Dora

Processor Expressiveness Comes (Nearly) Free in Zero-Knowledge for RAM Programs

Aarushi Goel

NTTResearch

Mathias Hall-Andersen

AARHUS UNIVERSITY

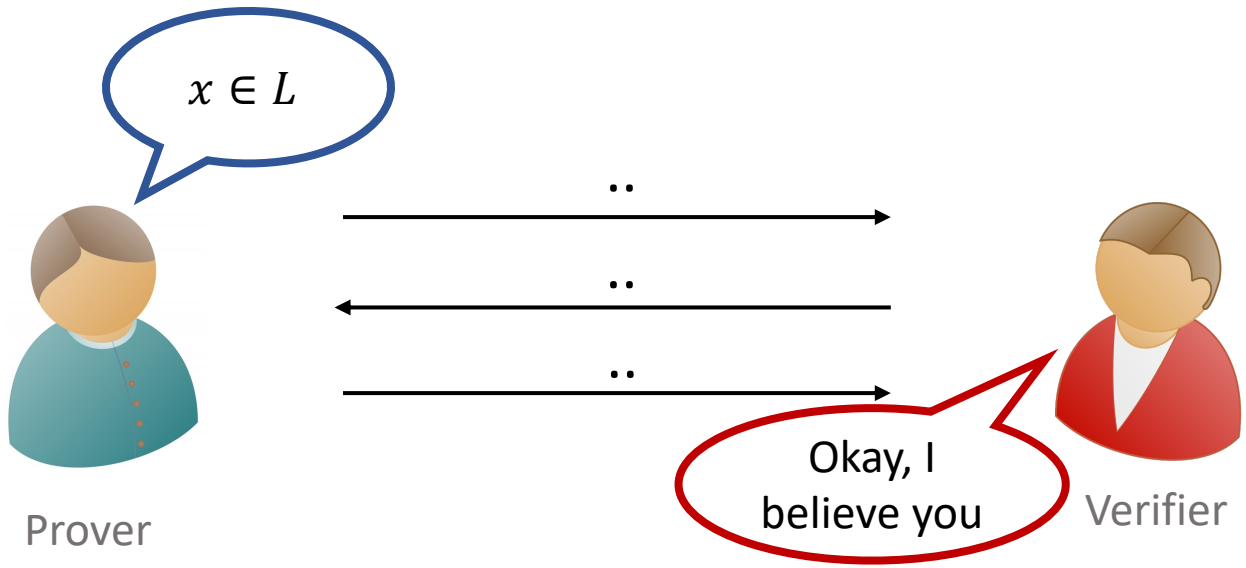
Gabriel Kaptchuk

BOSTON UNIVERSITY

Stanford Security Seminar; 11 January 2024

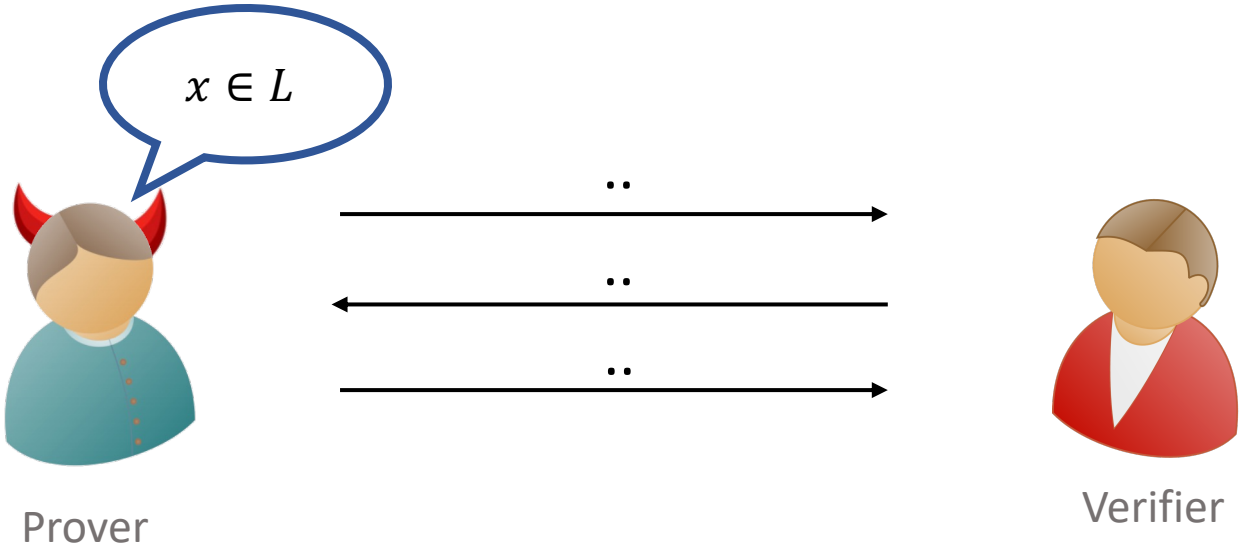
Zero-Knowledge Proofs [GMR85]

$L \in NP$



Zero-Knowledge Proofs [GMR85]

$L \in NP$



Soundness

Cheating prover cannot give accepting proof if $x \notin L$

Zero knowledge

Verifier should not learn the secret witness

Zero-Knowledge for RAM Programs

A 2.1 KHz Zero-Knowledge Processor with BubbleRAM

David Heath and Vladimir Kolesnikov
heath.davidanthony@gatech.edu, kolesnikov@gatech.edu

