Scalable Zero-Knowledge Protocols From Vector-OLE

Peter Scholl

24 January 2022, Bar-Ilan Winter School

Based on joint work with:

Carsten Baum, Alex Malozemoff, Marc Rosen,

Lennart Braun, Alex Munch-Hansen, Benoit Razet

AARHUS UNIVERSITY

Zero-knowledge for circuit satisfiability

Circuit $C: \mathbb{F}^n \rightarrow \mathbb{F}$

Prover

Witness $w \in \mathbb{F}^n$

…

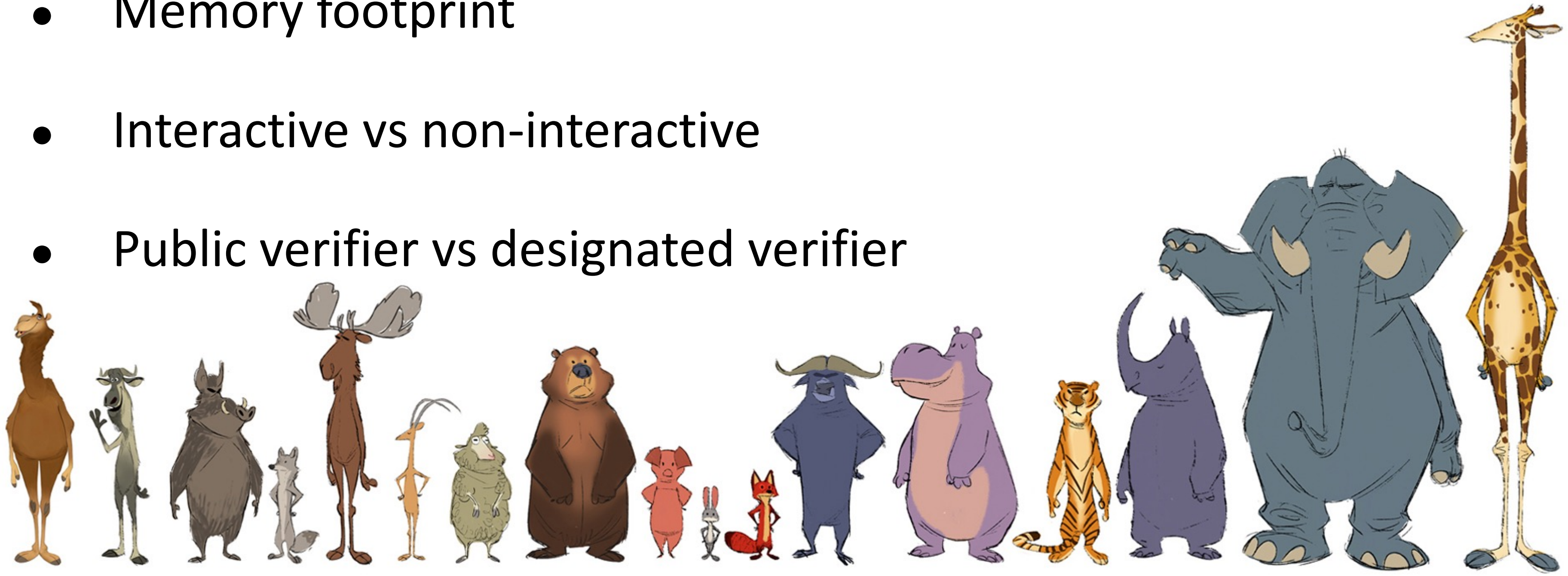
Verifier

Outputs 1 iff $C(w) = 0$

- **Properties:** completeness, soundness, zero-knowledge
  - This talk: proof of knowledge (honest verifier)

The Zero Knowledge Zoo: a few properties

- Runtime:
    - Prover, verifier
- Proof size
- Memory footprint
- Interactive vs non-interactive
- Public verifier vs designated verifier



AARHUS UNIVERSITY

ZK from VOLE: goals and properties

Goal: large-scale statements with low computation/memory overhead

- Prover runtime ≈ cost of evaluating C

Properties:

- Linear-size proofs (worst-case)
- Designated verifier, (possibly) interactive

Motivation: (DARPA SIEVE program)

