Zero-Knowledge Proofs for Classical Planning Problems

Augusto B. Corrêa, Clemens Büchner, Remo Christen

University of Basel, Switzerland
{augusto.blaascorrea,clemens.buechner,remo.christen}@unibas.ch

Abstract

In classical planning, the aim is to find a sequence of deterministic actions leading from the initial to a goal state. In this work, we consider the scenario where a party who knows the solution to a planning task, called the prover, wants to convince a second party, the verifier, that it has the solution without revealing any information about the solution itself. This is relevant in domains where privacy is important, for example when plans contain sensitive information or when the solution should not be revealed upfront. We achieve this by introducing a zero-knowledge protocol for plan existence. By restricting ourselves to tasks with polynomially-bounded plan length, we are able to construct a protocol that can be run efficiently by both prover and verifier. The resulting protocol does not rely on any reduction, has a constant number of rounds, and runs in time polynomial in the size of the task.

Introduction

Alice needs to solve a bunch of "real-world problems" but is not sure how to deal with them. She remembers her neighbor Bob talking about similar problems and asks if he has any advice. Upon hearing Bob's praise of *classical planning*, Alice excitedly models all her problems as classical planning tasks. With the models in hand, only the solutions are missing. Conveniently, Bob is the author of the famous Fast Bobward planning system (cf. Hoffmann and Nebel 2001; Helmert 2006), which is allegedly very fast, and he agrees to solve Alice's problems. Since Bob does classical planning for a living, he expects to be compensated. Alice is happy to pay her neighbor, but wants to be sure that Bob can actually deliver what he promises. Bob assures Alice that he solved the tasks already, but refuses to show her the solutions upfront. The two are left with two questions: How can Alice be sure that Bob really solved her tasks? And how can Bob convince Alice without revealing anything about the solutions?

In this paper, we help Alice and Bob. We show a *zero-knowledge protocol* (Goldwasser, Micali, and Rackoff 1985) for proving plan existence. The protocol is *interactive* and *probabilistic*. It is interactive because a prover (Bob) exchanges messages with a verifier (Alice) to prove that it indeed knows a plan for the task; it is probabilistic because the verifier is capable of generating random bits, and because a successful outcome of the protocol only gives a *high probability* that the prover is not fooling the verifier. Yet, the protocol *guarantees* that the verifier does not learn anything, besides that there exists a plan for the task.

We are interested in a protocol that can be executed *efficiently*. We want the prover to be able to convince the verifier using few messages, while also limiting both to polynomial-time computations. This limits the scope of our protocol: classical planning is **PSPACE**-complete (Bylander 1994) and, while every language in **PSPACE** has a zero-knowledge proof (Ben-Or et al. 1988; Lund et al. 1992; Shamir 1992), the only known protocols for a **PSPACE**-complete language are the ones by Shamir (1992) and by Shen (1992) for true quantified Boolean formulas (QBF). Unfortunately, these protocols do not have the desired efficiency, so reducing a planning task to a QBF would not meet our criteria. It is not known whether languages that are *not* in **NP** have efficient zero-knowledge protocols. Thus, our protocol is limited to tasks with polynomially-bounded plan lengths, an **NP**-complete fragment of classical planning.

Our protocol can be applied to areas where privacy is a concern. This is also the motivation for privacy-preserving multi-agent planning (Brafman 2015; Torreño et al. 2017), but the use-cases for each method are contrasting: privacy-preserving planners deal mostly with problems where actions and states should be kept secret; our protocol deals with problems where the *solution* should be kept secret. Moreover, the two approaches have different computational limitations. Privacy-preserving planners cannot be complete, strong privacy preserving, and efficient (for a particular notion of efficiency) at the same time (Tožička, Štolba, and Komenda 2017), while our protocol can only efficiently handle tasks in **NP** for which the prover already has a plan.

The idea of our protocol is to break up a plan into several transitions. By making transitions indistinguishable from each other, the prover guarantees that each individual transition does not reveal anything about the complete plan. At every execution of the protocol, the verifier checks a single transition to see if it is valid. The verifier then repeats the protocol until it is confident that the complete plan is valid. The presented protocol has a constant number of rounds (i.e., message exchanges), and only requires polynomial-time computation from the prover and the verifier.

Background

Classical Planning We use *propositional STRIPS* (Fikes and Nilsson 1971) extended with negative preconditions. A *planning task* (or simply *task*) $\Pi = \langle \mathcal{V}, \mathcal{A}, I, G \rangle$ has the following components. The set $\mathcal{V}$ is a finite set of propositional *variables*. For any $v \in \mathcal{V}$ we say that v itself and $\neg v$ are its *literals*; v is called a *positive literal* and $\neg v$ is called a *negative literal*. Given a literal l, we write $|l|$ to denote the variable associated with l. A *state* $s \subseteq \mathcal{V}$ is a subset of the variables denoting which variables are true in that state, i.e., $v \mapsto \top$ for all $v \in s$ and $v \mapsto \bot$ otherwise. A *condition* C is a set of literals such that there are no $l, l' \in C$ where $|l| = |l'|$. Let C^+ and C^- be the sets of all positive and negative literals in C, respectively. A state s *satisfies* a condition C (written $s \models C$) if $|l| \in s$ for all $l \in C^+$ and $|l| \notin s$ for all $l \in C^-$. We denote the variables mentioned in condition C as *vars*(C). The set $\mathcal{A}$ is a finite set of *actions* a, each with a *precondition pre*(a) and an *effect eff*(a). Both precondition and effect are conditions and, without loss of generality, we assume that there is no literal l such that $l \in pre(a)$ and $l \in eff(a)$ for any $a \in \mathcal{A}$. Finally, I is a state called the *initial state*, and G is a condition called the *goal*.

Action $a \in \mathcal{A}$ is *applicable* in state s if $s \models pre(a)$. If a is applicable in s, its application results in the *successor state* $s[\![a]\!] = (s \setminus eff(a)^-) \cup eff(a)^+$. If a is not applicable in s, $s[\![a]\!]$ is undefined. The notions of applicability and successor states are extended to finite sequences of actions $\pi = \langle a_1, \dots, a_n \rangle$ in the natural way. We say that an action sequence π has a *transition* (s_{i-1}, a_i, s_i) iff a_i is in π and $s_{i-1}[\![a_i]\!] = s_i$ where s_{i-1} and s_i are states and $1 \leq i \leq n$.