Abstract

SNARKs for C:
Verifying Program Executions Succinctly and in Zero Knowledge
(extended version)

Eli Ben-Sasson[1], Alessandro Chiesa[2], Daniel Genkin[2], Eran Tromer[3] and Madars Virza[2]

[1] Technion, {eli,danielg3}@cs.technion.ac.il
[2] MIT, {alexch,madars}@csail.mit.edu
[3] Tel Aviv University, tromer@cs.tau.ac.il

October 7, 2013

Abstract

An argument system for NP is a proof system that allows efficient verification of NP statements, given proo
interactive
anyone car
and proofs
We pre
system, m

Sublinear Zero-Knowledge Arguments for RAM Programs

Payman Mohassel[1], Mi

Efficient Proof of RAM Programs from Any Public-Coin Zero-Knowledge System

ini Titouan Tanguy

of RAM programs using any
work is an arithmetic circuit
zero-knowledge.
retely efficient zero-knowledge
lge proof system for Boolean
prover and verifier run times
time of the RAM program;
C++ implementation.

PrORAM
Fast $O(\log n)$ Authenticated Shares ZK ORAM

David Heath and Vladimir Kolesnikov

Georgia Institute of Technology, Atlanta, GA, USA
{heath.davidanthony,kolesnikov}@gatech.edu

Succinct Non-Interactive Zero Knowledge for a von Neumann Architecture

Eli Ben-Sasson Technion
Alessandro Chiesa MIT
Eran Tromer Tel Aviv University
Madars Virza MIT

y 5, 2019
d version)

stract
teractive zero-knowledge proofs (*zk-SNARKs*) for pro-

Efficient RAM and control flow in verifiable outsourced computation

Riad S. Wahby*, Srinath Setty†, Max Howald‡, Zuocheng Ren†,
Andrew J. Blumberg†, and Michael Walfish*

*New York University
{rsw,mwalfish}@cs.nyu.edu

†The University of Texas at Austin
{srinath@cs,ren@cs,blumberg@math}.utexa

‡The Cooper Union

Abstract. Recent work on *proof-based verifiable computation* has r
tools from complexity theory and cryptography to address a basic pr
a local computer to outsource the execution of a program while pr
guarantee of integrity and the remote computer with a guarantee of pri
that use RAM and control flow has been problematic. State of the art s
constructs (e.g., requiring static loop bounds), incur sizeable overhea

Verifying computations with state (extended version)

Benjamin Braun, Ariel J. Feldman*, Zuocheng Ren,
Srinath Setty, Andrew J. Blumberg, and Michael Walfish

The University of Texas at Austin *University of Pennsylvania

Nearly Linear-Time Zero-Knowledge Proofs for Correct Program Execution*

Jonathan Bootle, Andrea Cerulli, Jens Groth, Sune Jakobsen, Mary Maller **

Univ

Abstract. There have been
communication and verific
remains an important challe
the first zero-knowledge pr

Constant-Overhead Zero-Knowledge for RAM Programs

Nicholas Franzese* Jonathan Katz† Steve Lu‡ Rafail Ostrovsky§ Xiao Wang*
Chenkai Weng*

Efficient Zero-Knowledge Proofs of Non-Algebraic Statements with Sublinear Amortized Cost*

Zhangxiang Hu† Payman Mohassel‡ Mike Rosulek†

May 26, 2015

Abstract

We describe a zero-knowledge proof system in which a prover holds a large dataset M and can repeatedly prove NP relations about that dataset. That is, for any (public) relation R and x, the prover can prove that $\exists w : R(M, x, w) = 1$. After an initial setup phase (which depends only on M), each proof requires only a constant number of rounds and has communication/computation cost proportional to that of a *random-access machine (RAM)* implementation of R, up to poly-logarithmic factors. In particular, the cost per proof in many applications is sublinear in $|M|$. Additionally, the storage requirement between proofs for the verifier is constant.

Public-Coin Zero-Knowledge Arguments with (almost) Minimal Time and Space Overheads*

Alexander R. Block† Justin Holmgren‡ Alon Rosen§ Ron D. Rothblum¶
Pratik Soni‖

Abstract

Zero-knowledge protocols enable the truth of a mathematical statement to be certified by a verifier without revealing any other information. Such protocols are a cornerstone of modern cryptography and recently are becoming more and more practical. However, a major bottleneck in deployment is the efficiency of the prover and, in particular, the space-efficiency of the protocol.

For every NP relation that can be verified in time T and space S, we construct a public-coin zero-knowledge argument in which the prover runs in time $T \cdot \text{polylog}(T)$ and space $S \cdot \text{polylog}(T)$. Our proofs have length $\text{polylog}(T)$ and the verifier runs in time $T \cdot \text{polylog}(T)$ (and space

Time- and Space-Efficient Arguments from Groups of Unknown Order*

Alexander R. Block† Justin Holmgren‡ Alon Rosen§ Ron D. Rothblum¶
Pratik Soni‖

September 21, 2021

Abstract

We construct public-coin time- and space-efficient zero-knowledge arguments for **NP**. For every time T and space S non-deterministic RAM computation, the prover runs in time $T \cdot \text{polylog}(T)$ and space $S \cdot \text{polylog}(T)$, and the verifier runs in time $n \cdot \text{polylog}(T)$, where n is the input length. Our protocol relies on hidden order groups, which can be instantiated with a trusted setup from the hardness of factoring (products of safe primes), or without a trusted setup using class groups. The argument-system can heuristically be made non-interactive using the Fiat-Shamir transform.

