Collaborative zkSNARKs

Zero-knowledge proofs for distributed secrets

Alex Ozdemir, Dan Boneh
Stanford Security Lunch, 9 February 2022
Provable properties about *distributed* secrets

**Money Laundering**

Several banks prove that “Evil” has Colonial Pipeline’s ransom.

**Healthcare Statistics**

Several hospitals prove that procedure prices are “fair”.

**Authentication**

I prove that I know my password/secret key.

**Outline**

1. Why zkSNARKs are insufficient
2. New tool: *collaborative* zkSNARK
3. Building *collaborative* zkSNARKs
4. Surprising efficiency
Background

zkSNARKs
Witness relations

\[ P(x, w) \quad V(x) \]

\[ \exists w. (x, w) \in R \]

- not private
- \( w \) may be large

\[ \{0, 1\} \]
zkSNARKs

\[ \text{pp} \leftarrow \text{Setup()} \quad \text{oO} \quad \{\exists w. (x, w) \in R \} \]

\[ P(x, w) \quad V(x) \]

\[ \pi \leftarrow \text{Prove}(\text{pp}, x, w) \quad \text{Verify}(\text{pp}, x) \]

\{0, 1\}

- (zk) zero knowledge: hides \( w \)
- (S) succinct: short \( \pi \), fast Verify
- (N) non-interactive
- (AR) argument: \textit{computationally} sound
- (K) knowledge: \( P \) knows \( w \)
zkSNARKs: constructions and applications

• From pairings & elliptic curves
  • Groth16 [G’16]
  • Plonk [GWC’19] (using [KZG’10])
  • Marlin [CHMMVW’20] (…”)
  • …
• From hashing & coding theory
  • Fractal [COS’20]
  • …
• From …

I prove that I know my secret key.
• x: public key
• w: secret key
• R: x = PublicKeyGen(w)

For distributed secret data, who plays the prover?
Collaborative zkSNARKs

Definitions
Collaborative zkSNARKs

Syntax:
• Setup() -> pp
• Prove(pp, x, P_1(w_1), .., P_N(w_N)) -> \pi
• Verify(pp, x, \pi) -> \{0,1\}
t-zero-knowledge

Any adversary controlling \( \leq t \) provers learns nothing but whether \((x, w) \in R\)

- Formally
  - Adversary corrupts \( \leq t \) provers
  - ZK simulator is given
    - the corrupt witnesses
    - \( x \)
    - \( b = (x, \overrightarrow{w}) \in R \)
  - We use the random oracle model
Knowledge soundness

- Knowledge soundness
  - *could* mean $P_1$ knows $w_1$, ..., $P_N$ knows $w_N$
  - actually means $P_1$, ..., $P_N$ collectively know $w_1$, ..., $w_N$
    - “distributed knowledge” [Halpern, Moses ‘90]

- Formally
  - we use the random oracle model
  - extractor rewinds *all* provers
    - by re-programming the random oracle
R1CS: The Computational Model

- Class of relations
- Generalize arithmetic circuits

Definition:
- \( R: A, B, C \in \mathbb{F}^{n \times m} \)
- \( x \in \mathbb{F}^{k} \)
- \( w \in \mathbb{F}^{k} \)

Satisfied when:
- \( a = x \|| w \)
- \( Aa \circ Ba = Ca \)
Designing co-zkSNARKs

Overview of constructions
Approach: MPC the Prover

\[ \text{GenericMPC(zkSNARK.Prove, } \vec{w} \text{) } \rightarrow \pi \]

1,000x slower

1,000,000x slowdown ?!

avoid. instead achieve 1000x - 2000x slowdown
Potential Bottlenecks

- Elliptic curve operations
  - Especially multi-scalar multiplications
- Fourier transforms
- Polynomial divisions
- Partial products
- Merkle tree evaluations

- MPC-efficient
- MPC-efficient (for SNARK provers)
- special protocol
MPC Crash-Course

Computation: arithmetic circuit over a finite field

1. Secret-share wire values among N parties

\[ x = x^{(i)} + x^{(i)} + \ldots + x^{(N)} \]

2. Secure protocols for +, * on shares

3. Evaluate circuit, inputs to outputs

We use two MPCs: SPDZ (authenticated additive shares, malicious majority) and GSZ (Shamir shares, honest majority)
MPC-friendly elliptic curve arithmetic

Option 1: Share \((x, y)\) coordinates

\[
\begin{align*}
[g_1] \oplus [g_2] &= (x_1, y_1) \oplus (x_2, y_2) \\
&\rightarrow x = \frac{3x_1^2 + a}{2y_1} \\
&\rightarrow y = \frac{y_1^2 + b}{x_1}
\end{align*}
\]

Elliptic curve addition

\[
g_1 = (x_1, y_1) \in E(F) \\
g_2 = (x_2, y_2)
\]

\[g_1 \oplus g_2\]

Option 2: Elliptic curve sharing

\[g = g^{(1)} \oplus g^{(2)} \oplus \cdots \oplus g^{(N)} \rightarrow \text{add share-wise}\]