A *goal state* of Π is a state s such that $s \models G$. A *plan* for a state s of Π is an action sequence π that takes s to a goal state: $s[\![\pi]\!] \models G$. The *length* of a plan π corresponds to the number of actions in π and is denoted by $|\pi|$. A plan for I is also called a *plan for* Π. A task Π is *solvable* if there exists a plan for Π.

In this paper, we are interested in the language of all planning tasks that have plans with length bounded by some natural number k. We call this language BOUNDEDPLANEX.

Definition 1 (BOUNDEDPLANEX). *Given a planning task Π and $k \in \mathbb{N}$, the pair $\langle \Pi, k \rangle$ is in* BOUNDEDPLANEX *iff there exists a plan π for Π with $|\pi| \leq k$.*

Deciding if $\langle \Pi, k \rangle \in$ BOUNDEDPLANEX is **PSPACE**-complete in general (Bylander 1994). In this work, however, we are primarily interested in planning tasks that have plans of *polynomial length in the representation size of the task*. The *representation size* $\|\Pi\|$ of a planning task is the length of a reasonable encoding of Π. This fragment of classical planning is **NP**-complete: nondeterministically guess a polynomially-long action sequence and check sequentially whether it is a plan for Π.

Let F be a family of tasks and q be a positive polynomial such that for every task Π in F, there exists a plan π for Π such that $|\pi| \leq q(\|\Pi\|)$. Then there exists a constant $c \in \mathbb{N}$ such that $|\pi| \leq q(\|\Pi\|) \leq O(\|\Pi\|^c)$ for every task Π in F. Thus, we say that we are interested in families of tasks with plans bounded by $O(\|\Pi\|^c)$ for some constant c.[1]

[1]Note that c and q depend on the family of tasks F.

Computational Complexity We assume familiarity with the classes **P**, **NP**, and **PSPACE**, polynomial reductions, and the concepts of hardness and completeness. We explain the concepts of *interactive proofs* and the class **IP** (Babai 1985; Goldwasser, Micali, and Rackoff 1985) next. Due to space limitations, we refer to the seminal textbook by Arora and Barak (2009) for a more comprehensive introduction.

Intuitively, an interactive proof for a language L and some common input x is a *protocol* between a *prover* P, who has unlimited computational power, and a probabilistic polynomial-time *verifier* V. Prover P claims to have a *proof* (sometimes called a witness or certificate) that $x \in L$ and wants to convince V of that. P and V then perform a sequence of message exchanges, called *rounds*, so P can try to convince V. The verifier, however, does not trust P: it might be that P is a *dishonest* prover that does not have a proof for $x \in L$ and is only trying to cheat the verifier.

Let $\langle V, P \rangle(x)$ be the output of the interaction between V and P on a common input x. We write $\langle V, P \rangle(x) = 1$ if the verifier V *accepts* the protocol, i.e., if V was convinced by P that $x \in L$.

A language L is in **IP** if there exists a protocol with a polynomial number of rounds satisfying two conditions with respect to the common input x:

$$x \in L \implies \exists P \Pr[\langle V, P \rangle(x) = 1] \geq \frac{2}{3} \qquad (1)$$

$$x \notin L \implies \forall P \Pr[\langle V, P \rangle(x) = 1] \leq \frac{1}{3} \qquad (2)$$

where $\Pr[\langle V, P \rangle(x) = 1]$ denotes the probability that V accepts x after interacting with P. In words, if $x \in L$ then P can convince V to accept x with high probability, and if $x \notin L$ then P can only make V falsely accept x with low probability.[2]

Condition (1) is called *completeness*, and (2) is called *soundness*. Given a language L, if there exists a probabilistic polynomial-time verifier V and a protocol satisfying (1) and (2) within a polynomial number of rounds, then $L \in$ **IP**.

In more practical settings, there exists an additional efficiency requirement: P is allowed to use a lot of resources to find a proof for $x \in L$ but has limited resources to prove it to V. This models the idea that we do not want the verification of the solution to be as expensive as solving the problem from scratch. Although **IP** = **PSPACE** (Lund et al. 1992; Shamir 1992), it is unclear whether **PSPACE**-complete languages have interactive proofs with this efficiency property. While languages in **NP** have very efficient protocols, to the best of our knowledge, the only known protocols for languages *not* in **NP** (Lund et al. 1992; Shamir 1992; Shen 1992) rely on very powerful provers.

Zero-Knowledge Proofs An interactive proof is called a *zero-knowledge proof* if it does not provide a single additional bit of information to the verifier, besides whether $x \in L$ or not (Goldwasser, Micali, and Rackoff 1985). Goldreich, Micali, and Wigderson (1986) proved that any decision problem in **NP** has a zero-knowledge proof, under the

[2]Constants $2/3$ and $1/3$ can be strengthened or weakened without changing the power of **IP**. See Arora and Barak (2009) for details.

assumption that one-way functions exist. Later, Ben-Or et al. (1988) showed that, under the same assumption, every language in **IP** has a zero-knowledge proof.

Formally, zero-knowledge is defined under the *simulation paradigm* (Goldreich 2001): a protocol is zero-knowledge if the verifier can *simulate* the interaction with the prover on its own, *without the prover*. More formally, there should exist a polynomial-time probabilistic *simulator* M^* whose output distribution is *computationally indistinguishable* from the output distribution of the interaction between verifier V and prover P, denoted $\langle V, P\rangle(x)$. In other words, there is no efficient algorithm that can distinguish between $\langle V, P\rangle(x)$ and the output of M^* on input x, except with negligible probability.[3] This should also hold for any arbitrary *strategy* that verifier V uses, even if it does not follow the protocol.

Definition 2 (Zero-Knowledge Proofs)**.** *Given a protocol between a verifier V and a prover P with common input x, the protocol is said to be* zero-knowledge *for some language L iff for every probabilistic polynomial-time verifier* strategy V^**, there exists a probabilistic polynomial-time* simulator M^* *such that when* $x \in L$ *the output distribution of* $\langle V^*, P\rangle(x)$ *and the output distribution of* M^* *on* x *are* computationally indistinguishable.

