# A Framework to prove Strong Privacy in Multi-Agent Planning

**Patrick Caspari,**[1] **Robert Mattmüller,**[2] **Tim Schulte**[3]
Department for Computer Science
Albert-Ludwigs-Universität Freiburg
Georges-Köhler-Allee 52
79110 Freiburg, Germany
[1]caspari@informatik.uni-freiburg.de,
[2]mattmuel@informatik.uni-freiburg.de,
[3]schultet@tf.uni-freiburg.de

**Abstract**

Privacy in multi-agent planning is an important and well-discussed problem. But it is difficult to prove whether an algorithm allows the deduction of private values. We introduce a framework that allows us to prove whether the messages transmitted during a Multi-Agent planning instance maintain strong privacy of the problem. It can be verified for an algorithm in general or for a specific execution of the algorithm on a planning task. We then use the framework to show that an A$^*$-based secure-MAFS algorithm is in general not strongly privacy preserving.

## Introduction

With the commercialization of automated machines and increasingly flexible and automated industry as well as the increase in computing power, the demand for algorithms that allow cooperative planning is high. But especially in the industry, where business secrets can be the foundation of success and their disclosure might mean the loss of many jobs, any cooperation can only be secure with tight guarantees for the privacy of their assets.

Several concepts of privacy exist and are used, depending on the degree of privacy that is necessary in the adopted domain. On the lower end of this range stands the notion of *weak privacy*, which is satisfied if an algorithm doesn't explicitly communicate private values to partners. While this is easy to verify, it doesn't account for the fact, that the transmitted information might still be used to deduce private information. On the other end, *strong privacy* requires that no agent can deduce the value or even the existence of another agent's private variables (Brafman 2015). While this definition is very strong, it is unclear how to verify whether an algorithm actually upholds it.

With the definition of PST-indistinguishability (Beimel and Brafman 2018), a verifyable lower bound for strong privacy was proposed. It states that all messages sent by an agent as well as their order must be uniquely defined by the public search tree. Other privacy definitions demand that the identity of the other agents is not revealed (Faltings, Léauté, and Petcu 2008) or that agents can't construct an upper bound on the number of objects of a type (Maliah, Shani, and Stern 2018).

We propose a framework that allows us to verify whether an execution of a multi-agent planning algorithm on a specific problem leaks private information. We then abstract from the specific problem and use the same framework to prove whether any exectution of a planning algorithm can leak private information. To further visualize this, we apply it to an A$^*$-based secure-MAFS algorithm and show that it leaks private information.



## Multi-Agent Planning

A multi-agent planning problem (Brafman and Domshlak 2008) is defined as a 4-tupel $\Pi = (P, \{A_i\}_{i=1}^N, I, G)$ where $P$ is a set of propositions, $A_i$ is a set of actions of an agent $i$, $I$ the initial state and $G$ a set of goal states. A part of the propositions $P^{\text{pub}} \subseteq P$ are shared among all agents. Other propositions $P^{\text{priv},i}$ are private to a single agent $i$. The existence of a private variable is only known to the respective agent and their value can only be affected by this agent's actions.

A *global* state $s$ is defined as a valuation of all propositions $p \in P$. A state $s^i$ of agent $i$ is a valuation of all propositions $P^i = P^{\text{pub}} \cup P^{\text{priv},i}$ that affect this agent, that is all public propositions and those private to agent $i$. A *public* state $s^{\text{pub}}$ only contains the valuations of the public variables. As a consequence, each public state describes a set of global states that differ in the private states of the individual agents.

We call an action *public* if the effect contains public variables. If the effect only contains private variables, the action is *private*. From each private action, we can generate a *public projection* $a^{\text{pub}}$, by stripping its precondition and effect of all private propositions.