Nasty Formulas

- Expensive
  - multiple multiplications
  - communication

Efficient

- no communication
Merkle Trees

\[ H \]

\[ x_1, x_2, x_3, x_4 \]

\[ x_1^{(1)}, x_2^{(1)}, x_3^{(1)}, x_4^{(1)} \]

\[ x_1^{(2)}, x_2^{(2)}, x_3^{(2)}, x_4^{(2)} \]

\[ x_1^{(N)}, x_2^{(N)}, x_3^{(N)}, x_4^{(N)} \]

\[ R^{(1)} \]

\[ R^{(2)} \]

\[ R^{(N)} \]

non-linear and expensive

proof size (& verifier work)

[\( x, N \)]
Implementation
Implementation Goals

• Three base zkSNARKs
  • Groth16
  • Marlin/KZG
  • Plonk/KZG

• Two base MPCs
  • GSZ: Honest majority (t<N/2)
  • SPDZ: Malicious majority (t≤N-1)

• Goals:
  • Iterate on MPCs, sub-protocols
  • Compete with existing zkSNARKs
    • (well optimized!)
  • Don’t work too hard
An Opportunity

1. Arkworks has curve-generic provers:

```rust
fn prove<E: PairingEngine>(..) {
    ...
}
```

2. Curve interfaces define +, *, ...

```rust
trait PairingEngine {
    type ScalarField;
    type Curve;
    fn f_add(...); ...
    fn f_mul(...); ...
    fn c_add(...); ...
    ...
}
```

*Radically oversimplified*
Implementation Strategy

1. Implement MPCs for shared field and curve operations
   1. SDPZ
   2. GSZ
2. Wrap field/curve share types & implement artworks interfaces
3. Instantiate zkSNARK prover
   ➢ Mis-appropriates zkSNARK prover as a co-zkSNARK prover!

<table>
<thead>
<tr>
<th>Component</th>
<th>Lines (Rust)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Network Library</td>
<td>~700</td>
</tr>
<tr>
<td>Plonk</td>
<td>~1200</td>
</tr>
<tr>
<td>MPC protocols</td>
<td>~3000</td>
</tr>
<tr>
<td>Arkworks adapters</td>
<td>~2000</td>
</tr>
</tbody>
</table>
Performance
Experimental Setup

• Measure
  • Wall-clock proving time

• Vary
  • N: number of provers
  • n: R1CS size (# constraints)
  • c: link capacity
  • Base: Groth16/Marlin/Plonk
  • t: security threshold
    • <N/2 (honest majority) (GSZ)
    • <N (malicious majority) (SPDZ)

• Simplifications
  • No intra-prover parallelism
  • Skip MPC preprocessing
**Experiment 1: Good network, few parties**

Fix a 3Gb/s link, vary # rank-1 constraints

---

Honest majority => no slowdown

Malicious majority => 2x slowdown

---

![Graph showing time vs constraints for Groth16, Marlin, and Plonk.

- Groth16
- Marlin
- Plonk

MPC Type:
- 2PC: Dishonest Maj. (SDPZ)
- 3PC: Dishonest Maj. (SDPZ)
- 3PC: Honest Maj. (GSZ)
- Single Prover

---

Honest majority => no slowdown

Malicious majority => 2x slowdown
Experiment 2: Many provers

Fix 1024 constraints, 3Gb/s link, Groth16, vary # of provers

Slowdown grows quadratically with $N$; better for SPDZ
Experiment 3: Low-capacity link

Fix 1024 constraints, 2 provers, malicious majority (SPDZ)

Slowdowns $\leq 8x$ for $\geq 4$ Mb/s; Plonk is slightly worse
Discussion, Future Work

• Bandwidth is the bottleneck for many provers, low link capacity
  • Bad news: a 2-prover co-zkSNARK (additive sharing) requires $\Omega(n)$ communication
    • From randomized 2-party communication complexity of DISJOINT
    • Conjecture: $\Omega(\lambda n)$ (generalize DISJOINT from $\{0,1\}$ to finite field?)

• Exploit intra-prover parallelism

• Find a post-quantum co-zkSNARK with $o(N)$ proof-size
Collaborative zkSNARKs
Alex Ozdemir, Dan Boneh

Conclusions:
1. **Collaborative zkSNARKs** support **distributed secrets**
   - Multiple users, hospitals, banks, ...
2. **Very efficient**
   - $N/2$ corruptions => **no slowdown**
   - $N-1$ corruptions => **$2x$ slowdown**
3. Far better than typical MPC
   - ~$1000x$ slowdown

**Groth16 co-zkSNARK proving time**

![Graph showing Groth16 co-zkSNARK proving time](Image)

- Provers
  - 1 of 2 corrupt
  - 1 of 3 corrupt
  - 2 of 3 corrupt
  - one prover (baseline)

https://github.com/alex-ozdemir/multiprover-snark
https://ia.cr/2021/1530