Definition 2 asserts that if V could have learned anything from the protocol, then it could have also learned it by running the simulator M^* on x. In simpler terms, if there is an algorithm M^* that can imitate the interaction $\langle V, P\rangle(x)$, the verifier V could simply execute this algorithm instead of interacting directly with P. The definition also quantifies over all possible verifier strategies to guarantee that the protocol is zero-knowledge even when the verifier deviates from the established protocol (i.e., it tries to cheat the prover). The common way of proving that a protocol is indeed zero-knowledge is by showing that such a simulator M^* exists (e.g., Goldwasser, Micali, and Rackoff 1985; Blum 1986; Impagliazzo and Yung 1987; Goldreich 2001).

It is important to stress that the simulator must only be correct when $x \in L$, while there is no requirements when $x \notin L$. This models the idea that the verifier learns nothing about the proof that $x \in L$, and in the case where $x \notin L$ there is nothing to be learned.

Commitment Schemes An important ingredient in zero-knowledge protocols is the concept of *commitment schemes* (Goldreich 2001). Using a commitment scheme, the prover P can *commit* to a message while keeping it hidden from the verifier V. This commitment can be later *opened* whenever P wants to *reveal* the information of the message to V. Computationally, a commitment scheme is a function that produces a commitment string *Commit*(x) given an object x and some key $K_x \in \{0, 1\}^n$. The commitment *Commit*(x) can be opened using the same key K_x.

Commitment schemes have two important properties: *unambiguity* and *secrecy*. A scheme is unambiguous if using two different keys to commit the same objects always produces two different commitments; it is secret if given two objects x and y, their commitments *Commit*(x) and *Commit*(y) are computationally indistinguishable.

We write *Commit*(x) to represent the commitment of an object x. We say that the prover P *opens* or *reveals* x to V to indicate that P gives V the key K_x to open x. With some abuse of notation, we write *Commit*(L) to indicate the element-wise commitment of some sequence or set L.

Commitment schemes can be seen as functions to *encrypt* objects such that their original value is only revealed under demand. In our work, we assume that the function *Commit*$(\cdot)$ is unambiguous and secret. Our results hold for any function following these criteria. For more details on commitment schemes and concrete examples, we refer to the canonical book by Goldreich (2001).

Plan Existence Protocol

We now introduce our protocol, ZK-BOUNDEDPLANEX. With ZK-BOUNDEDPLANEX, a prover P who claims to know a plan π for Π, with $|\pi| \leq k \leq O(\|\Pi\|^c)$ for some constant $c \in \mathbb{N}$, can convince a verifier V that $\langle \Pi, k\rangle \in$ BOUNDEDPLANEX in a polynomial number of rounds, while not disclosing any knowledge about π. Our protocol is inspired by the zero-knowledge protocol for Hamiltonian cycles by Blum (1986), and by the zero-knowledge simulation of computation by Impagliazzo and Yung (1987).

Let us give the high-level idea first. To start, the prover P transforms the planning task Π into a new task $\hat{\Pi}$ where variables are permuted and actions are made indistinguishable. The trick here is that $\hat{\Pi}$ still preserves all plans of Π but actions and variables of $\hat{\Pi}$ cannot be easily identified. Next, P sends $\hat{\Pi}$ together with a commitment of $\hat{\pi}$ – an obfuscated version of the original plan π – to V. The prover then poses two possible options to the verifier: V can either choose to check the mapping from Π to $\hat{\Pi}$ and confirm if the transformation is valid, or V can check a part of the plan $\hat{\pi}$ on its own.

The verifier V randomly chooses one of the two options. If V decides to check the transformation, P provides the transformation function to V, and V simply compares Π to $\hat{\Pi}$, accepting if both are equivalent (otherwise, it rejects); if V decides to check the plan itself, it selects one random transition of $\hat{\pi}$, P reveals this transition to V, and V computes locally whether this transition is valid or not. If it is valid, it accepts; otherwise it rejects.

If $\hat{\pi}$ is a valid plan for $\hat{\Pi}$, then all transitions in the plan are valid. Otherwise, at least one transition is invalid (either not applicable or leads to a different successor state than the one commited by P) and V has $1/k = 1/q(\|\hat{\Pi}\|)$ chance of spotting it, where $q(\|\hat{\Pi}\|) \leq O(\|\hat{\Pi}\|^c)$ is a positive polynomial as previously defined.

There are some other details that V also needs to take care of. For example, it needs to verify that $\hat{\pi}$ starts at the initial state and ends at a goal state. We detail all these steps next.

P claims to have a plan π for Π but V cannot be sure that this π is *indeed* a plan – it can be some random sequence of actions, if P is a dishonest prover. We thus say that π

[3]A function $f(x)$ is called *negligible* iff $|f(x)| \leq 1/g(x)$ for all positive polynomials $g(x)$ and sufficiently large x. In our case, we consider that two distributions are computationally indistinguishable if the statistical distance between them is negligible.

is a *valid plan* if it is indeed a plan, and we say that π is an *invalid plan* if it is just some sequence of actions which does not lead from the initial state to some goal state – either because it is not applicable in the initial state, because it does not lead to the goal, or because a single transition is invalid. Similarly, we extend this notion to transitions: a transition (s_{i-1}, a_i, s_i) is invalid if $s_{i-1}[\![a_i]\!]$ is undefined or $s_{i-1}[\![a_i]\!] \neq s_i$.

ZK-BOUNDEDPLANEX: Step-by-Step

We formalize each step of our protocol next. A concrete example is available as a technical report (Corrêa, Büchner, and Christen 2022).

Step 0. P and V have as common input $\langle \Pi, k\rangle$, where $\Pi = \langle \mathcal{V}, \mathcal{A}, I, G\rangle$ and $k \leq O(\|\Pi\|^c)$ – for some constant $c \in \mathbb{N}$. The prover P claims to know a plan π for Π with $|\pi| \leq k$.

Step 1. P transforms Π into some new task $\hat{\Pi}$ so it is computationally hard for V to correctly identify actions and variables. To do this, P transforms the task in five different phases: (a) add a "dummy" action to the task that can be applied in any state without changing it; (b)–(c) make actions indistinguishable by padding preconditions and effects until they all have the same structure; (d) permute truth values of variables to hide their identity; and (e) define specific initial and goal states to allow deterministically checking that the plan starts from the expected initial state and ends in a valid goal state. We detail each phase next.

Starting from Π, each of the following phases creates a new planning task that is a modification of the previous one. We write Π_a for the task created in phase (a), Π_b for the task created in phase (b), and so on.