During the execution of a planning algorithm $X$, the planner generates a *Planning Tree* $PT_X(\Pi)$. The planning tree is a directed graph which represents the paths and states that the algorithm expanded during the search. Each node of the graph corresponds to an expanded state; every edge represents the occurence of an action. Since an algorithm might expand states several times during its execution, sev-

eral nodes can refer to the same state.
We define a planning tree as a 6-tupel $PT = (N, E, i, V, D, d)$ where

- $N = \{n_1, \dots n_{|N|}\}$ is a set of nodes.
- $E = \{e_1, \dots e_{|E|}\}$ is a set of directed labeled edges. Each edge $e_k = (n^{\text{pre}}, n^{\text{post}}, \text{lab})$ consists of a predecessor node $n^{\text{pre}}$, a successor node $n^{\text{post}}$ and a label $\text{lab}$ corresponding to the action labels of $\Pi$.
- $i : N \to [0, |N|-1]$ is the indexing function that assigns a unique index to each node.
- $V : N \to 2^P$ is the valuation function that assigns a valuation over the propositions in $\Pi$ to each node.
- $D$ is a set of additional descriptions that may apply to a state. For our purposes, it models additional information that is transmitted during the application of a Multi-Agent Planning algorithm. This might for example be the path cost $g(s)$ or the heuristic function $h(s)$.
- $d : N \to D$ is the description function assigns the additional information to a node.

The planning tree is generated iteratively. At first, the planning tree only consists of the initial state. In each step, an new state is added to the tree by application of an action to one of the currently expanded states. The index of the nodes represents the order in which they were added to the planning tree. The initial state always carries the index $0$.

We use an additional structure that we call the *Transmitted Tree* $\mathbf{T}$. It models the content of all messages transmitted between two agents. The transmitted tree has the same structure as the planning tree, with $\mathbf{T} = (N^T, E^T, i^T, V, D, d)$ and

$$N^T \subseteq N$$
$$E^T \subseteq T$$

The subsets $N^T$ and $E^T$ refer to the nodes and edges that were transmitted as messages by the algorithm. The indexing function $i^T$ denotes the order in which the messages were received. Since not necessarily all expanded nodes are transmitted, the indices can vary from those of the planning tree.

In a purely public problem, if all expanded states and applied actions are transmitted, the transmitted tree is identical to the planning tree.

## Strong Privacy

Brafman defined strong privacy as follows (Brafman 2015):
*"A variable or a specific value of a variable is strongly private if the other agents cannot deduce its existence from the information available to them."*
Following this definition, it is not only the value of a private variable or its function whithin the problem that must stay unknown to the observer; even the fact that a private variable exists and has any effect on the execution of the algorithm must be obscured. In epistemic terms, we can rephrase this definition:

***Definition 1*** *A variable is strongly private if an observer can't distinguish between a world where this variable exists and affects the execution of the algorithm, and one where it doesn't exist.*

Initially, this may seem like a paradox - if the variable can't have a visible effect, what would be the point in using it? However, the definition doesn't say that the worlds are identical, but only that an observing agent can't tell which world it is in.

To illustrate this difference, consider three friends playing a card game. They play a single round, which player A wins. What the other players don't know is that player A is a skilled cheater. In a fair game, the chances of A winning would have been $\frac{1}{3}$; since they cheated, it is $1$. Hence the *outcome* of the game differs from one where A plays fair. But from the perspective of B and C, the fact that they lost is in line with the assumption that the game was fair. So the other agents can't distinguish between a world where A cheated and one where they didn't.

The epistemic formulation implies another important property of strong privacy. The privacy of a value doesn't only depend on the algorithm but also on the observer. What is indistinguishable for one observer might be distinguishable for another.

For this, imagine that the card game from above is played with a non-standard deck of cards. Before the game, B looked through the cards and found out that two cards of each kind exist; player C didn't. Now A - still a cheater - has additional cards up their sleeve. They play out the third card of a kind, which is not supposed to exist. B's knowledge allows them to deduce that A cheated. From the perspective of C on the other hand the game still appears to be fair. So regarding C, the private property that A cheats is preserved; regarding B it isn't.

