ZKPROOF

Scalable Zero-Knowledge Protocols From Vector-OLE

Peter Scholl

27 April 2021, ZKProof Workshop

Joint work with:

Carsten Baum, Alex Malozemoff, Marc Rosen

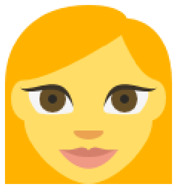
AARHUS UNIVERSITY

galois

AARHUS UNIVERSITY

Zero-knowledge setting

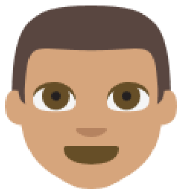
Circuit $C: \boldsymbol{F}^n \to \boldsymbol{F}$



Prover

Witness $w \in F^n$

…



Verifier

Outputs 1 iff $C(w) = 0$

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

Approach: apply state-of-the-art MPC techniques to ZK setting

Caveats:

- (possibly) interactive, designated verifier, not succinct

Overview

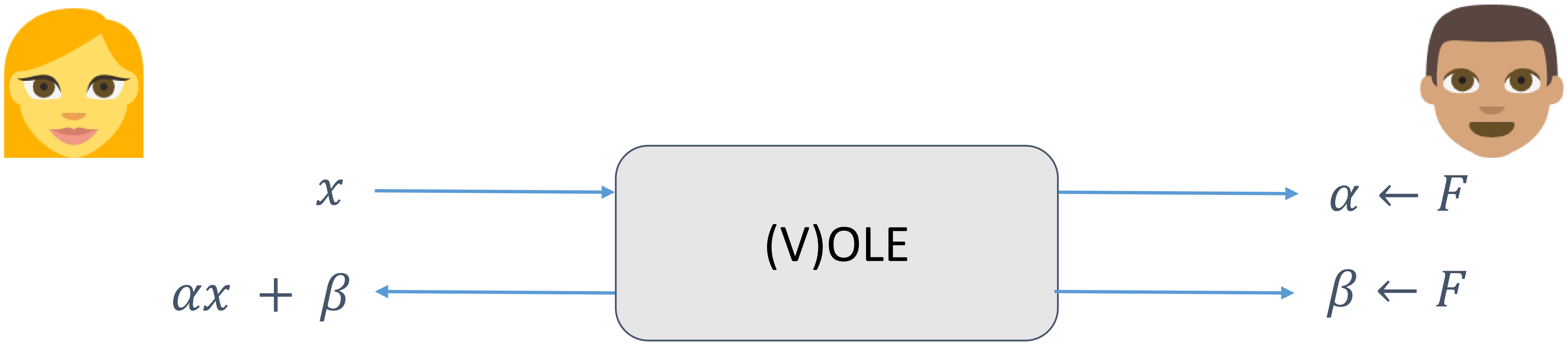
Homomorphic commitments from VOLE

Mac'n'Cheese: fast, "commit-and-prove" ZK

Non-interactive; streaming

Optimized proofs for disjunctive statements

Vector oblivious linear evaluation (VOLE)



VOLE: OLE with α fixed across iterations

- Fast protocols based on LPN [BCGI18, BCGIKRS19, WYKW20, BCGIKS20]
- Small communication (< 1 bit) + computation (~100 ns) per VOLE

Linearly homomorphic MACs from VOLE

Take *random* VOLE over F_p: $m = \alpha r + \beta$

Can view as MAC on r with key (α, β)
- Linearly homomorphic over F_p!
- Soundness *1/p*

Can use as commitment scheme:

- Commit(*x*) → [*x*]
  - Use random VOLE: **P** holds (r, m), **V** holds (α, β)
  - **P** sends $d = x - r$
  - **V** updates $\beta' = \beta + \alpha d$
  - Now: P holds (x, m), V holds (α, β')
- Open([*x*]) → x
  - **P** sends *x* and *m*