(a) P introduces a dummy action a_dummy, where

$$\mathit{pre}(a_\text{dummy}) = \mathit{eff}(a_\text{dummy}) = \{\}$$

and then creates the planning task $\Pi_\text{a} = \langle \mathcal{V}, \mathcal{A}_\text{a}, I, G\rangle$, where $\mathcal{A}_\text{a} = \mathcal{A} \cup \{a_\text{dummy}\}$.
P then uses the action a_dummy to *pad* any plan π with $|\pi| < k$ until it has length k. This allows our protocol to decide $\langle \Pi, k\rangle \in$ BOUNDEDPLANEX for tasks with plans strictly smaller than k while not revealing the length of the plan. The key property of a_dummy is, that its application leaves the given state unchanged and the existence of a_dummy does therefore not affect plan existence.

(b) To ensure that all actions look similar, every action must 1) have the same number of mutual variables across precondition and effect and 2) precondition (resp. effect) must have the same number of literals. We delay the second requirement to part (c).
More formally, let $m(a) = |\mathit{vars}(\mathit{pre}(a)) \cap \mathit{vars}(\mathit{eff}(a))|$ denote the number of mutual variables in the precondition and effect of a given action a. Our goal in this part of the transformation is to obtain a set of actions $\mathcal{A}_\text{b}$ such that for every pair of action $a_1, a_2 \in \mathcal{A}_\text{b}$ it holds that $m(a_1) = m(a_2)$.
To achieve this, let $m^* = \max_{a \in \mathcal{A}_\text{a}} m(a)$. For every action $a \in \mathcal{A}_\text{a}$, introduce $n = m^* - m(a)$ new variables $m_1^a, \dots, m_n^a$, and two new actions $a^\perp$ and $a^\top$ such that

$$\begin{aligned}
\mathit{pre}(a^\perp) &= \mathit{pre}(a) \cup \{\neg m_i^a | 1 \leq i \leq n\},\\
\mathit{eff}(a^\perp) &= \mathit{eff}(a) \cup \{m_i^a | 1 \leq i \leq n\},\\
\mathit{pre}(a^\top) &= \mathit{pre}(a) \cup \{m_i^a | 1 \leq i \leq n\}, \text{ and}\\
\mathit{eff}(a^\top) &= \mathit{eff}(a) \cup \{\neg m_i^a | 1 \leq i \leq n\}.
\end{aligned}$$

These actions change all newly introduced variables in unison, which is also the reason why P needs to duplicate the number of actions: one copy to simulate a in case they are all $\perp$, and one copy to simulate a in case they are all $\top$. Note that no other action affects the variables m_i^a and consequently all m_i^a always have the same value $\top$ or $\perp$. The initial state I remains unchanged because P considers all newly introduced variables to be $\perp$ initially, as they are not in I. The goal G also remains unchanged because the newly introduced variables do not influence the reachability of G. In summary, this first transformation leads to $\Pi_\text{b} = \langle \mathcal{V}_\text{b}, \mathcal{A}_\text{b}, I, G\rangle$ where

$$\begin{aligned}
\mathcal{V}_\text{b} &= \mathcal{V} \cup \bigcup_{a \in \mathcal{A}_\text{a}} \{m_i^a | 1 \leq i \leq m^* - m(a)\} \text{ and}\\
\mathcal{A}_\text{b} &= \{a^\perp \mid a \in \mathcal{A}_\text{a}\} \cup \{a^\top \mid a \in \mathcal{A}_\text{a}\}.
\end{aligned}$$

This phase can be done in $O(|\mathcal{V}||\mathcal{A}|)$ steps and increases the representation size of the task by $O(|\mathcal{V}||\mathcal{A}|)$.

(c) We continue by padding the action preconditions and effects so that all actions have the same number of preconditions and the same number of effects. For $a \in \mathcal{A}_\text{b}$, let $p(a) = |\mathit{pre}(a)|$ and $e(a) = |\mathit{eff}(a)|$. Consider Π_b as described above and let $p^* = \max_{a \in \mathcal{A}_\text{b}} p(a)$ and $e^* = \max_{a \in \mathcal{A}_\text{b}} e(a)$. For every action $a \in \mathcal{A}_\text{b}$, introduce $m = p^* - p(a)$ new variables $p_1^a, \dots, p_m^a$ and $n = e^* - e(a)$ new variables $e_1^a, \dots, e_n^a$. Furthermore, define a_c such that

$$\begin{aligned}
\mathit{pre}(a_\text{c}) &= \mathit{pre}(a) \cup \{\neg p_i^a \mid 1 \leq i \leq m\} \text{ and}\\
\mathit{eff}(a_\text{c}) &= \mathit{eff}(a) \cup \{\neg e_i^a \mid 1 \leq i \leq n\}.
\end{aligned}$$

In doing so, all newly introduced variables only occur as negative literals after this transformation of the planning task. By leaving them out of the initial state as well, P ensures that all actions remain applicable and all effects can only make them false, so they never change their value. The goal also remains unchanged so the newly introduced variables have no influence on the solvability of the task. In summary, this transformation leads to $\Pi_\text{c} = \langle \mathcal{V}_\text{c}, \mathcal{A}_\text{c}, I, G\rangle$ where

$$\begin{aligned}
\mathcal{V}_\text{c} &= \mathcal{V}_\text{b} \cup \bigcup_{a \in \mathcal{A}_\text{b}} \{p_i^a \mid 1 \leq i \leq p^* - p(a)\}\\
&\quad \cup \bigcup_{a \in \mathcal{A}_\text{b}} \{e_i^a \mid 1 \leq i \leq e^* - e(a)\} \text{ and}\\
\mathcal{A}_\text{c} &= \{a_\text{c} \mid a \in \mathcal{A}_\text{b}\}
\end{aligned}$$

This phase also takes $O(|\mathcal{V}||\mathcal{A}|)$ steps and increases the task representation by the same amount.

(d) To hide the identity of the original Π in its transformed version Π_c, P creates a permuted version of Π_c using a uniformly chosen function ρ that randomly permutes variables and their truth value (i.e., positive to negative and vice-versa) at the initial state I. The truth value of each variable is also permuted consistently in every place of occurrence (e.g., goal and actions). With some abuse of notation, we write $\rho(\mathcal{V}_c)$, $\rho(\mathcal{A}_c)$, $\rho(I)$, and $\rho(G)$ to denote the permuted versions of $\mathcal{V}_c$, $\mathcal{A}_c$, I, and G according to ρ.