This shows that in order to discuss privacy in a multi-agent planning setting, we need to model the knowledge of an agent. In a general case it can be assumed that every agent knows what algorithm the others use and how that algorithm works. So we need a way to transform a planning algorithm into a knowledge model that we can check for consistency. Only then we can decide whether the execution of the algorithm and the exchanged messages are in accordance with the assumption that no private variables exist.

For simplicity, we regard the case of two agents: a sender $S$ and a receiver $R$. The sending agent expands its private search tree and sends messages to $R$ whenever the algorithm requires it. $R$ acts as an honest, but curious agent, meaning that they apply the algorithm in the predefined manner, but try to deduce the private variables of $S$ from the messages it receives.

We assume that the deduction occurs after the planning algorithm is complete, so that all messages are sent and the knowledge of $R$ is static. $R$ has the following knowledge, as illustrated in Figure 1:

- a set of actions
  $\mathbf{A}_S^{\text{pub}} : \{(a^{\text{pub}}, \text{pre}_a^{\text{pub}}, \text{post}_a^{\text{pub}}\}$
  that represents the public projections of the actions of $S$
- The *transmitted tree* $\mathbf{T}_S$ which models the messages exchanged between the agents. It contains the state descriptions and actions that were transmitted as part of the messages and are therefore known to be part of the planning

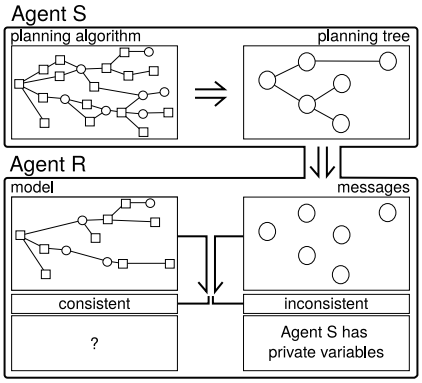


Figure 1: Schematic depiction of the deduction process.

tree.

- The *algorithm model* $\mathbf{M}_S$ which models the behavior of $S$'s algorithm.

Specifically, the knowledge model contains no private variables of $S$. If $R$ can construct a planning tree which is a possible product of the algorithm model and yields the messages that make up the transmitted tree, then $S$'s privacy is preserved. However, if such a planning tree doesn't exist, $R$'s knowledge isn't consistent with the actual execution of the algorithm. If we have a high trust in the algorithm model, the only possible reason for this inconsistency is that some private variable existed during the actual execution.

***Definition 2*** *We call the execution of a multi-agent planning algorithm on a planning problem* $\Pi$ publicly self-sufficient*, if a planning tree exists, which doesn't contradict the agent's algorithm model model or the transmitted tree and contains only public variables.*

Equivalently we could say that a problem is publicly self-sufficient if the transmitted messages and the algorithm model are consistent with the assumption that no private variables exist.

## Planning Logic

In order to prove whether tranmitted tree and algorihtm model are consistent, we need to represent both within a common framework, wich we call the *Planning Logic*. The planning logic is a set of propositions and logic rules that are able to model the execution of a multi-agent planning algorithm. It it based on the concept of planning as satisfiability (Kautz and Selman 1992). Since the valuations of the base propositions of the planning problem change from state to state, the logic needs a distinct set of base propositions for each state. Additionally, we need to model the actions with precondition and effect.

***Definition 3*** *Let* $\Pi$ *be a multi-agent planning problem and* $n_s$ *the number of states we want to consider. We call* $\mathcal{L}_\Pi$ *the* planning logic *induced by* $\Pi$ *over the set of propositional atoms*

$$P_{\mathcal{L}_\Pi} = \{ \begin{array}{ll} p_{s_k} & \forall\, p \in P;\ k = 1, \ldots, n_s \\ a_{s_k,s_j} & \forall\, a \in A_S;\ k, j = 1, \ldots, n_s \end{array} \} \tag{1}$$

For a planning problem with $|P| = n_p$ propositions and $|A_S| = n_a$ actions and $n_s$ reachable states, this yields a planning logic with $|P_{\mathcal{L}_\Pi}| = n_p \cdot n_s + n_a \cdot n_s^2$ propositional atoms.

