Zero-Knowledge Proofs for Classical Planning Problems

Augusto B. Corrêa, Clemens Büchner, Remo Christen

University of Basel, Switzerland

AAAI 2023
propositional classical planning tasks:

- propositional variables: \( \{x, y\} \)
- initial state: \( \{x, \neg y\} \)
- goal: \( \{y\} \)
- actions:
  
  - \( a_1 \):
    
    \[
    \begin{align*}
    \text{pre} &= \{\} \\
    \text{eff} &= \{\neg x\}
    \end{align*}
    \]

  - \( a_2 \):
    
    \[
    \begin{align*}
    \text{pre} &= \{\neg x\} \\
    \text{eff} &= \{y\}
    \end{align*}
    \]

- plan: \( \pi = \langle (s_0, a_1, s_1), (s_1, a_2, s_2) \rangle \)
I know a plan for some planning task.
But I won’t show you.

Can I convince you that I really know a plan?
How do I do that without revealing anything about the plan?
I know a plan for some planning task.
But I won’t show you.

Can I convince you that I really know a plan?
How do I do that without revealing anything about the plan?

Zero-Knowledge Proofs!
Formally

**Prover:**
- claims to have a plan \( \pi \) (somehow)
- wants to prove this without revealing anything about \( \pi \)

**Verifier:**
- wants to check that *Prover* is not lying
- do it efficiently
Formally

Prover:
- claims to have a plan $\pi$ (somehow)
- wants to prove this without revealing anything about $\pi$

Verifier:
- wants to check that Prover is not lying
- do it efficiently

bounded plan existence:
Given a planning task $\Pi$ and $k \in \mathbb{N}$ such that $k \leq poly(|\Pi|)$, is there a plan $\pi$ for $\Pi$ with $|\pi| \leq k$?
Formally

Prover:
- claims to have a plan $\pi$ (somehow)
- wants to prove this without revealing anything about $\pi$

Verifier:
- wants to check that Prover is not lying
- do it efficiently

bounded plan existence:
Given a planning task $\Pi$ and $k \in \mathbb{N}$ such that $k \leq poly(|\Pi|)$, is there a plan $\pi$ for $\Pi$ with $|\pi| \leq k$?
- no limitation on $k$ if you reduce to QBF protocols
- Prover needs to stronger than in our case; algebraic
Properties

- **completeness:** Verifier will never reject plan of an honest Prover
- **soundness:** Verifier might be fooled with low probability
- **zero-knowledge:** Verifier learns nothing about the plan
- **efficiency:** protocol executed efficiently by Verifier and Prover
Protocol Overview

Overview of all steps:

- **Prover** obfuscates $\Pi$
- **Verifier** chooses between
  - verifying if obfuscation was done correctly
  - verifying one transition of the claimed plan $\pi$

- if the chosen verification succeeds, **Verifier** accepts $\pi$
- otherwise, rejects

**Prover** communicates via encrypted messages
- opened using some key
propositional variables: \( \{x, y\} \)

initial state: \( \{x, \neg y\} \)

goal: \( \{y\} \)

actions:

\[ a_1: \quad \text{pre} = \emptyset \]
\[ \quad \text{eff} = \{\neg x\} \]

\[ a_2: \quad \text{pre} = \{\neg x\} \]
\[ \quad \text{eff} = \{y\} \]

plan: \( \pi = \langle (s_0, a_1, s_1), (s_1, a_2, s_2) \rangle \)
Prover Obfuscates $\Pi$

transform $\Pi$ into $\hat{\Pi}$ such that it is **hard to map $\hat{\Pi}$ back to $\Pi**

<table>
<thead>
<tr>
<th>variables</th>
<th>initial state</th>
<th>goal</th>
<th>actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>${x, y}$</td>
<td>${x, \neg y}$</td>
<td>${y}$</td>
<td>${} \rightarrow {\neg x}$</td>
</tr>
</tbody>
</table>
Prover Obfuscates $\Pi$

transform $\Pi$ into $\hat{\Pi}$ such that it is hard to map $\hat{\Pi}$ back to $\Pi$

<table>
<thead>
<tr>
<th>variables</th>
<th>initial state</th>
<th>goal</th>
<th>actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>${x, y}$</td>
<td>${x, \neg y}$</td>
<td>${y}$</td>
<td>${} \rightarrow {\neg x}$</td>
</tr>
</tbody>
</table>

make actions indistinguishable:

| $\{x, y, z\}$ | $\{x, \neg y, \neg z\}$ | $\{y\}$ | $\{\neg z\} \rightarrow \{\neg x\}$ | $\{\neg x\} \rightarrow \{y\}$ |
Prover Obfuscates $\Pi$

transform $\Pi$ into $\hat{\Pi}$ such that it is hard to map $\hat{\Pi}$ back to $\Pi$

<table>
<thead>
<tr>
<th>variables</th>
<th>initial state</th>
<th>goal</th>
<th>actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>${x, y}$</td>
<td>${x, \neg y}$</td>
<td>${y}$</td>
<td>${} \rightarrow {\neg x}$</td>
</tr>
</tbody>
</table>

make actions indistinguishable:

<table>
<thead>
<tr>
<th>variables</th>
<th>initial state</th>
<th>goal</th>
<th>actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>${x, y, z}$</td>
<td>${x, \neg y, \neg z}$</td>
<td>${y}$</td>
<td>${\neg z} \rightarrow {\neg x}$</td>
</tr>
</tbody>
</table>

change variable labels and sign:

<table>
<thead>
<tr>
<th>variables</th>
<th>initial state</th>
<th>goal</th>
<th>actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>${\chi, \upsilon, \zeta}$</td>
<td>${\chi, \upsilon, \neg \zeta}$</td>
<td>${\neg \upsilon}$</td>
<td>${\neg \zeta} \rightarrow {\neg \chi}$</td>
</tr>
</tbody>
</table>
Prover Obfuscates $\Pi$

Transform $\Pi$ into $\hat{\Pi}$ such that it is hard to map $\hat{\Pi}$ back to $\Pi$

<table>
<thead>
<tr>
<th>variables</th>
<th>initial state</th>
<th>goal</th>
<th>actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>${x, y}$</td>
<td>${x, \neg y}$</td>
<td>${y}$</td>
<td>${} \rightarrow {\neg x}$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>${\neg x} \rightarrow {y}$</td>
</tr>
</tbody>
</table>

Make actions indistinguishable:

<table>
<thead>
<tr>
<th>variables</th>
<th>initial state</th>
<th>goal</th>
<th>actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>${x, y, z}$</td>
<td>${x, \neg y, \neg z}$</td>
<td>${y}$</td>
<td>${\neg z} \rightarrow {\neg x}$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>${\neg x} \rightarrow {y}$</td>
</tr>
</tbody>
</table>

Change variable labels and sign:

<table>
<thead>
<tr>
<th>variables</th>
<th>initial state</th>
<th>goal</th>
<th>actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>${\chi, \upsilon, \zeta}$</td>
<td>${\chi, \upsilon, \neg \zeta}$</td>
<td>${\neg \upsilon}$</td>
<td>${\neg \zeta} \rightarrow {\neg \chi}$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>${\neg \chi} \rightarrow {\neg \upsilon}$</td>
</tr>
</tbody>
</table>

Add canonical initial and goal states:

(omitted)
Prover Obfuscates $\Pi$

transform $\Pi$ into $\hat{\Pi}$ such that it is hard to map $\hat{\Pi}$ back to $\Pi$

<table>
<thead>
<tr>
<th>variables</th>
<th>initial state</th>
<th>goal</th>
<th>actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>${x, y}$</td>
<td>${x, \neg y}$</td>
<td>${y}$</td>
<td>${} \rightarrow {\neg x}$</td>
</tr>
</tbody>
</table>

make actions indistinguishable:

<table>
<thead>
<tr>
<th>variables</th>
<th>initial state</th>
<th>goal</th>
<th>actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>${x, y, z}$</td>
<td>${x, \neg y, \neg z}$</td>
<td>${y}$</td>
<td>${\neg z} \rightarrow {\neg x}$</td>
</tr>
</tbody>
</table>

change variable labels and sign:

<table>
<thead>
<tr>
<th>variables</th>
<th>initial state</th>
<th>goal</th>
<th>actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>${\chi, \upsilon, \zeta}$</td>
<td>${\chi, \upsilon, \neg \zeta}$</td>
<td>${\neg \upsilon}$</td>
<td>${\neg \zeta} \rightarrow {\neg \chi}$</td>
</tr>
</tbody>
</table>

add canonical initial and goal states:
(omitted)

- change plan $\pi$ into $\hat{\pi}$
- done in polynomial time (details in the paper)
Protocol

Verifier

encrypted \( \hat{\Pi} \) and \( \hat{\pi} \)

Prover
Protocol

Verifier

encrypted $\hat{\Pi}$ and $\hat{\pi}$

$b \in \{0, 1\}$

Prover
Verifier chooses a random value $b \in \{0, 1\}$

- if $b = 0$: verifier checks the transformation $\Pi \rightarrow \hat{\Pi}$
- if $b = 1$: verifier checks one randomly chosen transition of $\hat{\Pi}$
Protocol: $b = 0$

If sent function indeed maps $\Pi$ into $\hat{\Pi}$, Verifier accepts. Otherwise, rejects.
Protocol: \( b = 1 \)

Verifier

encrypted \( \hat{\Pi} \) and \( \hat{\pi} \)

Prover

\( b = 1 \) together with \( i \in [k] \)

key to open \( i \)-th transition of \( \hat{\pi} \)

If transition \( (s_{i-1}, a_i, s_i) \) is valid, Verifier accepts. Otherwise, rejects.
Protocol: $b = 1$

If transition $(s_{i-1}, a_i, s_i)$ is valid, **Verifier accepts**. Otherwise, **rejects**.

in practice, a little bit harder:
- check if $s_0$ and $s_k$ are canonical initial and goal states
- check if $a_i$ is an action of $\hat{\Pi}$
Properties (Revisited)

completeness: Verifier will never reject plan of an honest Prover

soundness: Verifier might be fooled with probability $1 - \frac{1}{2^k}$
- run protocol multiple times to reduce this probability

zero-knowledge: Verifier learns nothing about the plan
- see paper for details

efficiency: protocol executed efficiently by Verifier and Prover
- constant number of messages per execution
- only polynomial-time computation
zero-knowledge proofs for classical planning problems:

- how to prove that you have a plan without revealing it
- works for problems with polynomially long plans
- executed efficiently