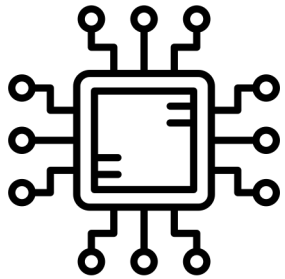
Our proof builds on DARK (Bünz et al., Eurocrypt 2020), a recent succinct and efficiently verifiable

act
-knowledge (ZK) proof system for RAM pro-
ation complexity as well as the running times
e of the memory N and the running time T
g an asymptotic improvement of prior work,
mprovements for RAM-based ZK proofs. In
ofs of private read/write accesses to 64 MB
s of communication per access, a more than
RAM protocol. We also design a lightweight
I instruction set, and for which our ZK proof

ZK for RAM Programs: Two Approaches

Transform source code representation of a RAM program directly into a circuit and then use a circuit-based zero-knowledge proof

```
1: static const char* SMALL_BOARD = "small_board_v11";
2: int* alloc_resources(const char* board_type) {
3:    int block_size;
4:    // The next line has a bug!!
5:    if (!strncmp(board_type, SMALL_BOARD, sizeof(SMALL_BOARD))) {
6:       block_size = 10;
7:    } else { block_size = 100; }
8:    return malloc(block_size * sizeof(int));  }
9: int incr_clock(const char* board_type, int* resources) {
10:    int clock_loc;
11:    if (!strncmp(board_type, SMALL_BOARD, strlen(SMALL_BOARD))) {
12:       clock_loc = 0;
13:    } else { clock_loc = 64 }
14:    (*(resources + clock_loc))++;
15:    return resources[clock_loc];  }
16: void snippet(const char* board_type) {
17:    int* res = alloc_resources(board_type);
18:    incr_clock(board_type, res);  }
```



Design custom zero-knowledge proofs to emulate RAM computations (eg. [BCG+13])

TinyRAM Architecture Specification
v2.000

Eli Ben-Sasson
StarkWare
eli@starkware.co

Alessandro Chiesa
UC Berkeley
alexch@berkeley.edu

Daniel Genkin
University of Michigan
genkin@umich.edu

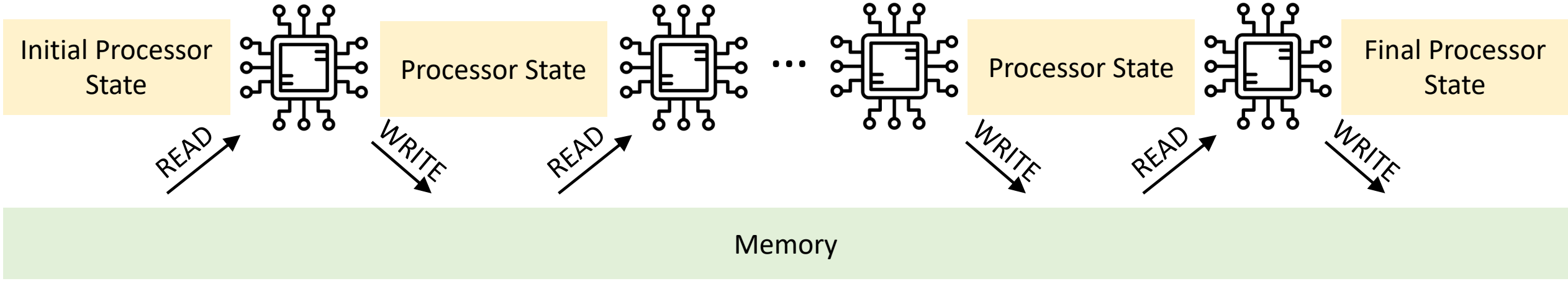
Eran Tromer
Tel Aviv University and Columbia University
tromer@cs.tau.ac.il

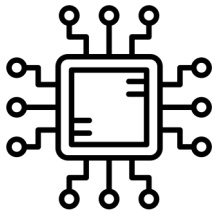
Madars Virza
MIT
madars@mit.edu

SCIPR Lab
scipr-lab.org

March 30, 2020

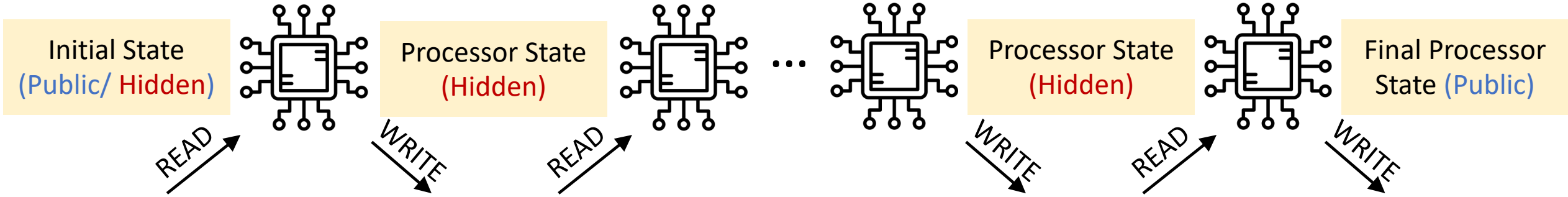
Emulating RAM (in more detail)