  - Prove properties of complex programs, e.g. exploit for bug bounty
  - Designated verifier and high interaction are fine in many settings (e.g. MPC)

Overview

Information-theoretic MACs from VOLE

ZK from VOLE: *Mac'n'Cheese* and friends

Non-interactive; streaming

Optimized proofs for disjunctive statements

A2B: arithmetic/binary conversions

VOLE as information-theoretic MACs

- View M_i as MAC on a_i under key (Δ, K_i)
- If Bob tries to open to $a_i' = a_i + e$:
  - Finding valid MAC M' implies $(M' - M_i) \cdot e^{-1} = \Delta$
  - Succeeds with pr. $1/q$

VOLE as information-theoretic MACs

- MAC can be seen as a commitment to a_i:
  Write $[a_i]$
- MACs are linearly homomorphic:
  - Given $[a], [b]$, P and V can locally compute $[a] + [b] \cdot c + d$
- What about small fields, like $\mathbb{F}_2$?
  - Use subfield VOLE: $M = K + a\Delta$ where $a \in \mathbb{F}_2$ and $M, K, \Delta \in \mathbb{F}_{2^k}$

AARHUS UNIVERSITY

Commit & Prove Protocols: instruction set

Commit $(x) \to [x]$:

- Take \$-VOLE element $[r]$
- **P** sends $d = x - r$
- Let $[x] := [r] + d$

Open$([x]) \to x$:

- **P** sends x
- AssertZero$([x] - x)$

AssertZero$([a_1], \dots, [a_m])$:

- **V** sends random $\chi_1, \dots, \chi_m \in \mathbb{F}$
- **P** sends $\chi_1 M_1 + \cdots + \chi_m M_m$
- **V** checks MAC

AARHUS UNIVERSITY

Mac’n’Cheese: *Commit-and-Prove* style ZK

[BMR**S** 21]

MAC the input: Commit(w_1), …, Commit(w_n) $\rightarrow [w_1], \ldots, [w_n]$

- Evaluate circuit gate-by-gate
- Linear gates: easy
- Multiply($[x]$, $[y]$)
  - Commit ($[z]$) (for $z = xy$)
  - Run verification to check that $z = xy$
- Output wire $[z]$: AssertZero($[z]$)

Multiplication in Mac'N'Cheese: simple version

[BMRS 21]

- For each product $[x], [y], [z]$
  - **P** commits to $[c]$ ($= [ay]$) for random $[a]$
  - **V** sends random challenge $e \in \mathbb{F}$
  - $d = \text{Open}(e \cdot [x] - [a])$
  - $\text{AssertZero}(e \cdot [z] - [c] - d \cdot [y])$

Soundness:

- Passing AssertZero implies

$$c - ay = e \cdot (z - xy)$$

- If $z - xy \neq 0$, have guessed e

Cost: P sends 3 field elements (for $[z]$, $[c]$ and d)

Multiplication in Mac'N'Cheese: fancy version

[BMR**S** 21]

- Batch verify $([x_i], [y_i], [z_i])$, for $i = 1, \dots, |C|$
  - Use polynomial based method from fully-linear IOPs [BBCGI 19]
  - **Cost:** $O(\log|C|)$ rounds and communication

Mac’N’Cheese: Simple vs Fancy

- **Communication:** $|w| + 3|C|$ vs. $|w| + |C| + O(\log |C|)$
  (ignoring \$-VOLE)
- **Computation:** $O(|C|)$
- **Rounds:** 1 vs. $O(\log|C|)$

Streaming zero-knowledge proofs

- For complex programs, storing circuit in memory is infeasible
  - E.g. 10s of billions of gates ⇒ hundreds of GB
- Streaming Mac'N'Cheese?
  - Fancy: requires batch verification ☹
  - Simple: batch AssertZero at end ☹
- What if we verify in smaller batches?
  - Worse round complexity ☹

Streaming with Mac'n'Cheese: Fiat-Shamir

- Ideally: want to stream proof while being non-interactive
  - Fiat-Shamir: take care when using on multi-round protocol
  - Worst-case, F-S soundness degrades exponentially with # rounds