Let $\Pi_d = \langle \mathcal{V}_d, \mathcal{A}_d, I_d, G_d \rangle$ where

$$\begin{aligned} \mathcal{V}_d &= \rho(\mathcal{V}_c), \\ \mathcal{A}_d &= \rho(\mathcal{A}_c), \\ I_d &= \rho(I), \text{ and} \\ G_d &= \rho(G). \end{aligned}$$

This phase does not influence the size of the task representation but takes time $O(|\mathcal{V}_c|)$ which is polynomial in $|\mathcal{V}|$, as discussed in the previous transformation.

(e) In this last transformation, the task is changed to have a specific initial state and a unique goal state. To achieve this, P introduces two new variables v_I and v_*, and two new actions a_I and a_*, where

$$\begin{aligned} \mathit{pre}(a_I) &= \{v_I\} \cup \{\neg v \mid v \in \mathcal{V}_d\}, \\ \mathit{eff}(a_I) &= \{\neg v_I\} \cup I_d, \\ \mathit{pre}(a_*) &= \{\neg v_I\} \cup G_d, \text{ and} \\ \mathit{eff}(a_*) &= \{v_*\} \cup \{\neg v \mid v \in \mathcal{V}_d\}. \end{aligned}$$

P changes the initial state to $I_e = \{v_I\}$ and the goal to $G_e = \{v_*\} \cup \{\neg v \mid v \in \mathcal{V}_d\}$. To ensure that only a_I is applicable in I_e, P adds $\neg v_I$ to the preconditions of all actions and therefore defines a_e for all $a \in \mathcal{A}_d$ where

$$\begin{aligned} \mathit{pre}(a_e) &= \mathit{pre}(a) \cup \{\neg v_I\} \text{ and} \\ \mathit{eff}(a_e) &= \mathit{eff}(a). \end{aligned}$$

In summary, this last transformation leads to $\Pi_e = \langle \mathcal{V}_e, \mathcal{A}_e, I_e, G_e \rangle$ where

$$\begin{aligned} \mathcal{V}_e &= \mathcal{V}_d \cup \{v_I, v_*\} \text{ and} \\ \mathcal{A}_e &= \{a_e \mid a \in \mathcal{A}_d\}. \end{aligned}$$

This transformation can be done in polynomial time, too.

Let $\hat{\Pi} = \langle \hat{\mathcal{V}}, \hat{\mathcal{A}}, \hat{I}, \hat{G} \rangle$ where $\hat{\mathcal{V}} = \mathcal{V}_e$, $\hat{\mathcal{A}} = \mathcal{A}_e$, $\hat{I} = I_e$, and $\hat{G} = G_e$. The task $\hat{\Pi}$ is our *transformed task* and it will be used through the rest of the protocol. The overall transformation from Π to $\hat{\Pi}$ can be done in time polynomial in $\|\Pi\|$, and $\hat{\Pi}$ is only polynomially larger than Π. In the rest of the paper, we say that σ is a *transformation function* such that $\sigma(\Pi) = \hat{\Pi}$. Note that σ can be defined simply by describing each step taken by P, which takes polynomial time and space.

P transforms the plan π for Π of length $|\pi| \leq k$ into a plan $\hat{\pi} = \langle a_1, \dots, a_{k+2} \rangle$ for $\hat{\Pi}$. This transformation is fairly straightforward so, due to space limits, we do not discuss it here. However, we note that the new initial and goal states in $\hat{\Pi}$ imply that $a_1 = a_I$ and $a_{k+2} = a_*$. As these two actions were not in the original task Π, $\hat{\pi}$ has two steps more than π. To simplify notation, let $\ell = k + 2$ and thus $\hat{\pi} = \langle a_1, \dots, a_\ell \rangle$.

Step 2. P creates the sequence S of states:

$$S = \langle s_0, \dots, s_\ell \rangle,$$

where s_0 and s_ℓ correspond to our unique initial and goal states, respectively, and $s_{i-1}[\![a_i]\!] = s_i$ for all $1 \leq i \leq \ell$. In words, sequence S corresponds to the states visited by $\hat{\pi}$, if $\hat{\pi}$ is indeed a valid plan.

Next, P commits the task $\hat{\Pi}$ and each element in $\hat{\pi}$ and S individually using some arbitrarily chosen commitment scheme. Recall that we write *Commit*(x) to represent the encrypted commitment of some object x, and *Commit*(L) represents the element-wise commitment of some sequence L. With some abuse of notation, we define $\mathit{Commit}(\hat{\Pi}) = \langle \mathit{Commit}(\hat{\mathcal{V}}), \mathit{Commit}(\hat{\mathcal{A}}), \mathit{Commit}(\hat{I}), \mathit{Commit}(\hat{G}) \rangle$ where each $\hat{\mathcal{V}}$, $\hat{\mathcal{A}}$, $\hat{I}$, and $\hat{G}$ are also commited element-wise.

Finally, P sends *Commit*$(\hat{\Pi})$, *Commit*$(\hat{\pi})$, and *Commit*(S) to V.

Step 3. If $|\mathit{Commit}(\hat{\pi})| > \ell$, the verifier rejects the protocol; otherwise it picks a random bit $b \in \{0, 1\}$ and sends it to P.

If $b = 0$, P reveals the function σ to V together with all keys to open *Commit*$(\hat{\Pi})$. V opens *Commit*$(\hat{\Pi})$ to obtain $\hat{\Pi}$ and checks if $\sigma(\Pi) = \hat{\Pi}$. If this is the case, V accepts the protocol; otherwise it rejects.

If $b = 1$, P reveals s_0 and s_ℓ from $\mathit{Commit}(S) = \langle \mathit{Commit}(s_0), \dots, \mathit{Commit}(s_\ell) \rangle$ to V.

Step 4. V verifies that $s_0 = \{v_I\}$, the expected initial state, and $s_\ell = \{v_*\}$, the expected goal state. If either of these comparisons fails, V rejects the protocol. Otherwise, it uniformly chooses an integer $m \in \{1, \dots, \ell\}$ by flipping $\log \ell$ fair coins and sends m to P. The value of m corresponds to the m-th transition of $\hat{\pi}$ that V wants to verify on its own.

