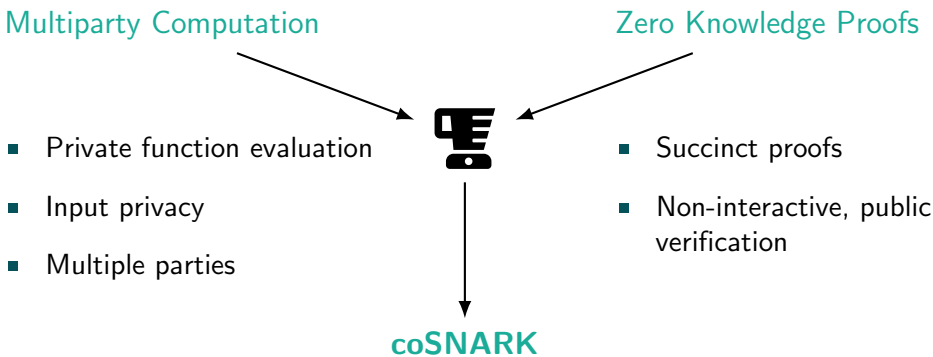
TACEO

coSNARKs - Marrying MPC and ZK

Daniel Kales

May 26th, 2025 - TUM Blockchain & Cybersecurity Salon

coSNARKs - Marrying MPC and ZK

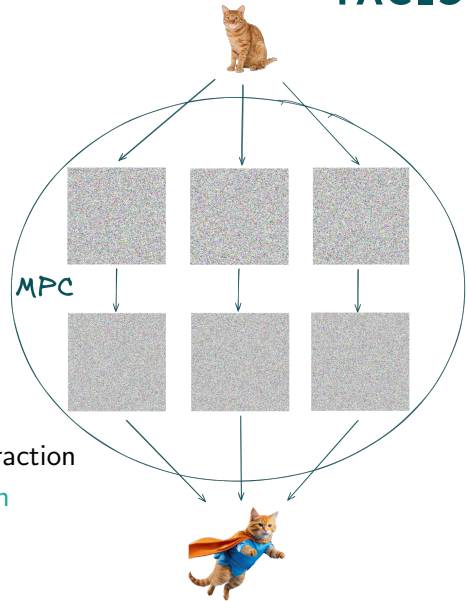


Quick MPC-Primer

- Share data for n parties
  - Each share is essentially random
  - Together, parties can reconstruct data

  $\Rightarrow$ Non-collusion assumption
- Computing functions:
  - Sharing schemes are linear!

  $\Rightarrow$ Share addition, constant addition, constant multiplication can be computed without interaction
  - Share multiplication requires party-interaction
    - Communication often bottleneck in MPC
- Inputs and intermediate results remain private



Collaborative SNARKs

- Basically: Executing a SNARK Prover in MPC
- Why combine two primitives with considerable overhead?
  - Private proof delegation
    - Outsourcing proof generation to more powerful hardware
    - But: Gives away witnesses $\Rightarrow$ use MPC

    $\Rightarrow$ TACEO:Proof
  - Private shared state
    - Blockchains like Aztec store commitments on chain $\rightarrow$ private state
    - But: How to compute with this data?

    $\Rightarrow$ Compute functions and SNARK in MPC
  - Auditable MPC
    - Compute function on secret inputs from multiple parties
    - Prove result to third party

Performance of coSNARKs

- Are ZK proof systems MPC-friendly?

Performance of coSNARKs (cont.)

FFT

$$\vec{y} = \mathrm{FFT}(\vec{x})$$

- Linear operation in $\vec{x}$
- Linearity of secret-sharing scheme
  - Addition, ConstMult is ''free''
  - Perform FFT on shares of $\vec{x}$
- Can reuse GPU/HW-acceleration

MSM

$$C = \sum_{i=0}^{n-1} s_i \cdot G_i$$

- Linear operation in s_i
- Most use-cases: G_i are public
  - Perform MSM on shares of s_i
- Can reuse GPU/HW-acceleration

Performance of coSNARKs

- Are ZK proof systems MPC-friendly?
- Basic SNARKs
  - Mainly linear operations $\rightarrow$ apply directly to shares
  - FFT, MSM, evaluating polynomials at public points, sum-check, ...

  $\Rightarrow$ Very efficient in MPC
  - Groth16 [Gro16], Plonk [GWC19]
  - Marlin, UltraPlonk, UltraHonk
  - Halo2, other KZG/IPA-based ones

Performance of coSNARKs (cont.)

- More advanced features
  - ZK lookup tables $\rightarrow$ protocols require sorting values or counting lookups
  - Often combined with LUT: decomposition of larger values into small chunks
  - Hash-based commitment schemes: Not very MPC-friendly
    - Workarounds exist, but make compatibility with original STARK harder.

