Eos: Efficient Private Delegation of zkSNARK provers

Alessandro Chiesa
EPFL, UC Berkeley

Pratyush Mishra
Aleo, UPenn

Ryan Lehmkuhl
MIT

Yinuo Zhang
UC Berkeley
zkSNARKs [Mic94, Groth10, GGPR13, Groth16…
…, GWC19, CHMMVW20, …]

**Goal:** Prove that a private value satisfies some property

---

Prover

Verifier
zkSNARKs

Goal: Prove that a private value satisfies some property

I know $w$ s.t. $\text{SHA256}(w) = x$

public hash: $x$

Prover

Verifier

[Mic94, Groth10, GGPR13, Groth16...]
[... GWC19, CHMMVW20, ...]
zkSNARKs

Goal: Prove that a private value satisfies some property

I know \( w \) s.t. \( \text{SHA256}(w) = x \)

Public hash: \( x \)

Prover -> \( \pi \) zkSNARK proof -> Verifier

[Mic94, Groth10, GGPR13, Groth16…, GWC19, CHMMV20, …]
zkSNARKs [Mic94, Groth10, GGPR13, Groth16…, GWC19, CHMMVW20, …]

Goal: Prove that a private value satisfies some property

I know \( w \) s.t. \( \text{SHA256}(w) = x \)

Zero Knowledge: Verifier learns nothing about \( w \) except that \( \text{SHA256}(w) = x \)
Problem: Proving is *really* slow

I know \( w \) s.t. \( \text{SHA256}(w) = x \)

public hash: \( x \)

Prover \( \pi \) zkSNARK proof Verifier
Problem: Proving is really slow

~50KB

I know \( w \) s.t. SHA256(\( w \)) = \( x \)

Public hash: \( x \)

Prover \( \pi \) zkSNARK proof

Verifier
Problem: Proving is *really* slow

I know \( w \) s.t. SHA256(\( w \)) = \( x \)

~50KB

Public hash: \( x \)

Prover

\( \pi \) zkSNARK proof

Verifier

~1 min runtime
Problem: Proving is *really* slow

- I know \( w \) s.t. SHA256(\( w \)) = \( x \)
- Public hash: \( x \)
- Prover: \( \pi \) zkSNARK proof
- Verifier

~50KB

\( w \) s.t. SHA256(\( w \)) = \( x \)

> 1 min runtime
Potential Solution: Delegate Proving!

Prover

\[ w \rightarrow \text{private witness} \]

Workers

\[ \pi \rightarrow \text{zkSNARK proof} \]

DIZK [WZCPS18]
This leaks the private witness to the workers!
Goal: Delegate Proving with Privacy

Prover

π zkSNARK proof

w private witness

Workers
Goal: Delegate Proving with Privacy

Goal 1: Efficiency  
Delegation should be faster than local proving

Goal 2: Privacy  
The witness should remain hidden from the workers if at least one worker is honest
Our results

Private delegation of ‘algebraic’ zkSNARK provers* in the presence of N-1 malicious workers.

*MBKM19, GWC19, CHMMVW20, CFFQR21, BGH19
Our results

Eos

Private delegation of ‘algebraic’ zkSNARK provers* in the presence of N-1 malicious workers.

*{MBKM19, GWC19, CHMMVW20, CFFQR21, BGH19}

Compared to local proving, running Eos on a mobile phone is:

1) **26x** faster
2) Uses **256x** less memory
Contributions

- Efficient circuits for zkSNARK provers
- Prover-assisted MPC
- Lightweight techniques for malicious security
- Systems optimizations
Contributions

- Efficient circuits for zkSNARK provers
- Prover-assisted MPC
- Lightweight techniques for malicious security
- Systems optimizations
Starting point: MPC

Allows multiple parties to compute a function $F$ over their private inputs

**N-1 Malicious Security:**
Privacy + correctness holds if at least one party is honest
Starting point: MPC

Allows multiple parties to compute a function $F$ over their private inputs

N-1 Malicious Security:
Privacy + correctness holds if at least one party is honest

$F(s_1, s_2, s_3)$
Starting point: MPC

Allows multiple parties to compute a function $F$ over their private inputs

**N-1 Malicious Security:**
Privacy + correctness holds if at least one party is honest

$$F(s_1, s_2, s_3)$$

Generic MPC is expensive!
Efficient Circuits for zkSNARK Provers

MPC Protocol [SPDZ]

1) Express function as an arithmetic circuit
2) All parties execute circuit gate-by-gate

\[ s_1 + s_2 \times s_3 \]
Efficient Circuits for zkSNARK Provers

MPC Protocol [SPDZ]

1) Express function as an arithmetic circuit
2) All parties execute circuit gate-by-gate

Circuit for zkSNARK prover is large! Need to support polynomial arithmetic, group operations, and random oracle calls
Efficient Circuits for zkSNARK Provers

MPC Protocol [SPDZ]

1) Express function as an arithmetic circuit
2) All parties execute circuit gate-by-gate

Idea: Extend the circuit model [SA19, OB22]!