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

Assume commitments: $[w_1], \ldots, [w_n]$

- Evaluate circuit gate-by-gate
- Linear gates: easy
- Multiply($[x]$, $[y]$)
  - **P** commits to $[z]$ ($= [xy]$) and $[c]$ ($= [ay]$) for random $[a]$
  - **V** sends random challenge e
  - $\varepsilon = \text{Open}(e \cdot [x] - [a])$
  - $\text{AssertZero}(e \cdot [z] - [c] - \varepsilon \cdot [y])$

AssertZero($[x]$) checks if $\mathrm{x} = 0$

AARHUS UNIVERSITY

Streaming with Mac'n'Cheese

- Multiply$([x],[y])$
  - **P** commits to $[z]$ $(= [xy])$ and $[c]$ $(= [ay])$ for random $[a]$
  - **V** sends random challenge e
  - $\varepsilon = \text{Open}(e \cdot [x] - [a])$
  - AssertZero$(e \cdot [z] - [c] - \varepsilon \cdot [y])$

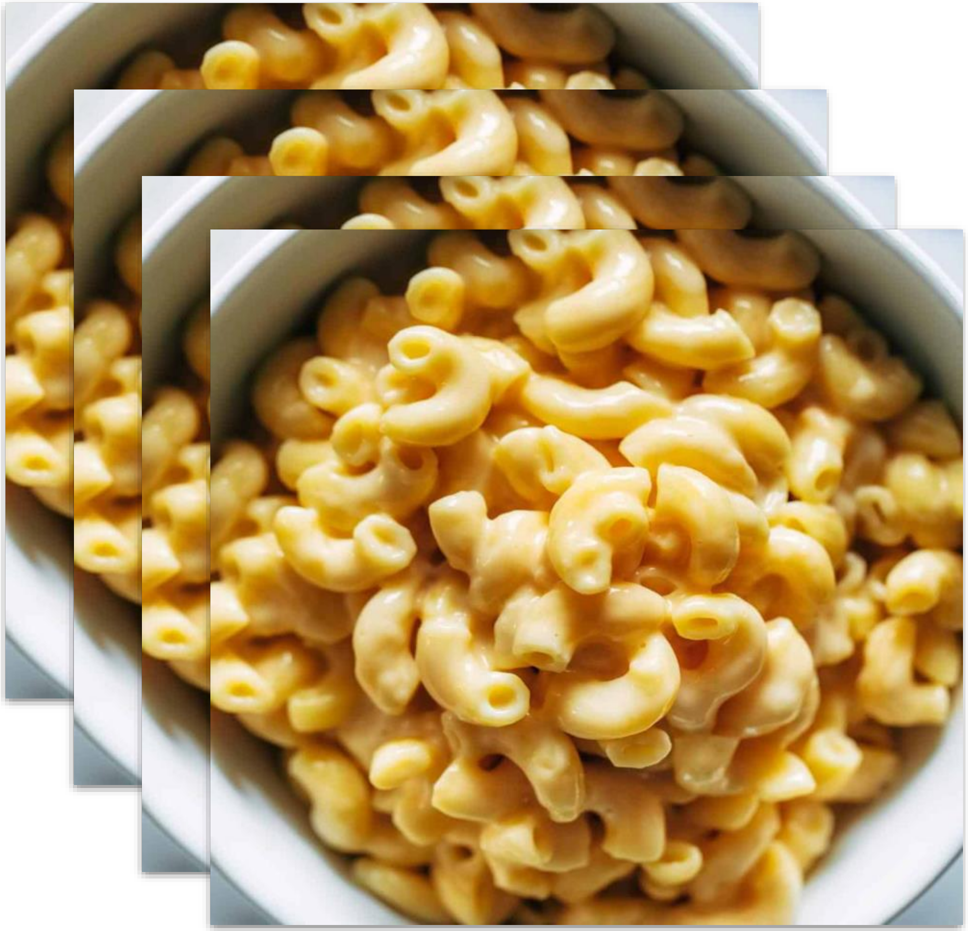
NB: Multiply as above is easily streamable