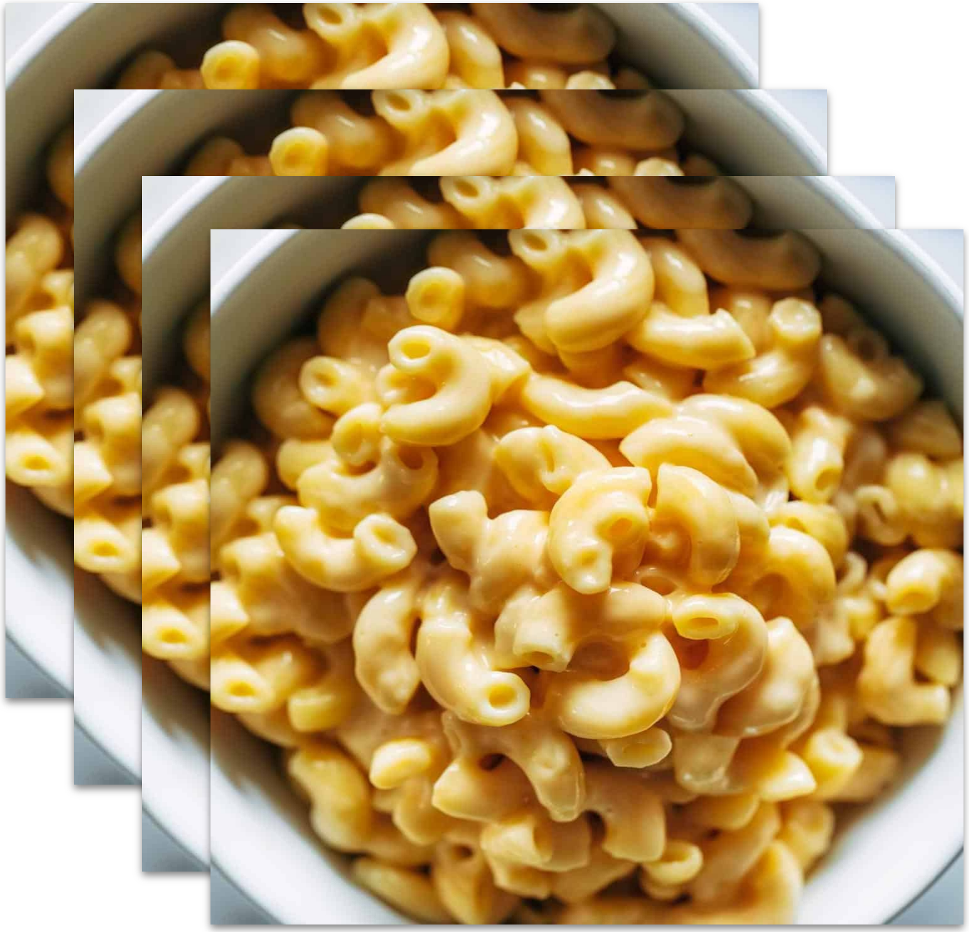
- Mac'n'Cheese satisfies round-by-round soundness [CCHLRR 19]
  - Soundness error $\approx Q/|\mathbb{F}|$ for Q random oracle queries

  (independent of round complexity!)

