Constant Overhead: Checking RAM Accesses

Franzese et al. (CCS 2021)



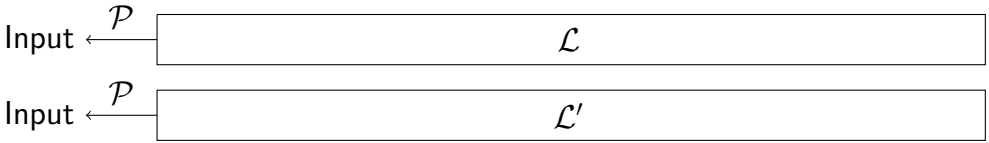
2 Constant Overhead: Checking RAM Accesses

Franzese et al. (CCS 2021)



2 Constant Overhead: Checking RAM Accesses

Franzese et al. (CCS 2021)



Prove Boolean $\leftarrow \mathcal{P}$

For $i \in [1, N + T - 1]$:

$$(\ell'_i < \ell'_{i+1}) \vee \Big((\ell'_i = \ell'_{i+1}) \wedge (t'_i < t'_{i+1})\Big) \bigwedge$$

$$(\ell'_i \neq \ell'_{i+1}) \vee (d'_i = d'_{i+1}) \vee (\mathsf{op}'_{i+1} = \mathsf{write}) \bigwedge$$

$$(\ell'_i = \ell'_{i+1}) \vee (\mathsf{op}'_{i+1} = \mathsf{write})$$

2 Constant Overhead: Checking RAM Accesses

Franzese et al. (CCS 2021)

Arithmetic check of permutation: if $[b_1, \ldots, b_n] = \pi([a_1, \ldots, a_n])$, then

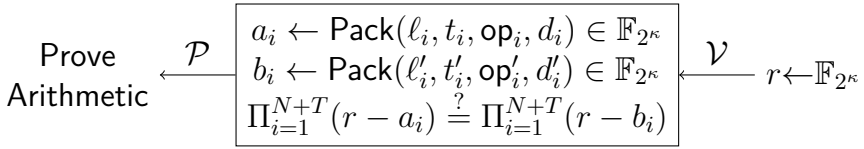
$$\Pi_{i=1}^{n}(x - b_i) = \Pi_{i=1}^{n}(x - a_i).$$

2 Constant Overhead: Checking RAM Accesses

Franzese et al. (CCS 2021)

Arithmetic check of permutation: if $[b_1, \ldots, b_n] = \pi([a_1, \ldots, a_n])$, then

$$\Pi_{i=1}^{n}(x - b_i) = \Pi_{i=1}^{n}(x - a_i).$$

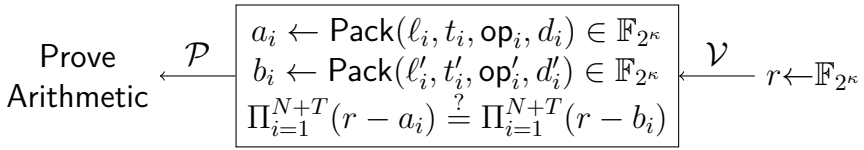


2 Constant Overhead: Checking RAM Accesses

Franzese et al. (CCS 2021)

Arithmetic check of permutation: if $[b_1, \ldots, b_n] = \pi([a_1, \ldots, a_n])$, then

$$\Pi_{i=1}^{n}(x - b_i) = \Pi_{i=1}^{n}(x - a_i).$$



- Efficient conversion Boolean $\leftrightarrow$ Arithmetic requires **private coins**
- Efficient packing only works in **binary** fields.

3 Table of Contents

1. MPC-in-the-Head
2. Constant-Overhead from RAM Access Verification
3. Public-coin Verification of RAM Access

3 Public-coin Constant Overhead ZK RAM Access

We design C_{check} for arbitrary $\mathbb{F}$, *without conversion*.

1. EqCheck circuit for equality check.
2. PermCheck circuit for permutation (same as Franzese et al.). Inner-product compression instead of packing
3. BdCheck circuit for inequalities: $0 < t'_{i+1} - t'_i \implies t'_i < t'_{i+1}$.

3 Public-coin Constant Overhead ZK RAM Access

We design C_{check} for arbitrary $\mathbb{F}$, *without conversion*.

1. EqCheck circuit for equality check.
2. PermCheck circuit for permutation (same as Franzese et al.). Inner-product compression instead of packing
3. BdCheck circuit for inequalities: $0 < t'_{i+1} - t'_i \implies t'_i < t'_{i+1}$.

- Can make *non-interactive* with any public-coin ZK system.