- Drawback: many rounds of interaction! ☹
- Alternative: batch multiply. No longer constant memory ☹

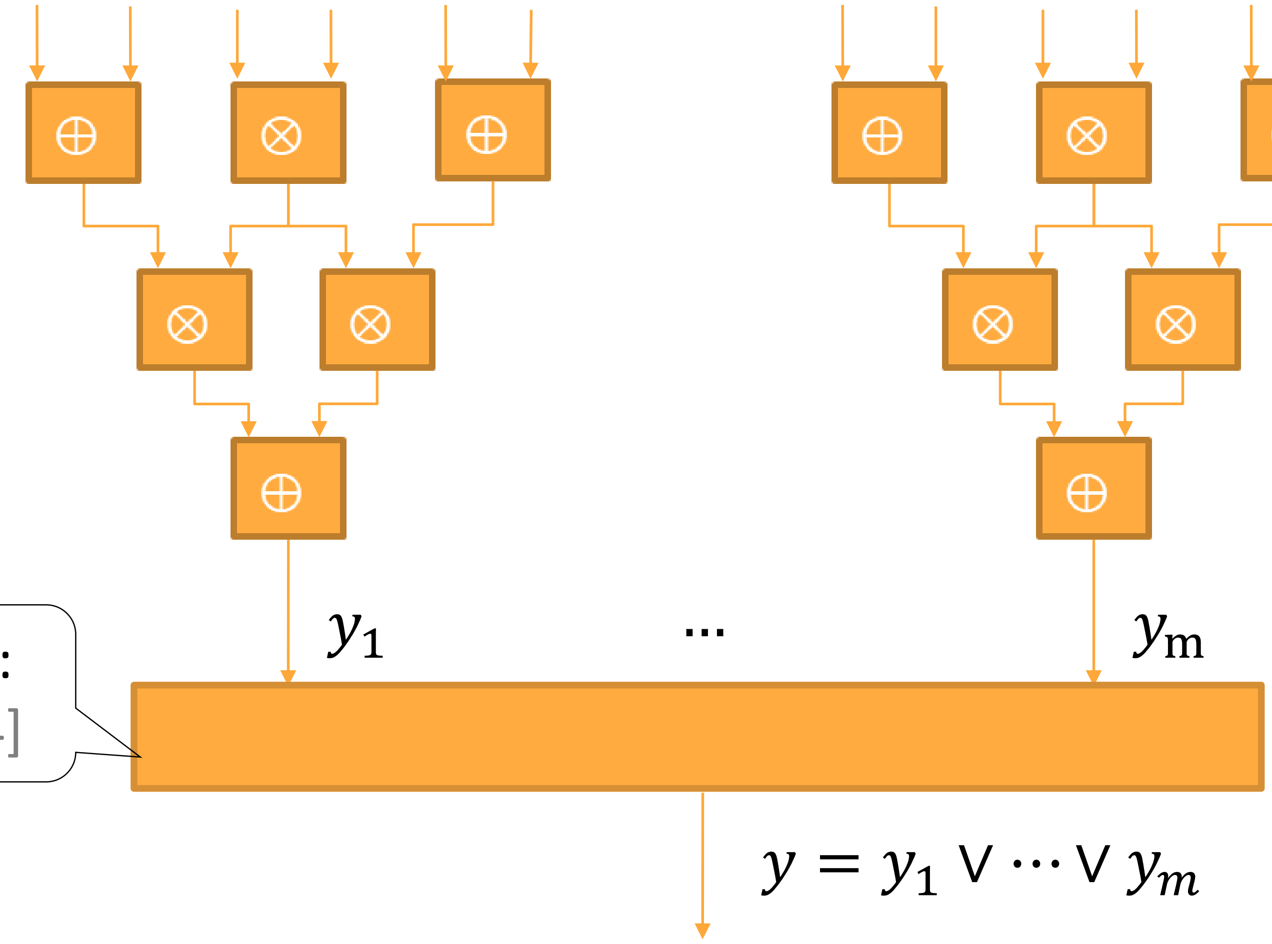
Solution: Fiat-Shamir to squash round complexity

- Security loss depends on #queries, not #rounds!