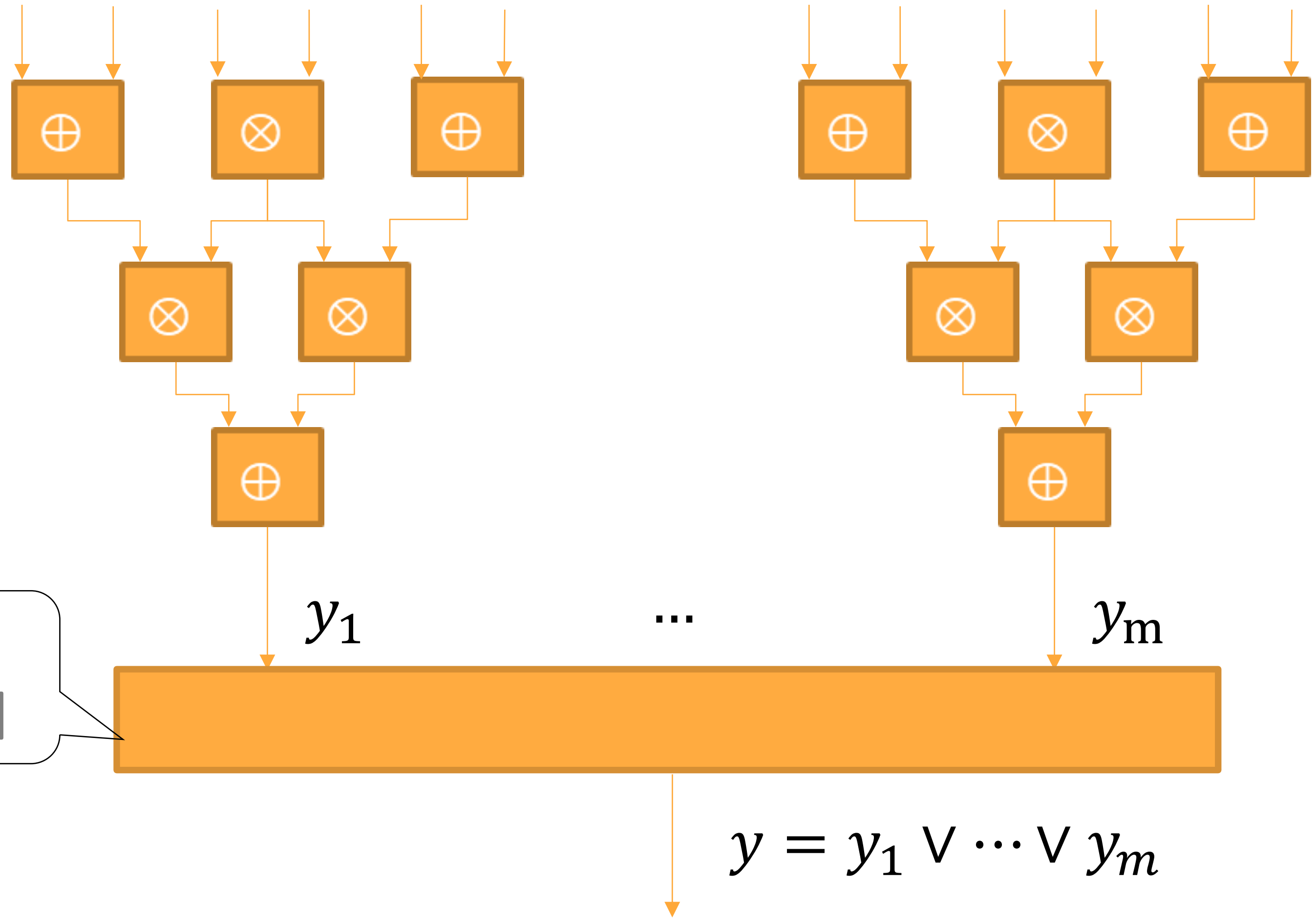
- Gives streamable designated-verifier NIZK (with $-VOLE preprocessing)

AARHUS UNIVERSITY

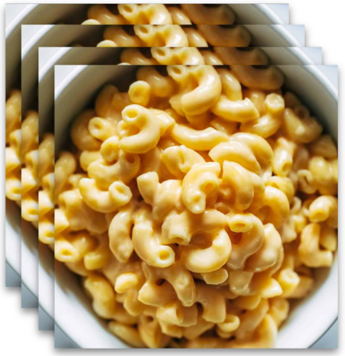
Disjunctions in Commit-and-Prove Systems



Disjunctions



Classic approach:
OR proof [CDS 94]



Optimizing Disjunctions

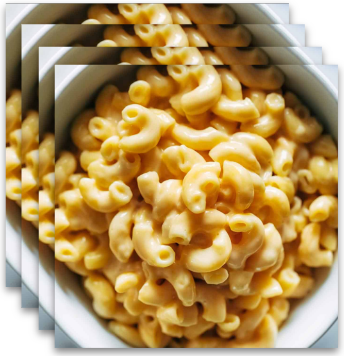
- Want to communicate only information proportional to the longest branch

- **Key observation:**

- Prover's messages in proving $C_i(w)$ are all random elements, or AssertZero
- Given random elements, Verifier doesn't know whether they're for C_1 or C_2.
- *Only send messages of true branch!* $\Longrightarrow$ Verifier uses same messages to evaluate both.