Each state $s_k$ in $\Pi$ is defined by the valuation of the propositions $P$. We define each state in $\mathcal{L}_\Pi$ using a set of propositions $p_{s_k}$, with $p \in P$ and $k = 1, \ldots, n_s$. A positive literal $p_{s_k}$ means equivalently that the proposition $p$ evaluates as true in state $s_k$.

An action $a \in A_S$ is defined with a set of propositions $a_{s_k,s_j}$ and two functions $\mathrm{pre}_a(s_k)$ and $\mathrm{apply}_a(s_k, s_j)$. The literal $a_{s_k,s_j}$ denotes whether the action $a$ was used in the transition from predecessor state $s_k$ to successor state $s_j$. The function $\mathrm{pre}_a(s_k)$ returns the precondition of $a$ in the propositions $p_{s_k}$ of the provided predecessor state. Similarly, the function $\mathrm{apply}_a(s_k, s_j)$ returns the postcondition of $a$ in the propositions $p_{s_j}$ of the successor state and adds an additional term $p_{s_j} \leftrightarrow p_{s_k}$ for each base proposition $p$ not mentioned in the postcondition of $a$.

During or after the execution of an algorithm, the generated planning tree can be represented as a valuation of all propositions of the planning logic. Let $X$ be a planning algorithm and $\Pi$ a planning problem. We call $\mathcal{L}(\cdot)$ the function that transfers a planning tree or a transmitted tree into a logic formular based on the planning logic.

## Algorithm Models

In order to argue if the transmitted tree is compatible with the applied algorithm, we need a way to model the algorithm in the terms of the planning logic. To do this, we construct a set of axioms that define what planning trees this algorithm might produce.

Some axioms model properties that are common to all planning algorithms, for example that each node in the planning tree must be connected. Others may represent properties which are unique to the algorithm. An algorithm that searches the planning domain as a breadth-first search for example would need an axiom that defines that all reachable states are part of the planning tree.

In order to use the axioms we need to prove that they actually form a valid representation of the algorithm.

***Definition 4*** *Let* $X$ *be a multi-agent planning algorithm and* $A_X$ *a set of axioms.* $A_X$ *is a* sound axiomisation *of* $X$ *iff for any planning problem* $\Pi$

$$\forall PT \in \mathbf{PT}_X(\Pi)\ :\ \mathcal{L}(PT) \vDash A_X, \tag{2}$$

*where* $\mathbf{PT}_X(\Pi)$ *is the set of all planning trees the algorithm might generate from* $X$ *on the problem* $\Pi$*.*

This means that all planning trees generated from $\Pi$ with $X$

are consistent with the axioms in $A_X$. For a fully deterministic algorithm, $\mathbf{PT}_X(\Pi)$ contains only one entry. In a randomized algorithm, a repeated application of an algorithm $X$ on a problem $\Pi$ can lead to different planning trees. In that case, $\mathbf{PT}_X(\Pi)$ contains all possible planning trees that can be created that way.

The definition above guarantees that a set of axioms describes the algorithm. But it allows different axiomisations of the same algorithm. In fact, an empty set of axioms would be a sound axiomisation of any planning algorithm. This can be used to depict different levels of knowledge that an agent has. But in general, one would assume that an opposing agent has perfect knowledge about the applied algorithm. Accordingly, the knowledge model has to be a complete model of the applied algorithm.