Step 5. The prover reveals s_{m-1} and s_m from *Commit*(S) and a_m from *Commit*$(\hat{\pi})$. It also reveals $\hat{\mathcal{V}}$ and a_m from *Commit*$(\hat{\mathcal{A}})$.[4] Finally, V compares the action a_m in $\hat{\pi}$ with the action obtained from $\hat{\mathcal{A}}$, which should be the same, and verifies that all variables used in s_{m-1}, a_m and s_m are indeed in $\hat{\mathcal{V}}$. If any of these checks fail, V rejects the protocol.

Last, V checks locally if $s_{m-1}[\![a_m]\!] = s_m$. If this also passes, V accepts the protocol. Otherwise, V rejects it. □

All computations of the protocol can be done in polynomial-time by the prover and the verifier. More explicitly, the verifier needs to pick $\log \ell$ random bits, transform the task and compare the resulting task, compare two states to their expected value, and check a transition locally. This can all be easily done in polynomial-time by a probabilistic algorithm.

[4]Note that P reveals two copies of a_m: one from the transition and one from the set of actions $\hat{\mathcal{A}}$. This is done so the verifier can be sure that the action used in the transition is indeed part of the task $\hat{\Pi}$.

In fact, all the computations performed by the prover P are also polynomial. This means that any prover that simply happens to have the solution (e.g., it obtained this information from a third-party) could participate in the protocol. Note that this is only possible because the transformations done in Step 1 can also be applied to the plan.

The protocol also only needs a constant number of rounds. Together, the constant number of rounds and the polynomial-time steps make ZK-BOUNDEDPLANEX efficient enough to be performed in practice.

Completeness and Soundness

We first analyze the completeness and soundness of our protocol. Throughout the rest of the paper, let $x = \langle \Pi, k \rangle$. We rewrite Equations (1) and (2) in terms of our language BOUNDEDPLANEX:

$$x \in \text{BOUNDEDPLANEX} \implies \exists P \Pr[\langle V, P \rangle(x) = 1] \geq \frac{2}{3}$$

$$x \notin \text{BOUNDEDPLANEX} \implies \forall P \Pr[\langle V, P \rangle(x) = 1] \leq \frac{1}{3}$$

where the first implication refers to the protocol's completeness and the second to the protocol's soundness.

We discuss completeness first. The verifier can only reject a valid input $\langle \Pi, k \rangle$ in one of the following cases (1) P provides an invalid transformation function σ (Step 3); (2) $\hat{\pi}$ does not start in $s_I = \{v_I\}$ or does not end in $s_* = \{v_*\}$ (Step 4); (3) one of the transitions in $\hat{\pi}$ is invalid (Step 5). Under the assumption of an all-powerful prover, any honest prover P that can compute a plan can also trivially satisfy the cases above. Hence, our protocol has completeness of

$$\Pr[\langle V, P \rangle(x) = 1] = 1.$$

For soundness, we denote by $\overline{P}$ an arbitrary *dishonest* prover that claims to have a plan for Π but in reality it does not. Consider how $\overline{P}$ can try to trick V into accepting x although $x \notin$ BOUNDEDPLANEX.

First, $\overline{P}$ can try to trick V by performing a wrong task transformation in Step 1. The verifier V would then reject it with probability $1/2$, the probability of $b = 0$ in Step 3. By repeating the protocol t times, the probability of V accepting when $x \notin$ BOUNDEDPLANEX decreases to $(1 - 1/2)^t$ which is smaller than $1/3$ when $t \geq 2$.

Second, $\overline{P}$ can perform a correct task transformation in Step 1 but provide an invalid plan in Step 2. We write $\overline{\pi}$ to denote this invalid plan. Note that the plan $\overline{\pi}$ is invalid when there is at least one wrong transition in $\overline{\pi}$. This means that for some transition (s_{i-1}, a_i, s_i), the action a_i is either not applicable in s_{i-1} or it does not lead to s_i. The verifier can only spot this mistake if $b = 1$ in Step 3 and $m = i$ in Step 5. Hence, the probability of V accepting x is $1 - 1/2\ell$. By repeating the protocol 3ℓ times, this probability decreases to $(1 - 1/2\ell)^{3\ell} < 1/3$ since $\ell \geq 2$ by definition. As $\ell = k + 2$ and k is polynomial in $\|\Pi\|$, the protocol only needs to be repeated a polynomial number of times.

Note that, in practice, P can optimize parts of the protocol in the case it needs to be repeated several times. For example, steps (a)–(c) in Step 1 can be computed only once and be cached for future reuse. As Step 1 is the most expensive step of the protocol – although it still runs in polynomial time – this caching can be very beneficial in practice.

Proof of Zero-Knowledge

We now prove that the ZK-BOUNDEDPLANEX protocol is zero-knowledge. We need to prove that there exists a polynomial-time probabilistic simulator M^* with an output distribution that is computationally indistinguishable from the output distribution of $\langle V^*, P \rangle(x)$, where V^* is a probabilistic polynomial-time verifier with an arbitrary strategy (i.e., it does not need to follow the protocol). Recall that the zero-knowledge definition is conditioned to cases where the input x belongs to the language, so M^* only needs to produce an output that is computationally indistinguishable from the output of $\langle V^*, P \rangle(x)$ for the case of an honest prover P.

Our simulation relies on the common technique that the simulator M^* *guesses* the random choices of V^* in advance. In our case, M^* guesses the values of b and m. If it does not guess correctly, then it simply *restarts* the simulation (e.g., Blum 1986; Arora and Barak 2009). The trick here is that M^* only needs to restart the simulation a polynomial number of times, and it only outputs once it guesses b and m correctly. Although it is possible that an exponential (for example) number of restarts is needed, such corner cases occur with negligible probability, so they are not sufficient to distinguish both distributions. In such cases, M^* can just abort the process entirely after a certain number of steps.

Next, we explain how M^* works. We consider that M^* executes V^* as a subroutine, so it can send messages and simulate V^* independently of its strategy. The simulation works as follows: M^* begins by randomly guessing $b' \in \{0, 1\}$ and $m' \in \{1, \ldots, \ell\}$. It then starts to act as a prover P by generating $\hat{\Pi} = \langle \hat{\mathcal{V}}, \hat{\mathcal{A}}, \hat{I}, \hat{G} \rangle$ as in Step 1 using some function σ. As previously argued, this transformation takes polynomial time so M^* can do it within its time limits. The only difference to Step 1 is that M^* does not compute a plan $\hat{\pi}$ for $\hat{\Pi}$.