Contains multiple instructions, one only of which is executed at each step

Emulating RAM (in more detail)



Need to Prove

Correctness of state update at each execution step

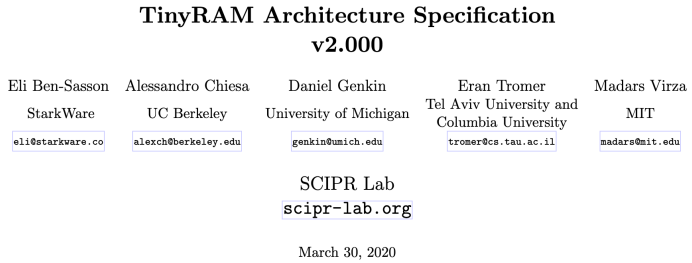
Computation depends on the size of all instructions
[DIO21,YSWW21, WYYXW22, BMRS21]

Consistency of memory reads and writes

(Amortized) constant overhead per access. Large constants!
[DdSGOTV22]

Processor Expressiveness Trade-off

Option 1: Minimal Instruction Set Architecture



TinyRAM Architecture Specification
v2.000

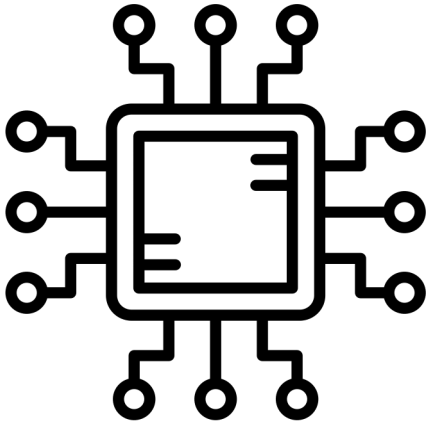
Eli Ben-Sasson, StarkWare, eli@starkware.co
Alessandro Chiesa, UC Berkeley, alexch@berkeley.edu
Daniel Genkin, University of Michigan, genkin@umich.edu
Eran Tromer, Tel Aviv University and Columbia University, tromer@cs.tau.ac.il
Madars Virza, MIT, madars@mit.edu

SCIPR Lab
scipr-lab.org

March 30, 2020

👍 Very few “wasted” gates

👎 Longer trace to emulate missing instructions



Processor Circuit

Option 2: Expressive Instruction Set Architecture

intel

Intel® 64 and IA-32 Architectures
Software Developer’s Manual

Combined Volumes:
1, 2A, 2B, 2C, 2D, 3A, 3B, 3C, 3D, and 4

NOTE: This document contains all four volumes of the Intel 64 and IA-32 Architectures Software Developer’s Manual: *Basic Architecture*, Order Number 253665; *Instruction Set Reference A-Z*, Order Number 325383; *System Programming Guide*, Order Number 325384; *Model-Specific Registers*, Order Number 335592. Refer to all four volumes when evaluating your design needs.

👍 Reason over “real” trace, no instruction emulation

👎 Lots of “wasted” gates

Is there a way out of this expressiveness trade-off?

Dora (Our Result)

ZK for RAM Programs, where the communication and computation complexity of proving each step of an execution is independent of the total number of supported instructions.