***Definition 5*** *Let $X$ be a multi-agent planning algorithm and $A_X$ an axiomisation of this algorithm. Furthermore, let $\Pi$ be a planning problem, and $\mathbf{PT}_X(\Pi)$ the set of all planning trees that the algorithm might generate from $X$.*
*The axiomisation $A_X$ of $X$ is* complete *iff for any planning problem $\Pi$*

$$\begin{aligned} &\forall PT_\Pi \in \mathbf{PT}_X(\Pi) \ : \ \mathcal{L}(PT_\Pi) \vDash A_X \text{ and} \\ &\forall PT'_\Pi \notin \mathbf{PT}_X(\Pi) \ : \ \mathcal{L}(PT'_\Pi) \nvDash A_X \\ &\text{with } P_{\mathcal{L}(PT'_\Pi)} = P_{\mathcal{L}(PT_\Pi)} \end{aligned} \tag{3}$$

*where $PT'_\Pi$ is another planning tree based on the problem $\Pi$.*

In other words, it is impossible to construct a planning tree $PT'_\Pi$ based on $\Pi$ that fulfills the axioms $A_X$, but differs from those generated by $X$. This means that the axiomisation is a perfect representation of the planning algorithm. For a deterministic algorithm, a complete axiomisation is compatible with exactly one planning tree: the one that is generated by the planning algorithm on the public problem. For a randomized algorithm, the axioms are compatible with all planning trees in $\mathbf{PT}_X(\Pi)$.

## Proving Public Self-Sufficiency

In Definition 2 we defined public self-sufficiency for one execution of an algorithm on a specific problem. In order to analyse whether an algorithm is publicly self-sufficent, we have to prove whether there are situations in which the algorithm can lead to a privacy leak.

***Definition 6*** *An algorithm $X$ is* publicly self-sufficient *iff for any problem $\Pi$ and any Execution $E$ of the algorithm on $\Pi$*

$$\mathcal{L}(\mathbf{T}_X^E(\Pi)) \vDash A_X \tag{4}$$

*where $A_X$ is a complete axiomisation of $X$ and $\mathbf{T}_X^E(\Pi)$ the transmitted tree generated during the execution $E$ of the algorithm on problem $\Pi$.*

## Secure-MAFS

In the following, we will apply the discussed framework to an A*-based secure-MAFS algorithm (Brafman 2015). We will first construct an axiomisation of the A* algorithm and show that it is complete. Then, we will show that the algorithm is not publicly self-sufficient.

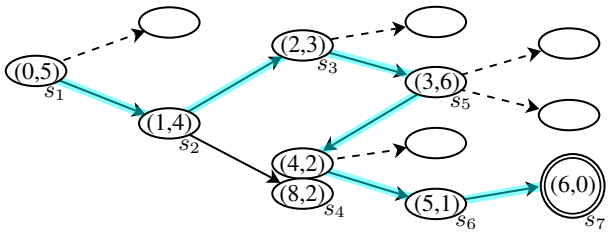


Figure 2: Possible search tree of a MAFS problem. The states are designated by their $(g(s), h(s))$ values. Note that $s_4$ is expanded twice with different paths costs. $s_7$ is a goal state.

### Axiomisation

The A* algorithm doesn't expand the whole search tree. Instead, the algorithm keeps a list which contains all states that are whithin one action from an already expanded state. In every step it always expands the state $s_k$ with the lowest expected path cost $f(s_k)$ from the list. If two states have the same expected path cost, a tie-breaking mechanism is applied. For our analysis we assume that the expected path cost function is chosen in a way that doesn't allow ties. This can be achieved by incorporating the tie-breaking mechanism into an expanded expected path cost function $\hat{f}(\cdot)$.

