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
Zero-knowledge for circuit satisfiability

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
ZK from VOLE: goals and properties

**Goal:** large-scale statements with low computation/memory overhead
   - Prover runtime \( \approx \) 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 $(\Delta, K_i)$
- 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 homomorphomic:
  
  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 $M, K, \Delta \in \mathbb{F}_{2^k}$
Commit & Prove Protocols: instruction set

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

**Open**\(([x]) \rightarrow x :\)
- \(P\) sends \(x\)
- \(AssertZero([x] - x)\)

**AssertZero**\(([a_1], \ldots, [a_m]) :\)
- \(V\) sends random \(\chi_1, \ldots, \chi_m \in \mathbb{F}\)
- \(P\) sends \(\chi_1 M_1 + \cdots + \chi_m M_m\)
- \(V\) checks MAC
Mac’n’Cheese: *Commit-and-Prove* style ZK

[BMRS 21]

MAC the input: \( \text{Commit}(w_1), \ldots, \text{Commit}(w_n) \rightarrow [w_1], \ldots, [w_n] \)

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

[BMRS 21]

- Batch verify ($[x_i], [y_i], [z_i]$), for $i = 1, ..., |C|$
  - 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 $\text{-VOLE}$ preprocessing)
Disjunctions in Commit-and-Prove Systems
Disjunctions

Classic approach: OR proof [CDS 94]

\[ y = y_1 \lor \cdots \lor y_m \]
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!* $\implies$ 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: $O(\max(C_j)) + O(m)$
- Naive approach: $O(\Sigma C_j)$
- $\implies$ 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: $k \cdot \max(|C_j|) + O(m)$
- Naïve: $\Sigma |C_j|$

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 $\Delta$

$$[x], [y], [z] \rightarrow [c]$$

$c$ is a valid MAC iff $z = xy$

- Batch MAC check $\Rightarrow$ batch mult. check with $O(1)$ communication!
Comparing Performance of VOLE-based protocols

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Boolean</th>
<th>Arithmetic</th>
<th>Disjunctions</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Comm.</td>
<td>Mmps</td>
<td>Comm.</td>
</tr>
<tr>
<td>Stacked garbling [HK20]</td>
<td>128</td>
<td>0.3</td>
<td>—</td>
</tr>
<tr>
<td>Mac’n’Cheese (simple) [BMRS21]</td>
<td>9</td>
<td>—</td>
<td>3</td>
</tr>
<tr>
<td>Mac’n’Cheese (batched) [BMRS21]</td>
<td>$1 + \epsilon$</td>
<td>6.9</td>
<td>$1 + \epsilon$</td>
</tr>
<tr>
<td>QuickSilver [YSWW21]</td>
<td>1</td>
<td>12.2</td>
<td>1</td>
</tr>
</tbody>
</table>

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

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

Performance metric: #AND/multiplications
Appenzeller2Brie in a nutshell

Mac’n’Cheese over $\mathbb{F}_2$

$[x_0], \ldots, [x_m]$ \downarrow \Pi'

A2B: Is the same!

Mac’n’Cheese over $\mathbb{F}_p$

$\Pi$ \downarrow

$[x]$

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

Use “EdaBits”, similar to [EGK+20,WYX+21]
Appenzeller2Brie in a nutshell

**Mac’n’Cheese over** $\mathbb{F}_2$

$[x_0], \ldots, [x_m]$

$[y_0^B], \ldots, [y_m^B]$

**Mac’n’Cheese over** $\mathbb{F}_p$

$[x]$

$[y^B]$

**Idea:**

P opens $x + e \cdot y^j$ in both worlds

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

**Problems:**

1. $e \in \{0,1\}$ only gives soundness $\frac{1}{2}$
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

- 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