Problem: how to AssertZero in the right branch?
Solution: small "OR proof" to check 1-out-of-m sets of AssertZero

Disjunctive proofs in Mac'n'Cheese



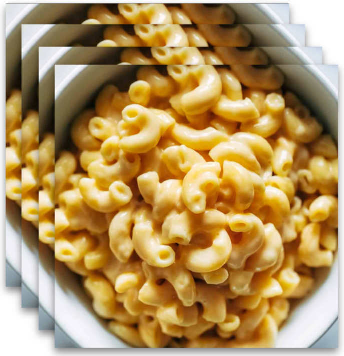
Prove disjunction of clauses $C_1, \ldots, C_m$ where $C_i = 1$

- Prover runs protocol for C_i
- Verifier sends random challenges (as normal)
- End of protocol:
  - **P** needs to prove $[z_i] = 0$, but **V** shouldn't know *i*!
  - Idea: Both parties can define all possible commitments $[z_1], \ldots, [z_m]$
    - All values "garbage" *except* for z_i
  - Run OR proof to show that $\exists i$ such that $z_i = 0$ [CDS94]

Overall communication: $\mathbf{O(\max(C_j)) + O(m)}$

- Naive approach: $O(\Sigma C_j)$
- $\Longrightarrow$ Up to a factor m savings!

Optimizing Disjunctions: Summary



Disjunctions can be optimized for any linear IOP-like protocol

- Recently, also certain sigma protocols [GGHK21]

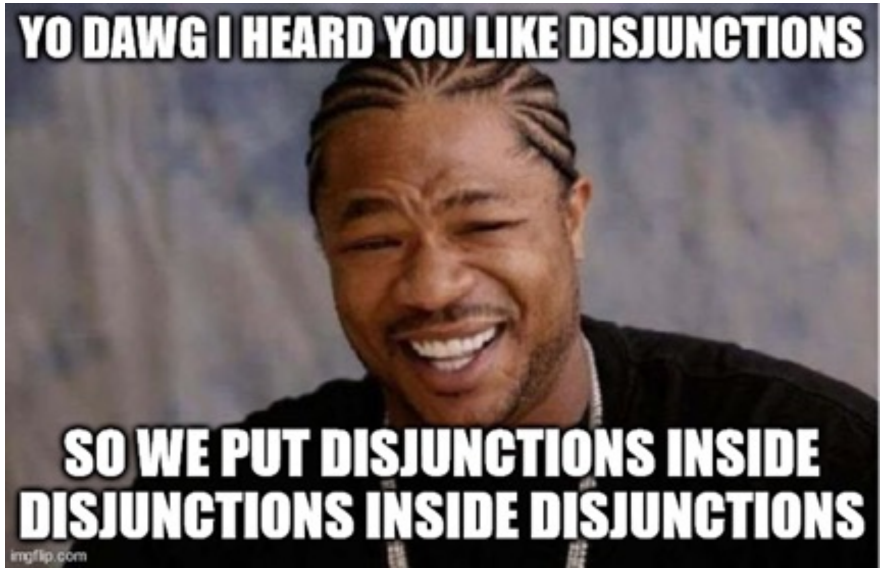
Also support threshold disjunctions for satisfying k-out-of-m clauses $C_1, \ldots, C_m$:

Communication: $\mathrm{k} \cdot \max(|C_j|) + O(m)$

- Naïve: $\sum |C_j|$

Disjunctions inside disjunctions (inside disjunctions…)