Disjunctions in Commit-and-Prove Systems

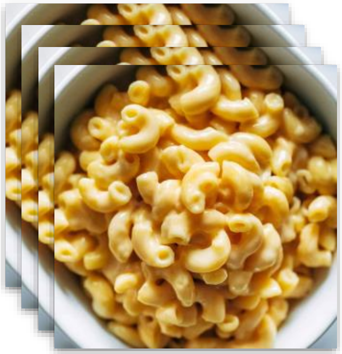


Disjunctions



Classic approach:
OR proof [CDS 94]

Optimizing Disjunctions



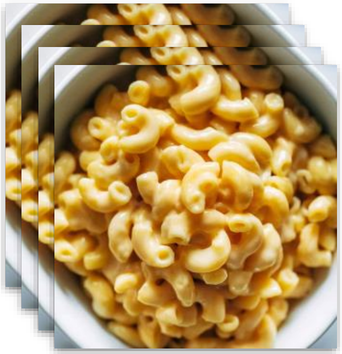
- Want to communicate only information proportional to the longest branch

- **Key observation:**

- Prover's messages in proving $C_i(w)$ are all random elements from Mult (apart from AssertZeros)
- 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 verify the right branch?
Solution: small "OR proof" to check 1-out-of-m sets of AssertZero

Optimizing Disjunctions: Summary



Disjunctions can be optimized for any linear IOP-like protocol

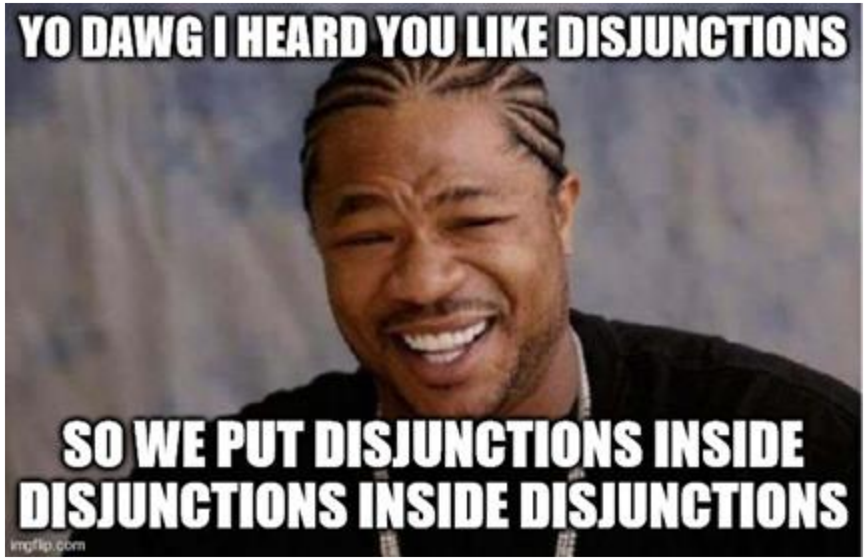
- Recently, also certain sigma protocols [GGHK21]

For m clauses $C_1, \dots, C_m$:

- Total communication $\max(|C_j|) + O(m)$
- vs $\sum |C_j|$ with [CDS94]

With nesting + recursion:

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



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

Conclusion

- VOLE ⇒ lightweight homomorphic commitment scheme
  - Powerful for scalable zero-knowledge with low memory costs
- “Stacked” OR proof technique
  - Optimizes disjunctions in many settings
- Open questions
  - Smaller proofs: succinct vector commitments from VOLE?
  - Beyond designated verifier?

Thank you!