$$Complexity = O(\#instructions + \#steps \times size\ of\ one\ instruction)$$

Zero-Knowledge Bag (ZKBag) using linearly homomorphic commitments

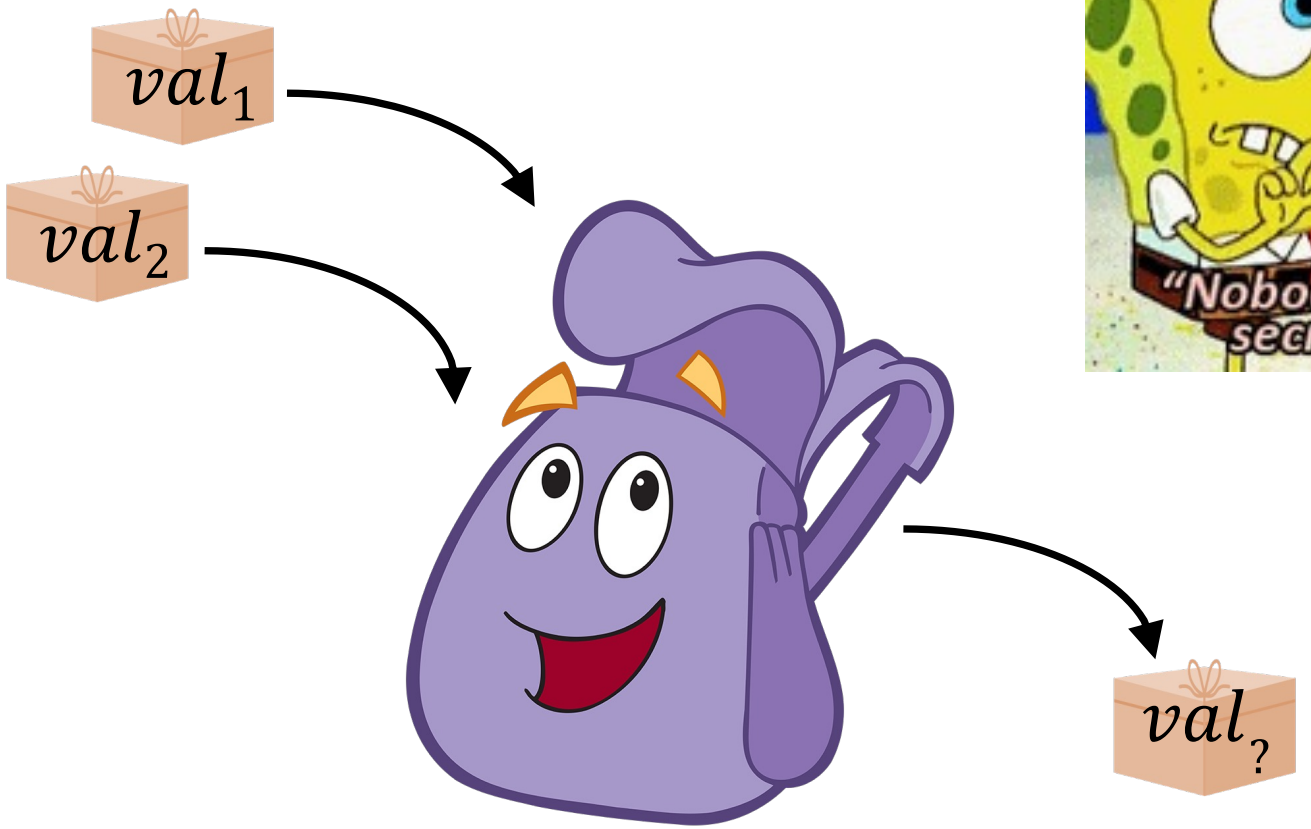
Checking Memory Consistency using ZKBag

Checking Processor Execution using ZKBag and a Folding Scheme

Dora = Memory ZKBag + Processor ZKBag

Concurrent Work: [YHHKV23] and [YH24]

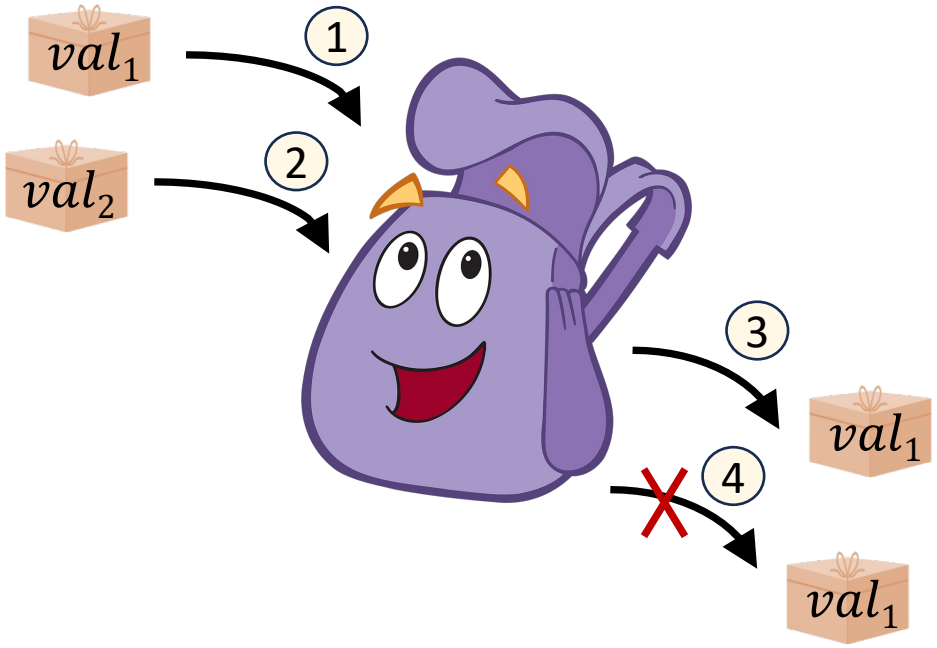
Zero-Knowledge Bag



(Amortized) constant-time insertions and removals

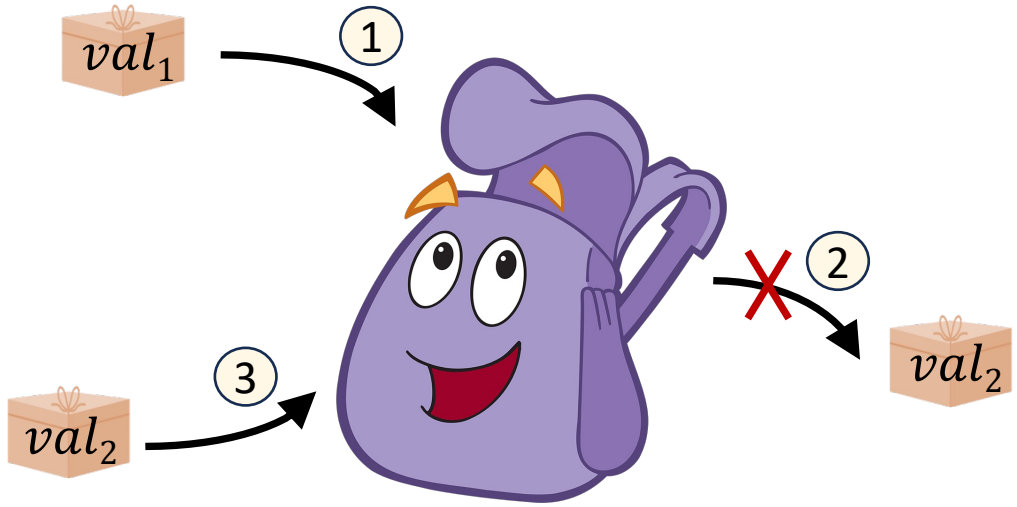
Zero-Knowledge Bag (Properties)

Unique Removal



Elements can only be removed once!