It then produces $\hat{\pi}$ and S but differently than P would, as M^* does not know a plan $\hat{\pi}$. It starts with the sequence of states S and it splits this procedure into two cases based on whether $m' \in \{1, \ell\}$ or not. First, assume $m' \notin \{1, \ell\}$. Let a' be a randomly selected action of $\hat{\Pi}$ and let s' be a state such that $s' \models \mathit{pre}(a')$, $v_I \notin s'$, $v_* \notin s'$, and all other variables in $\hat{\mathcal{V}}$ are set to true or false uniformly at random. Then M^* defines $S = \langle s_0, \ldots, s_\ell \rangle$ where $s_0 = s_I = \{v_I\}$, $s_\ell = s_* = \{v_*\}$, $s_{m'-1} = s'$, $s_{m'} = s'[\![a']\!]$, and all other states in S are chosen randomly (by uniformly assigning each variable to true or false at each state). The sequence $\hat{\pi}$ is defined as $\hat{\pi} = \langle a_1, \ldots, a_\ell \rangle$ where $a_{m'} = a'$ and all other actions are chosen randomly from $\hat{\mathcal{A}}$. Note that $\hat{\pi}$ generated by M^* is (very) probably not a valid plan, but we use the same notation and terminology for simplicity.

When $m' \in \{1, \ell\}$, instead of choosing a' randomly, M^* must select a specific action according to the value of m. If $m' = 1$, then $a_1 = a_I$, and we define $s_0 = s_I$ and $s_1 = s_0[\![a_I]\!]$ in S. If $m' = \ell$, then $a_\ell = a_*$, $s_\ell = s_*$, $s_{\ell-1}$ is defined such that $s' \models \mathit{pre}(a_*)$, $v_I, v_* \notin s'$ and all other variables are set to true or false uniformly at random. The rest of the procedure is the same.

After that, M^* commits $\hat{\pi}$, S, and $\hat{\Pi}$ and sends them to V^*, as P would do at the end of Step 2. V^* picks $b \in \{0,1\}$ and sends to M^*. If $b = b'$, M^* continues the simulation; otherwise it restarts it.

In the case where $b = b' = 0$, M^* can send V^* the function σ used to transform Π into $\hat{\Pi}$ so V^* can check that $\sigma(\Pi) = \hat{\Pi}$. This can be trivially done by M^* as it created σ on its own, so V^* will accept the protocol. In this case, the interaction between V^* and P and the interaction between V^* and M^* are indistinguishable.

In the case where $b = b' = 1$, M^* continues simulating P (by sending the keys to s_0, s_ℓ, etc.) until V^* sends m to M^*. If $m \neq m'$, M^* restarts the simulation. Otherwise, it reveals the m-th transition to V^*, as P would do. Since this transition was constructed in a way that it is guaranteed to be valid, V^* will accept the protocol.

The key idea of the simulation is that the only transition a verifier can view is the one they randomly choose based on m. However, M^* only knows that the m'-th transition is valid. Therefore, M^* can only guarantee that V^* accepts when $m = m'$. Otherwise, it needs to restart the simulation.

Before proving that the protocol ZK-BOUNDEDPLANEX is zero-knowledge, we prove some lemmas.

Lemma 1. *Commit*$(\hat{\Pi})$, *Commit*$(\hat{\pi})$*, and Commit*(S) *computed by M^* are computationally indistinguishable from Commit*$(\hat{\Pi})$, *Commit*$(\hat{\pi})$*, and Commit*(S) *computed by a prover P.*

Proof. Note that *Commit*$(\hat{\Pi})$ is computed identically by M^* and P, so they are trivially indistinguishable. *Commit*$(\hat{\pi})$ and *Commit*(S) are computed differently (M^* generates random states and actions, while P uses the plan for that) but under the assumption of a commitment scheme that is hiding and binding, they are computationally indistinguishable when encrypted. □

Lemma 2. *The probability that $b = b'$ and $m = m'$ is at least $1/2\ell$.*

Proof. Given Lemma 1 and the fact that the elements in *Commit*(S) are also indistinguishable between themselves (analogously to *Commit*$(\hat{\pi})$), we can assume that a probabilistic polynomial-time V^* chooses b and m in an oblivious manner (i.e., independent of the content of the encrypted information). For simplicity, we can then assume that it selects b and m even before the protocol with M^* or P has started. Thus, it follows that Pr$[b = b'$ and $m = m'] = 1/2\ell$. □

Lemma 3. *Let (s_{m-1}, a_m, s_m) be the single transition revealed to V^*. Then, (s_{m-1}, a_m, s_m) computed by M^* is computationally indistinguishable from (s_{m-1}, a_m, s_m) computed by a prover P.*

Proof. Recall that the transformation of Step 1 makes actions indistinguishable (except for a_I and a_*) by padding the actions and permuting variables and truth values. Thus, the action a_m revealed to V^* is indistinguishable from any other action – and so are states s_{m-1} and s_m. However, (s_{m-1}, a_m, s_m) was computed differently by M^* and by P. When computed by P, it is a transition occurring in $\hat{\pi}$. When computed by M^*, it is a randomly selected action in $\hat{\mathcal{A}}$ with randomly generated states s_{m-1} and s_m. But since variables are permuted according to a function chosen uniformly at random by both M^* and P, both (s_{m-1}, a_m, s_m) computed by M^* and (s_{m-1}, a_m, s_m) computed by a prover P are computationally indistinguishable to each other as well. □

We are now set to prove that ZK-BOUNDEDPLANEX is indeed a zero-knowledge protocol.

Theorem 4. *The protocol* ZK-BOUNDEDPLANEX *is zero-knowledge.*

Proof. In other words, we want to prove that the output of the interaction between V^* and M^* is indistinguishable from the output of the interaction between V^* and P for any probabilistic polynomial-time verifier strategy V^*, when conditioned to cases where $x \in$ BOUNDEDPLANEX, and that M^* runs in probabilistic polynomial time.

The fact that the interactions are computationally indistinguishable follows directly from Lemmas 1 and 3, as these are the only messages revealed by the prover to V^*.