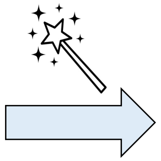
- $O(m)$ becomes $O(\log m)$



ZK from VOLE: other approaches

- Line-point ZK [DIO 21], QuickSilver [YSWW 21]
  - Non-black box use of VOLE
  - Idea: locally multiplying MACs gives a quadratic relation in key Δ

$$[x], [y], [z]$$



$$[c]$$

c is a valid MAC iff $z = xy$

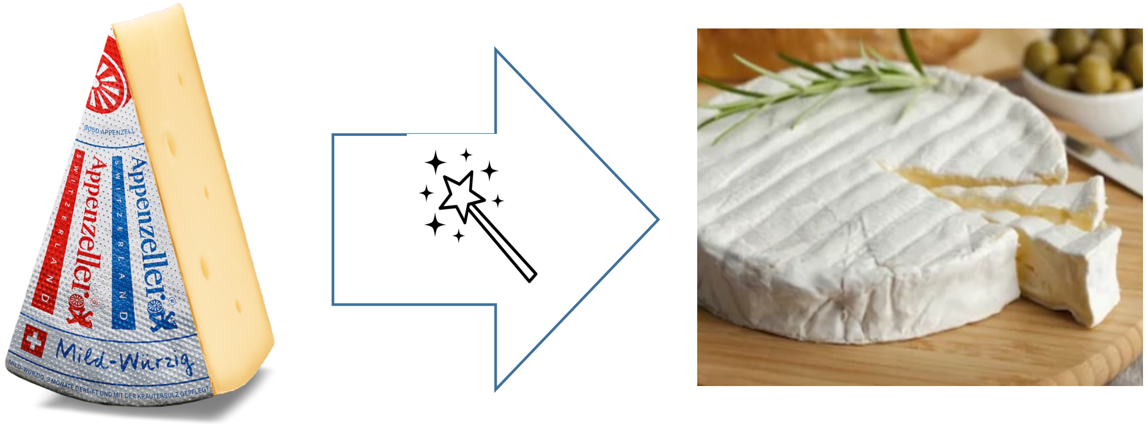
  - Batch MAC check $\Rightarrow$ batch mult. check with $O(1)$ communication!

Comparing Performance of VOLE-based protocols

Protocol	Boolean		Arithmetic		Disjunctions
	Comm.	Mmps	Comm.	Mmps	
Stacked garbling [HK20]	128	0.3	—	—	✓
Mac′n′Cheese (simple) [BMRS21]	9	—	3	—	✓
Mac′n′Cheese (batched)[BMRS21]	$1+\epsilon$	6.9	$1+\epsilon$	0.6^4	✓
QuickSilver [YSWW21]	1	12.2	1	1.4	✗

Mmps: millions of mults per sec

Conversions in ZK protocols



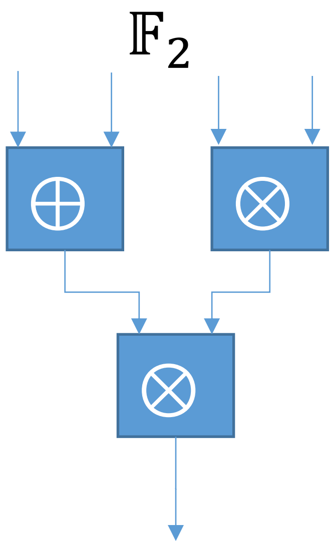
Appenzeller to Brie: Efficient conversions between $\mathbb{F}_2, \mathbb{F}_p$ *and* $\mathbb{Z}_{2^k}$
[Baum, Braun, Munch-Hansen, Razet, **S** ‘21]

Efficient conversion with Appenzeller2Brie

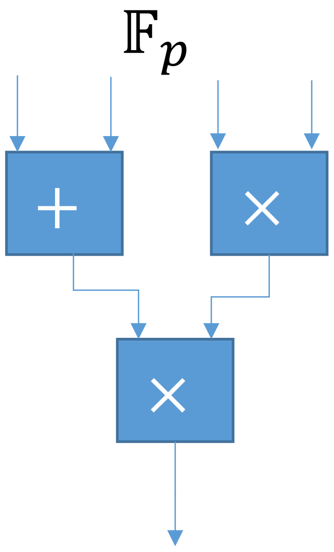
Motivation:
Proof systems only support input in $\mathbb{F}_2$ or $\mathbb{F}_p$
Certain circuits are simpler over other field