Ordered Binding



Only previously inserted elements can be removed!

Zero-Knowledge Bag (Construction)

Prover

Verifier

Insertion

$com_1 = Com(val_1; op_1)$

com_1 →

$tag_1 \leftarrow \mathbb{F}$

← tag_1

$tcom_1 = Com(tag_1; top_1)$

$tcom_1 = Com(tag_1; top_1)$

Removal

$com_2 = Com(val_1; op_2)$

$tcom_2 = Com(tag_1; top_2)$

$com_2, tcom_2$ →

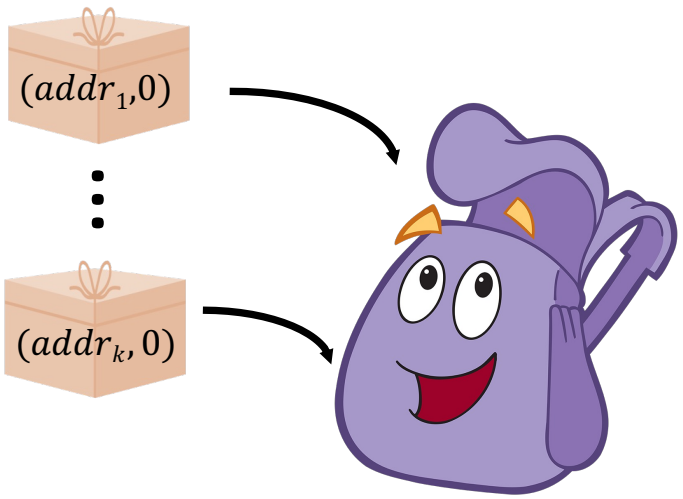
At the End

Check that the bag is empty: →

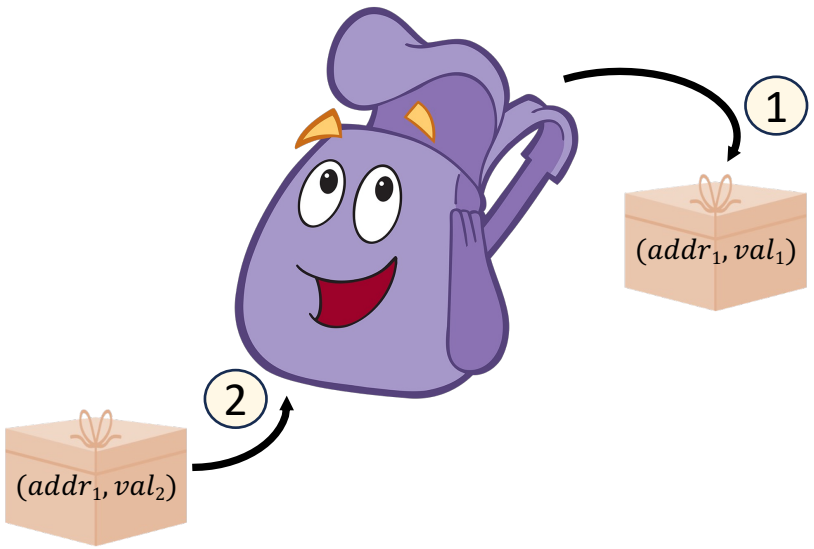
← Demonstrate that insertion commitments are a permutation of removal commitments [Nef01]