- Add gates for $G$-ops and random oracle calls
- New, efficient subcircuits for polynomial arithmetic
Using the Asymmetric Threat Model

Prover

MPC for zkSNARK

Workers

\[ \pi \text{ zkSNARK proof} \]

\[ \mathbf{w} \text{ private witness} \]
Using the Asymmetric Threat Model

Prover

\[ \pi \] zkSNARK proof

\[ w \] private witness

MPC for zkSNARK

Workers

Prover is always honest and knows the witness!
Idea: Use the weaker prover to ‘assist’ the MPC.
Prover-assisted MPC

MPC Protocol [SPDZ]

1) Express function as an arithmetic circuit
2) All parties execute circuit gate-by-gate
Prover-assisted MPC

MPC Protocol [SPDZ]

1) Express function as an arithmetic circuit
2) All parties execute circuit gate-by-gate

Computing multiplications and random oracles is expensive in MPC but cheap in plaintext
Prover-assisted MPC

MPC Protocol [SPDZ]

1) Express function as an arithmetic circuit
2) All parties execute circuit gate-by-gate

- Workers share subcircuit input wires with prover
Prover-assisted MPC

MPC Protocol [SPDZ]

1) Express function as an arithmetic circuit
2) All parties execute circuit gate-by-gate

- Workers share subcircuit input wires with prover
- Prover executes the subcircuit in plaintext
Prover-assisted MPC

MPC Protocol [SPDZ]

1) Express function as an arithmetic circuit
2) All parties execute circuit gate-by-gate

- Workers share subcircuit input wires with prover
- Prover executes the subcircuit in plaintext
- Prover sends subcircuit output wires to workers

\[ s_1 +_{\mathbb{F}} \rho +_{\mathbb{F}} +_{\mathbb{G}} s_2 \times_{\mathbb{G}} +_{\mathbb{G}} s_3 \]
Achieving Malicious Security

Standard techniques for malicious security incur a large overhead

MPC for zkSNARK

Prover

Workers

\[ w \] private witness

\[ \pi \] zkSNARK proof
Achieving Malicious Security

Standard techniques for malicious security incur a large overhead.

zkSNARKs are “error-detecting”: an incorrectly-generated proof will fail to verify.

MPC for zkSNARK

Workers
Achieving Malicious Security

Standard techniques for malicious security incur a large overhead

Idea: Use the “error-detecting” property of zkSNARKs to reduce the overhead of malicious security

MPC for zkSNARK

Workers

Prover

\( \pi \) zkSNARK proof

\( w \) private witness
Achieving Malicious Security

\[ \pi \text{ zkSNARK proof} \]

Prover

Verify(\(\pi\)) \(\equiv 1\)

MPC for zkSNARK

Workers
Achieving Malicious Security

If a worker deviates from the honest protocol, then the end proof will fail to verify.
Achieving Malicious Security

\[ \text{Prover, Verify}(\pi) = 1 \]

MPC for zkSNARK

Workers

Prover, zkSNARK proof

private witness
Achieving Malicious Security

\[ \text{Prover, Verify}(\pi) = 1 \]
Achieving Malicious Security

Prover
Verify(π) = 1

Workers

MPC for zkSNARK

zkSNARK proof

private witness

\[ \pi \]
Achieving Malicious Security

\[
\text{Prover, } \text{Verify}(\pi) = 1
\]

\[
\text{MPC for zkSNARK}
\]

\[
\text{Workers}
\]
Achieving Malicious Security

If workers see the proof, then selective-failure attacks are possible => leaks one bit of the witness

Prover, Verify(π) = 1

MPC for zkSNARK

Workers

π zkSNARK proof

w private witness
Achieving Malicious Security

If workers see the proof, then selective-failure attacks are possible => leaks one bit of the witness

Idea:
Use additional properties of algebraic zkSNARKs to eliminate these attacks*

Prover

Verify(π) = 1

MPC for zkSNARK

Workers
Implementation

- We implemented Eos as a Rust library in the arkworks ecosystem
- Eos produces a delegation protocol for any “algebraic” zkSNARK

Experimental Setup

- We evaluated our protocols for the Marlin zkSNARK [CHMMVV20]
- 2 workers (AWS c5.24xlarge) in us-west-1 and us-east-1 regions
Eos speeds up proving time by 26x for mobile-phones

<table>
<thead>
<tr>
<th>Prover</th>
<th>Network Throughput</th>
<th>Speedup</th>
<th>Memory reduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>r4.xlarge (AWS)</td>
<td>3 Gbps</td>
<td>9x</td>
<td>256x</td>
</tr>
<tr>
<td>r4.xlarge (AWS)</td>
<td>350 Mbps</td>
<td>6x</td>
<td>256x</td>
</tr>
<tr>
<td>Pixel 4A</td>
<td>350 Mbps</td>
<td>26x</td>
<td>256x</td>
</tr>
</tbody>
</table>

Eos vs. local proving for $2^{20}$ constraints
Eos is 8x faster than existing techniques
Eos is only 10% slower than insecure delegation

Eos vs. worker local proving time
Eos is only 10% slower than insecure delegation.
Eos is only 10% slower than insecure delegation.

Privacy costs $0.03 and 10% increased latency.
Thank You!

Paper/code: www.usenix.org/conference/usenixsecurity23/presentation/chiesa

(Updated version coming soon to ePrint)

Ryan Lehmkuhl
ryanleh@mit.edu