3 EqCheck: Equality in ZK

1: $\texttt{Input}\ \langle r \rangle = \begin{cases} (\langle x \rangle - \langle y \rangle)^{-1} & \text{if } x \neq y \\ \text{random non-zero} & \text{if } x = y \end{cases}$

2: $\texttt{Input}\ \langle r^{-1} \rangle$

3: Check that $\langle r \rangle \cdot \langle r^{-1} \rangle$ is equal to 1. ▷ Asserts $r \neq 0$

4: $\langle b \rangle \leftarrow (\langle x \rangle - \langle y \rangle) \cdot \langle r \rangle$

5: Check that $(1 - \langle b \rangle) \cdot \langle b \rangle$ is equal to 0. ▷ Asserts $b \in \{0, 1\}$

6: **return** $1 - \langle b \rangle$

3 PermCheck without packing

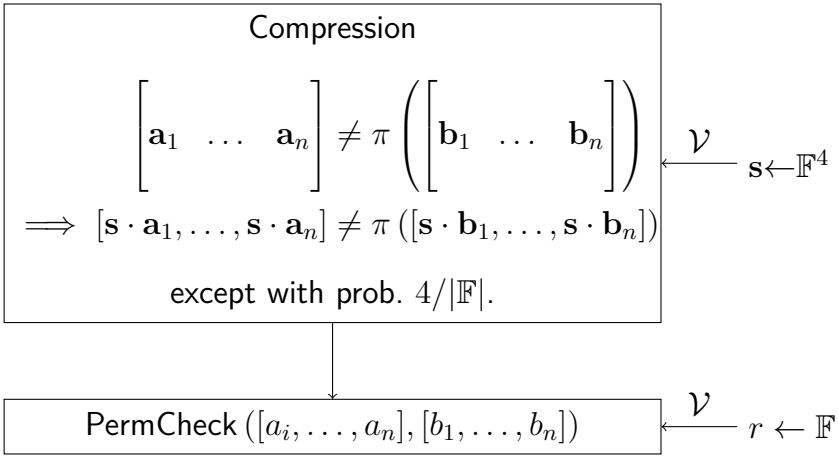
Compression

$$\begin{bmatrix} \mathbf{a}_1 & \dots & \mathbf{a}_n \end{bmatrix} \neq \pi \left(\begin{bmatrix} \mathbf{b}_1 & \dots & \mathbf{b}_n \end{bmatrix} \right) \xleftarrow{\mathcal{V}} \mathbf{s} \leftarrow \mathbb{F}^4$$

$$\implies [\mathbf{s} \cdot \mathbf{a}_1, \dots, \mathbf{s} \cdot \mathbf{a}_n] \neq \pi \left([\mathbf{s} \cdot \mathbf{b}_1, \dots, \mathbf{s} \cdot \mathbf{b}_n] \right)$$

except with prob. $4/|\mathbb{F}|$.

3 PermCheck without packing



3 Constant Overhead Bound Test

This algorithm tests that $A \leq x_i \leq B$ for $\{x_i\}_{i=1}^{T}$:

1: Arrange array $\langle \mathcal{L} \rangle = [A, A+1, \ldots, B, \langle x_1 \rangle, \langle x_2 \rangle, \ldots, \langle x_T \rangle]$ of size $S = B_2 - B_1 + 1 + T$.
2: $\texttt{Input}\langle \mathcal{L}' \rangle$ containing $\mathcal{L}$ sorted from lowest to highest.
3: PermCheck$(\langle \mathcal{L} \rangle, \langle \mathcal{L}' \rangle)$
4: **for** $i \in [S-1]$ **do**
5: $\quad \langle \alpha_i \rangle \leftarrow$ EqCheck$(\langle \mathcal{L}'[i] \rangle, \langle \mathcal{L}'[i+1] \rangle)$
6: $\quad \langle \beta_i \rangle \leftarrow \langle \mathcal{L}'[i+1] \rangle - \langle \mathcal{L}'[i] \rangle$
7: $\quad$ Check that $\langle \alpha_i \rangle + \langle \beta_i \rangle$ is equal to 1.
8: Check that $\langle \mathcal{L}'[1] \rangle = A$ and that $\langle \mathcal{L}'[S] \rangle = B$.

3 Comparison

Scheme	Coins	Structure	per access		
			Complexity	Time (ms)	Comms (KB)
BubbleRAM[(1)]	Private	$GF(2^{64}-59)$[(2)]	$O(\log^2 N)$	0.15	1.5
Franzese et al.	Private	$GF(2^{40}-87)$	$O(1)$	0.01	0.031
Ours	Public	$GF(2^{61}-1)$	$O(1)$	0.2–1.6	4.1–1.8

(1) $N = 2^{18}$.

(2) But only computes on 32-bit words.

500kB of RAM in ≈ 15s (Apollo mission used 4kB RAM and 72kB ROM).

3 Links

Limbo ia.cr/2021/215

RAM gates ia.cr/2022/313