Example: Bit decomposition

Standard ZK:

- $19 \rightarrow (1, 0, 0, 1, 1)$
- Even with large $\mathbb{F}$ in *ns* range

In MPC, input is secret shared:

- $[[19]] \rightarrow ([[1]], [[0]], [[0]], [[1]], [[1]])$
- Complex sub-protocol, requires network communication $\rightarrow$ *ms*

Performance of coSNARKs (cont.)

- More advanced features
  - ZK lookup tables $\rightarrow$ protocols require sorting values or counting lookups
  - Often combined with LUT: decomposition of larger values into small chunks
  - Hash-based commitment schemes: Not very MPC-friendly
    - Workarounds exist, but make compatibility with original STARK harder.

Example: Bit decomposition

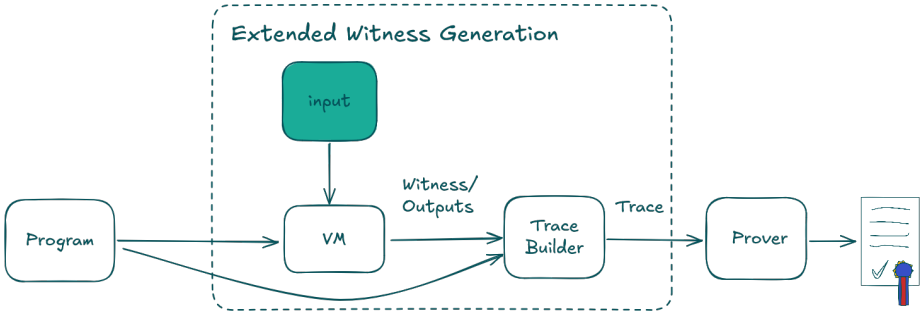
Standard ZK:

- $19 \rightarrow (1, 0, 0, 1, 1)$
- Even with large $\mathbb{F}$ in *ns* range

In MPC, input is secret shared:

- $[[19]] \rightarrow ([[1]], [[0]], [[0]], [[1]], [[1]])$
- Complex sub-protocol, requires network communication $\rightarrow$ *ms*

Challenge: Extended Witness Generation



- Papers focus on proof system building blocks, only talk about extended witness generation in passing
- MPC witness extension for arbitrary programs essentially requires MPC-VM
  - Many ''gadgets'' that might be MPC-unfriendly
    - (Bit) Decomposition, lookups, ROM/RAM, ...

TACEO

Usability of coSNARKs

Frontends: Domain Specific Languages

Goal: Allow devs to use existing, familiar tooling

- Circom → coCircom
  - Old-school circuit format for ZK proofs
  - Groth16 & Plonk backend in `snarkJS`
- Noir → coNoir
  - Modern DSL that is very Rust-like
  - UltraHonk proof system (≈ Hyperplonk + LUT + custom gates)
  - Used and developed mainly by Aztec

`https://github.com/TaceoLabs/co-snarks`

Example: Noir Program in coNoir

main.nr

```
use dep::poseidon;

fn main(input1: [Field; 4], input2: [Field; 4]) -> pub Field {
    let input = [
      input1[0], input1[1], input1[2], input1[3],
      input2[0], input2[1], input2[2], input2[3]
    ];
    poseidon::bn254::hash_8(input)
}
```

Example: Noir Program in coNoir

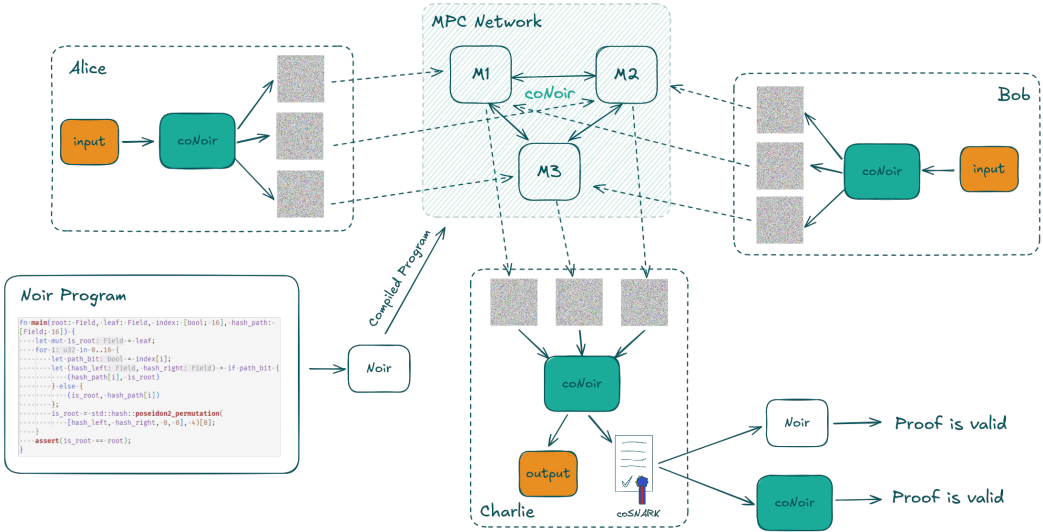
Alice.toml

```
input1 = ["0", "1", "2", "3"]
```