Checking Memory Consistency using ZKBag

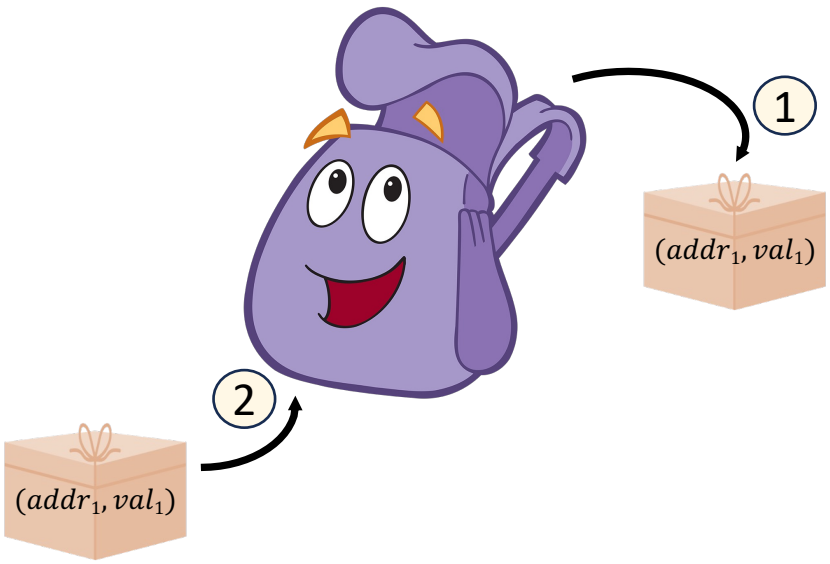
Memory Initialization



Read and Write



Read Only



Processor Execution using ZKBag

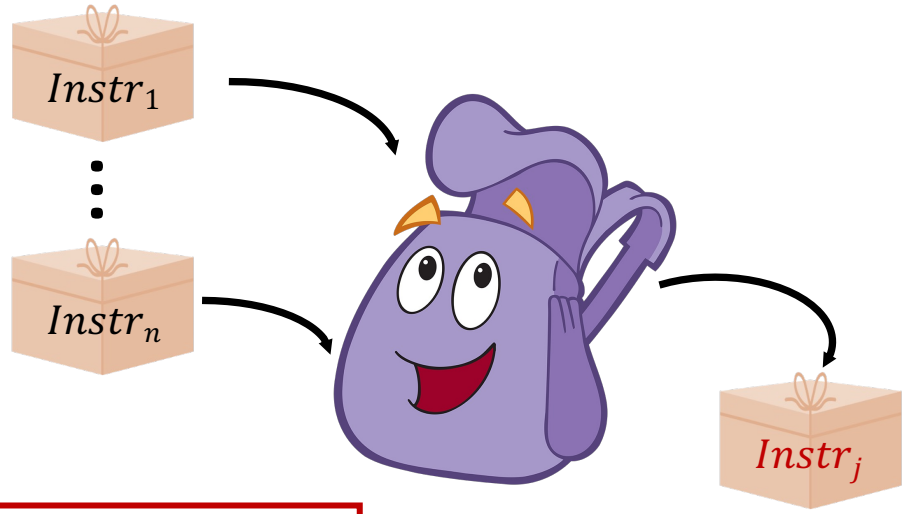
$Set\ of\ Instructions: I = (Instr_1, \ldots, Instr_n)$

We want to verify the following at every execution step:

- $Instr_j \in I$
- $i^{th}\ Execution\ Step:\ state_i \xrightarrow{Instr_j} state_{i+1}$

Strawman Approach: Use ZKBag as a disjunction proof.

Then prove $(state_i \xrightarrow{Instr_j} state_{i+1})$ using any ZK proof.



👎 Need to prove state update over committed instruction

Our Idea: Use Nova [KST22] Folding Scheme!!

Recap: Nova Folding Scheme [KST22]

R1CS $(\boldsymbol{A}.\vec{z}) \circ (\boldsymbol{B}.\vec{z}) = (\boldsymbol{C}.\vec{z}), where\ \vec{z} = (\vec{w} \parallel \vec{x} \parallel 1)$

Extended R1CS $(\boldsymbol{A}.\vec{z}) \circ (\boldsymbol{B}.\vec{z}) = u.(\boldsymbol{C}.\vec{z}) + \vec{e}, where\ \vec{z} = (\vec{w} \parallel \vec{x} \parallel u)$

Folding Relaxed R1CS

$Given\ \boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}, (\overrightarrow{z_1}, \overrightarrow{e_1}), (\overrightarrow{z_2}, \overrightarrow{e_2})\ and\ a\ random\ r$

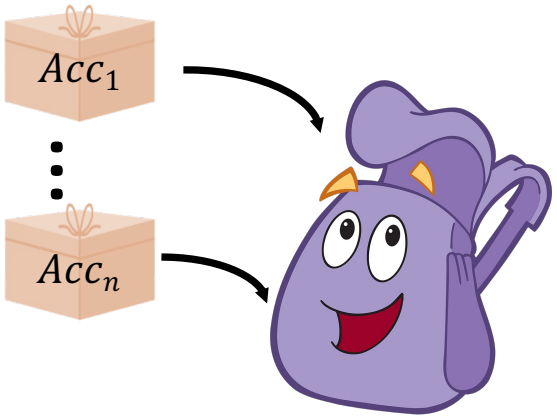
$We\ can\ compute\ \left(\overrightarrow{\vec{z}, \vec{e}}\right), such\ that$

$(\boldsymbol{A}.\vec{z}) \circ (\boldsymbol{B}.\vec{z}) = u.(\boldsymbol{C}.\vec{z}) + \vec{e}\ if\ and\ only\ if$

$(\boldsymbol{A}.\overrightarrow{z_1}) \circ (\boldsymbol{B}.\overrightarrow{z_1}) = u_1.(\boldsymbol{C}.\overrightarrow{z_1}) + \overrightarrow{e_1}\ and\ (\boldsymbol{A}.\overrightarrow{z_2}) \circ (\boldsymbol{B}.\overrightarrow{z_2}) = u_2.(\boldsymbol{C}.\overrightarrow{z_2}) + \overrightarrow{e_2}$

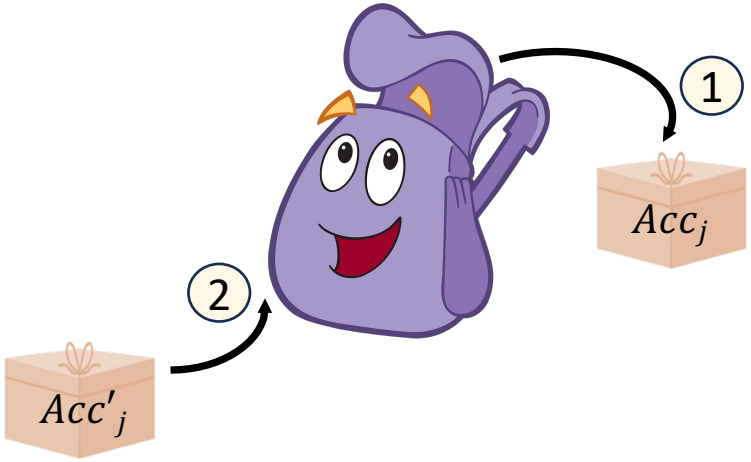
Processor Execution (Our Approach)