Ideally: convert to the most efficient data format for each task during the proof

The problem

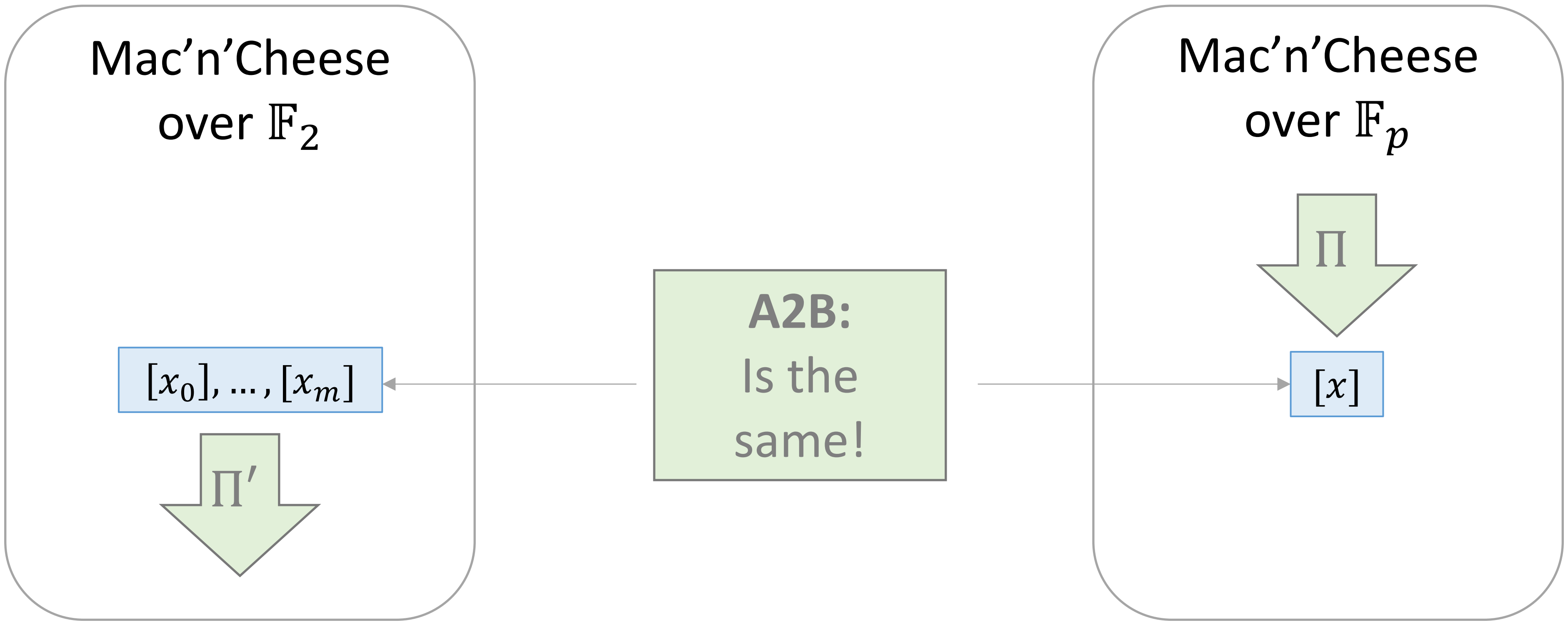


Performance metric:
#AND/multiplications



1. Integer multiplication has a large binary circuit
2. Comparison/truncation expensive to emulate in $\mathbb{F}_p$

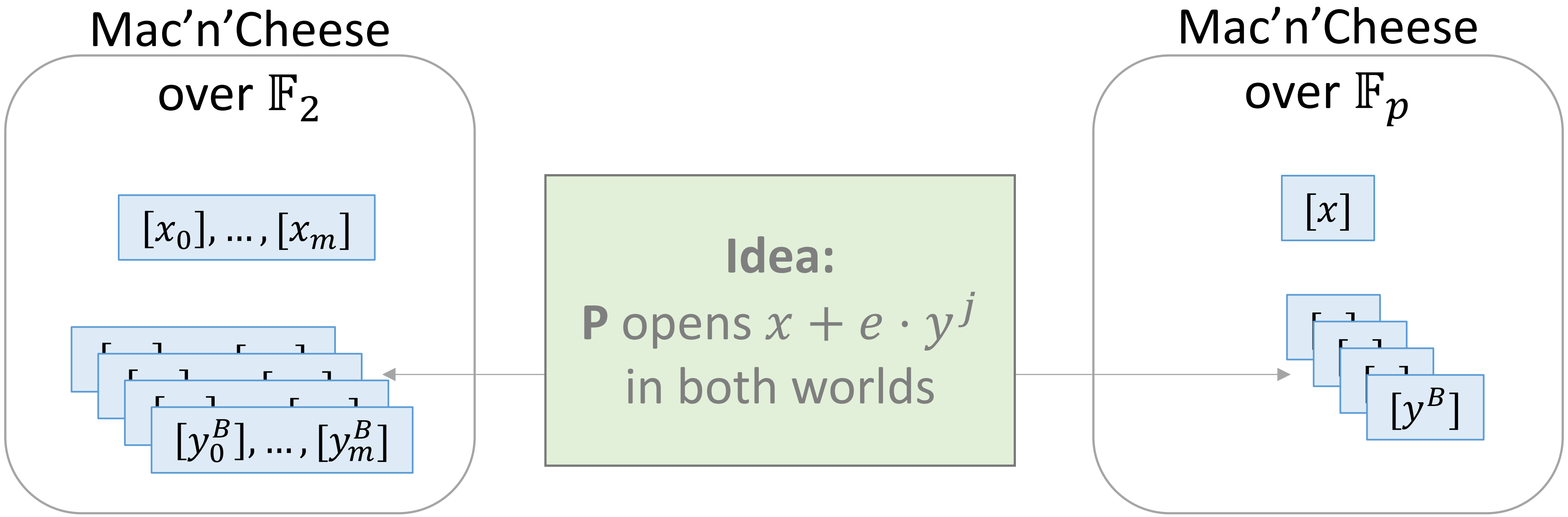
Appenzeller2Brie in a nutshell



We require $p > 2^{m+1}$, approach works for bounded x

Use "EdaBits", similar to [EGK+20,WYX+21]

Appenzeller2Brie in a nutshell



Similar to "EdaBits", used in [EGK+20,WYX+21]

Problems:

1. $e \in \{0,1\}$ only gives soundness ½
2. Larger e is expensive in binary world

A2B: summary

- Instead of randomizing with challenge e, use cut-and-choose
  - Place random conversion tuples into buckets, open small fraction
- Cost: $\approx B$ addition circuits for buckets of size $B \geq 3$
- Optimizations, extensions:
  - Binary circuits for checking conversions allowed to be faulty
  - Use to verify truncations and comparisons

Zero-Knowledge over $\mathbb{Z}_{2^k}$

Mac'n'Cheese does not work over $\mathbb{Z}_{2^k}$ naively.

Solution 1: Emulate operations over $\mathbb{F}_2$ (done in QuickSilver)
Solution 2: Extend Mac'n'Cheese to $\mathbb{Z}_{2^k}$

Problems:

1. MAC and multiplication check fails due to zero divisors
2. VOLE not efficient for $\mathbb{Z}_{2^k}$

A2B: solves (1) using SPDZ2k tricks. (2): still open!

Conclusion

- VOLE ⇒ information-theoretic MACs
  - Powerful for lightweight and scalable zero-knowledge with low memory costs
- "Stacked" OR proof technique
  - Optimizes disjunctions in many settings
- Appenzeller to Brie
  - Conversion gadgets for $\mathbb{F}_2$, $\mathbb{F}_p$ and $\mathbb{Z}_{2^k}$

Open questions

Thank you!

- Sublinear proofs for general circuits
  - Succinct vector commitments from VOLE?
- Beyond designated verifier
  - Some recent progress for multi-verifier setting (2022/082 and 2022/063)
- Improve conversions and $\mathbb{Z}_{2^k}$ support