Bob.toml

```
input2 = ["4", "5", "6", "7"]
```

coNoir Pipeline



Implementation Status

coCircom

- Full Circom language support
  - except for uncommon edge cases like unconstrained dynamic loops
  - MPC witness generation phase implementation not optimized much
- Groth16 and Plonk prover
  - MPC-Groth16 prover has almost no overhead compared to `arkworks` baseline
- Circom 2.2.2

coNoir

- Field and Integer datatypes
- Comparisons, Decompositions, Casts
- ROM, RAM model
- Blackbox functions
  - Poseidon, RangeChecks, SHA256, MultiScalarMul, AND, XOR, ...
  - Few still missing: Keccak, AES, ECDSA, RecursiveProofVerify
- Noir 1.0.0-beta.6 & bb 0.86.0

coCircom Benchmarks

- Benchmarks on 3x m7a.4xlarge instances
  - 3.7 GHz, 16 core, 64 GB RAM, 12.5 Gbps

Test	MPC WitEx *ms*	MPC Proof *ms*	snarkJS WitEx *ms*	snarkJS Proof *ms*	rapidSNARK Proof *ms*
Poseidon	16.87	9.63	90.00	540.00	8.57
MT Proof (d=16)	281.21	75.46	120.00	770.00	61.58
MT Proof (d=32)	601.84	132.04	140.00	970.00	121.72
MT (1024)	16.81 s	3.18 s	1.59 s	12.75 s	1.52 s
MT (16384)	254.41 s	48.65 s	24.10 s	155.13 s	20.38 s

coNoir Benchmarks

- Benchmarks on 3x m7a.4xlarge instances
  - 3.7 GHz, 16 core, 64 GB RAM, 12.5 Gbps

Test	MPC WitEx *ms*	MPC Trace *ms*	MPC Proof *ms*	Nargo WitEx *ms*	BB Trace *ms*	BB Proof *ms*
Poseidon2	16.68	45.57	248.16	1.29	14.47	24.73
MT Proof ($d = 16$)	279.09	617.85	2556.45	18.54	25.34	76.44
MT Proof ($d = 32$)	559.17	1227.42	4864.33	37.57	39.59	123.14
Poseidon2 Blackbox function (essentially a custom gate in the proof backend):						
Poseidon2	5.92	4.77	44.49	0.04	14.61	10.29
MT Proof ($d = 16$)	77.52	70.87	356.72	0.49	15.57	27.72
MT Proof ($d = 32$)	143.70	136.50	623.47	1.25	17.40	33.36

Conclusion

coSNARKs - Marrying MPC and ZK

- New Applications
  - Private Proof delegation
  - Private State known to no single entity
- Implementations
  - Build around existing DSLs
  - Actively being worked on at TACEO
- Research
  - Still somewhat small academic niche
  - Lots of room for novel research!

TACEO

Further Links

Workshop at ZK-Summit 12

CoNoir Docs

CoSNARKs Repository

TACEO Blog

Bibliography I

[CLMZ23] Alessandro Chiesa, Ryan Lehmkuhl, Pratyush Mishra, and Yinuo Zhang. **"Eos: Efficient Private Delegation of zkSNARK Provers".** In: USENIX Security Symposium. USENIX Association, 2023, pp. 6453–6469.

[GGJ+23] Sanjam Garg, Aarushi Goel, Abhishek Jain, Guru-Vamsi Policharla, and Sruthi Sekar. **"zkSaaS: Zero-Knowledge SNARKs as a Service".** In: USENIX Security Symposium. USENIX Association, 2023, pp. 4427–4444.

[Gro16] Jens Groth. **"On the Size of Pairing-Based Non-interactive Arguments".** In: EUROCRYPT (2). Vol. 9666. Lecture Notes in Computer Science. Springer, 2016, pp. 305–326.

[GWC19] Ariel Gabizon, Zachary J. Williamson, and Oana Ciobotaru. **"PLONK: Permutations over Lagrange-bases for Oecumenical Noninteractive arguments of Knowledge".** In: IACR Cryptol. ePrint Arch. (2019), p. 953.

[LZW+24] Xuanming Liu, Zhelei Zhou, Yinghao Wang, Bingsheng Zhang, and Xiaohu Yang. **"Scalable Collaborative zk-SNARK: Fully Distributed Proof Generation and Malicious Security".** In: IACR Cryptol. ePrint Arch. (2024), p. 143. URL: https://eprint.iacr.org/2024/143.

Bibliography II

[OB22] Alex Ozdemir and Dan Boneh. **"Experimenting with Collaborative zk-SNARKs: Zero-Knowledge Proofs for Distributed Secrets".** In: USENIX Security Symposium. USENIX Association, 2022, pp. 4291–4308.