Initialization



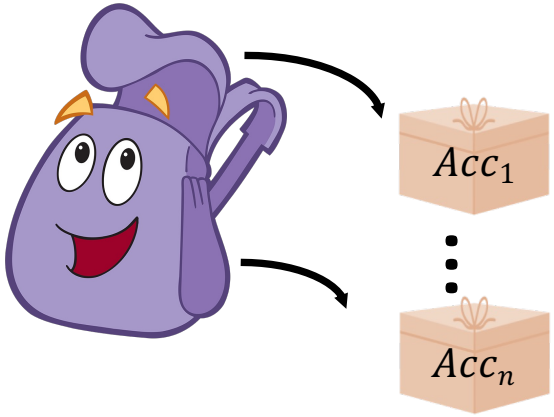
Initialize a relaxed R1CS accumulator for each instruction $(Acc_1, \ldots, Acc_n)$.

i^{th} *Execution Step:* $state_i \xrightarrow{Instr_j} state_{i+1}$



Compute the extended witness to prove $(state_i \xrightarrow{Instr_j} state_{i+1})$ and fold it onto Acc_j to compute updated accumulator Acc'_j

Verification



Retrieve all accumulators, and check if they verify

Putting **Dora** Together

Memory Bag

Processor Bag

Initialize: Initialize Memory Bag and Processor Bag

Processor Tick:

- Read opcode from memory, Read/Write relevant memory address for op
- Fetch current instruction from processor bag and fold

Cleanup/Finish:

- Remove the current state of memory (opening as appropriate)
- Remove all instructions, randomize, and prove
- Verify bags are empty (and executed honestly)

Benchmarks (2020 Laptop, Intel i7@ 2.30GHz)

Mul. Per Clause	2^3	2^6	2^9	2^{12}	2^{15}
2^6	1138.10	1090.46	1166.00	1123.02	663.91
2^9	423.42	418.81	417.60	376.82	240.04
2^{12}	144.03	140.27	147.69	133.81	86.00
2^{15}	58.60	59.23	56.77	54.47	

← No. of Instructions

No. of processor steps Dora proves per second averaged over 50,000 repititions

Network Latency ↓

$t = 2^{23}$	2^{12}	2^{14}	2^{16}	2^{18}	2^{20}
0 ms	545387	545352	545210	547773	543268
10 ms	501591	513535	508092	517432	514070
100 ms	202618	204475	206942	200344	192602

← Memory Space

No. of memory operations Dora can check per second averaged over 2^{23} operations

The code was written in Rust by Mathias and is compliant with the DARPA SIEVE specs

Concurrent Works

Batchman and Robin [YHHKV23]

Batchman and Robin:
Batched and Non-batched Branching for Interactive ZK

Yibin Yang* David Heath† Carmit Hazay‡ Vladimir Kolesnikov§
Muthuramakrishnan Venkitasubramaniam¶

August 20, 2023

Abstract

Vector Oblivious Linear Evaluation (VOLE) supports fast and scalable interactive Zero-Knowledge (ZK) proofs. Despite recent improvements to VOLE-based ZK, compiling proof statements to a control-flow oblivious form (e.g., a circuit) continues to lead to expensive proofs. One useful setting where this inefficiency stands out is when the statement is a disjunction of clauses $\mathcal{L}_1 \vee \cdots \vee \mathcal{L}_B$. Typically, ZK requires paying the price to handle all B branches. Prior works have shown how to avoid this price in communication, but not in computation.

Our main result, Batchman, is asymptotically and concretely efficient VOLE-based ZK for batched disjunctions, i.e. statements containing R repetitions of the same disjunction. This is crucial for, e.g., emulating CPU steps in ZK. Our prover and verifier complexity is only $\mathcal{O}(RB + R|\mathcal{C}| + B|\mathcal{C}|)$, where $|\mathcal{C}|$ is the maximum circuit size of the B branches. Prior works'

For Batched Disjunction

Two Shuffles Make a RAM [YH24]

Two Shuffles Make a RAM:
Improved Constant Overhead Zero Knowledge RAM

Yibin Yang* David Heath†

November 3, 2023

Abstract

We optimize Zero Knowledge (ZK) proofs of statements expressed as RAM programs over arithmetic values. Our arithmetic-circuit-based read/write memory uses only 4 input gates and 6 multiplication gates per memory access. This is an almost 3× total gate improvement over prior state of the art (Delpech de Saint Guilhem et al., SCN'22).

We implemented our memory in the context of ZK proofs based on vector oblivious linear evaluation (VOLE), and we further optimized based on techniques available in the VOLE setting. Our experiments show that (1) our total runtime improves over that of the prior best VOLE-ZK RAM (Franzese et al., CCS'21) by 2-20× and (2) on a typical hardware setup, we can achieve $\approx 600K$ RAM accesses per second.

We also develop improved read-only memory and set ZK data structures. These are used internally in our read/write memory and improve over prior work.

For Checking Consistency of Memory Accesses

Can be combined to obtain a Zero-knowledge proof system for RAM Programs

Similar asymptotic complexities to Dora. Comparison between concrete efficiency is unclear

Future Directions

Working with **Trail of Bits** team to benchmark x86 circuits on real traces

Understanding the concrete efficiency differences resulting from techniques used in Dora and [YHHKV23, YH24]

Can some combination of these different techniques yield better performance?

Thanks!