It remains to show that M^* runs in probabilistic polynomial time. As shown in Lemma 2, Pr$[b = b'$ and $m = m'] = 1/2\ell$ and thus the expected number of restarts M^* makes until $b = b'$ and $m = m'$ is 2ℓ. Therefore, the expected running time of M^* is $2\ell\, t(|x|)$, where $t(|x|)$ is the running time of V^* with input x. There is a very small probability that the algorithm needs more than polynomial number of restarts. To solve this, M^* can output some default value after $|x|$ restarts. This adds $1/2\,(\ell-1/\ell)^{|x|}$ statistical distance between the distributions, but this is negligible as $1/2\,(\ell-1/\ell)^{|x|} \leq 1/q(|x|)$ for any positive polynomial q and sufficiently large $|x|$. As $\ell \leq O(\|\Pi\|^c)$ for some constant c, we have that $2\ell\, t(|x|) \leq O(\|\Pi\|^c)t(|x|)$ and since $t(|x|)$ is polynomial in $\|\Pi\|$, too, we have $O(\|\Pi\|)t(|x|) \leq O(\mathit{poly}(\|\Pi\|))$. □

Conclusion

We introduce ZK-BOUNDEDPLANEX, a zero-knowledge protocol to prove plan existence for tasks with polynomially-long plans. At the core of our protocol lies a transformation of the original planning task Π to the new task $\hat{\Pi}$, where variables are permuted to hide their identities and actions are made indistinguishable. After the prover commits to $\hat{\Pi}$ and plan $\hat{\pi}$, the verifier makes a probabilistic choice to either check the task transformation or the transformed plan. In the latter case, the verifier checks a single transition as well as the initial and goal states of $\hat{\pi}$. If all checks pass, the verifier accepts the protocol.

ZK-BOUNDEDPLANEX runs in a constant number of rounds, with every step being computable in time polynomial in $\|\Pi\|$; the verifier has high confidence that a plan exists after repeating the protocol a polynomial number of times, while the prover is absolutely sure that no knowledge about the plan is revealed.

Acknowledgments

We especially thank André G. Pereira for giving us feedback on earlier versions of this work, and Adi Shamir for pointing us to relevant literature on zero-knowledge proofs. We also thank Simon Dold and Malte Helmert for helpful discussions.

This work was funded by the Swiss National Science Foundation (SNSF) as part of the project "Certified Correctness and Guaranteed Performance for Domain-Independent Planning" (CCGP-Plan). Furthermore, this research was also partially supported by TAILOR, a project funded by EU Horizon 2020 research and innovation programme under grant agreement no. 952215.

References

Arora, S.; and Barak, B. 2009. *Computational Complexity: A Modern Approach*. Cambridge University Press.

Babai, L. 1985. Trading Group Theory for Randomness. In Sedgewick, R., ed., *Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing (STOC '85)*, 421–429. ACM Press.

Ben-Or, M.; Goldreich, O.; Goldwasser, S.; Håstad, J.; Kilian, J.; Micali, S.; and Rogaway, P. 1988. Everything Provable is Provable in Zero-Knowledge. In Goldwasser, S., ed., *Proceedings of the Eighth Annual International Cryptology Conference (CRYPTO 1988)*, volume 403 of *Lecture Notes in Computer Science*, 37–56. Springer.

Blum, M. 1986. How to Prove a Theorem So No One Else Can Claim It. In *Proceedings of the International Congress of Mathematicians (ICM 1986)*, volume 2, 1444–1451.

Brafman, R. I. 2015. A Privacy Preserving Algorithm for Multi-Agent Planning and Search. In Yang, Q.; and Wooldridge, M., eds., *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI 2015)*, 1530–1536. AAAI Press.

Bylander, T. 1994. The Computational Complexity of Propositional STRIPS Planning. *Artificial Intelligence*, 69(1–2): 165–204.

Corrêa, A. B.; Büchner, C.; and Christen, R. 2022. Zero-Knowledge Proofs for Classical Planning Problems: Concrete Example. Technical Report CS-2022-002, University of Basel, Department of Mathematics and Computer Science.

Fikes, R. E.; and Nilsson, N. J. 1971. STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. *Artificial Intelligence*, 2: 189–208.

Goldreich, O. 2001. *The Foundations of Cryptography — Volume 1: Basic Techniques*. Cambridge University Press.

Goldreich, O.; Micali, S.; and Wigderson, A. 1986. Proofs that Yield Nothing But their Validity and a Methodology of Cryptographic Protocol Design (Extended Abstract). In *Proceedings of the Twenty-Seventh Annual Symposium on Foundations of Computer Science (FOCS 1986)*, 174–187. IEEE Computer Society.

Goldwasser, S.; Micali, S.; and Rackoff, C. 1985. The Knowledge Complexity of Interactive Proof-Systems (Extended Abstract). In Sedgewick, R., ed., *Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing (STOC '85)*, 291–304. ACM Press.

Helmert, M. 2006. The Fast Downward Planning System. *Journal of Artificial Intelligence Research*, 26: 191–246.

Hoffmann, J.; and Nebel, B. 2001. The FF Planning System: Fast Plan Generation Through Heuristic Search. *Journal of Artificial Intelligence Research*, 14: 253–302.

Impagliazzo, R.; and Yung, M. 1987. Direct Minimum-Knowledge Computations. In Pomerance, C., ed., *Proceedings of the Eighth Annual International Cryptology Conference (CRYPTO 1987)*, volume 293 of *Lecture Notes in Computer Science*, 40–51. Springer.

Lund, C.; Fortnow, L.; Karloff, H. J.; and Nisan, N. 1992. Algebraic Methods for Interactive Proof Systems. *Journal of the ACM*, 39(4): 859–868.

Shamir, A. 1992. IP = PSPACE. *Journal of the ACM*, 39(4): 869–877.

Shen, A. 1992. IP = PSPACE: Simplified Proof. *Journal of the ACM*, 39(4): 878–880.

Torreño, A.; Onaindia, E.; Komenda, A.; and Štolba, M. 2017. Cooperative Multi-Agent Planning: A Survey. *ACM Computing Surveys*, 50(6): 1–32.

Tožička, J.; Štolba, M.; and Komenda, A. 2017. The Limits of Strong Privacy Preserving Multi-Agent Planning. In Barbulescu, L.; Frank, J.; Mausam; and Smith, S. F., eds., *Proceedings of the Twenty-Seventh International Conference on Automated Planning and Scheduling (ICAPS 2017)*, 297–305. AAAI Press.