We introduced a set of descriptors $D$ to model the additional information transmitted during the execution of a MAP algorithm. In the case of secure-MAFS, this would be the codomains $\mathbb{D}_g$ and $\mathbb{D}_h$ of the path cost function and the heuristic function.

$$\begin{aligned} g \ &: \ S \to \mathbb{D}_g \\ h \ &: \ S \to \mathbb{D}_h \\ D \ &:= \mathbb{D}_g \times \mathbb{D}_h \end{aligned} \tag{5}$$

Since secure-MAFS is an informed search algorithm, and since we want to assume full knowledge of the observer, we have to assume that the functions $g(\cdot)$ and $h(\cdot)$ are known to the observer.

An axiomisation $A_{\mathrm{A}^*}$ of the A* algorithm can be defined as follows:

1. *Action definition* An action $a$ can only be applied in a state $s$ if the action's precondition is satisfied in $s$. Similarly, the valuation of the successor state $s'$ must satisfy the postcondition of $a$. All propositions not mentioned in the postconditions must stay unchanged between $s$ and $s'$.

$$a_{s,s'} \to \mathrm{pre}_a(s) \wedge \mathrm{apply}_a(s, s') \tag{6}$$

   The functions $\mathrm{pre}_a(s)$ and $\mathrm{apply}_a(s, s')$ are as defined earlier. This axiom follows directly definition of an action and therefore has to apply to all possible planning trees and planning algorithms.

2. *Reachability* Every state except for the initial state is added to the open_list as successor of a previously expanded state. As a consequence, the first time a state is

expanded, its predecessor state must have a lower index.

$$\forall s_k \text{ with } k > 1 \; \exists a, s_j \text{ with } j < k \; : \; a_{s_j,s_k} \tag{7}$$

Since the initial state is the state with the smallest index, and the only one that doesn't need a predecessor with smaller index, this axiom impies that all states are reachable from the initial state $s_1$.

3. *Singleton States* A state is only reexpanded if a shorter path to it is found. So if two states have increasing expected path costs $f$, they have to differ in at least one proposition.

$$\begin{aligned} &\forall s_j, s_k, j < k \text{ with } f(s_j) < f(s_k) \; : \\ &\quad \exists p : (p_{s_k} \wedge \neg p_{s_j}) \vee (\neg p_{s_k} \wedge p_{s_j}) \end{aligned} \tag{8}$$

4. *No Backward Edges* Before a state $s_k$ is expanded, it has to be added to the open_list. This only happens if an action from a previously expanded state leads to $s_k$. So its predecessor must have been expanded before and thus have a lower index. As we defined above, all nodes in the planning tree are expansions of a state.

$$\forall s_k, s_j, k \leq j \; : \; \neg a_{s_j,s_k} \tag{9}$$

5. *Expansion Order* Before a state $s_k$ is expanded, a number of possible states is considered reachable (in the open_list). They can be seen as the successor states of all currently applicable actions. Out of these states, $s_k$ must have the lowest $f$-value.

$$\begin{aligned} &\forall s_k, s_j, \text{ with } j < k \; : \\ &\forall a \text{ with } \mathrm{pre}_a(s_j) \text{ and } f(\mathrm{apply}_a(s_j)) < f(s_k) \; : \\ &\exists s_l \text{ with } l < k \; : \; a_{s_j,s_l} \end{aligned} \tag{10}$$

The formular reads as follows: all states reachable from $s_j$ whose expected path cost is lower than that of $s_k$ must be expanded before $s_k$.

6. *Shortest Path* If two actions are both applicable in different states, and both would expand a new state with identical valuations, only the one with the lower expected path cost is applied.

$$\begin{aligned} &\forall s_j, s_k, s_l \text{ with } j, k < l \text{ and} \\ &a^A, a^B \text{ with } \mathrm{pre}_{a^A}(s_j) \wedge \mathrm{pre}_{a^B}(s_k) \text{ and} \\ &\mathrm{apply}_{a^A}(s_j) = \mathrm{apply}_{a^B}(s_k) = V(s_l) \text{ and} \\ &f\big(\mathrm{apply}_{a^A}(s_j)\big) < f\big(\mathrm{apply}_{a^B}(s_k)\big) \; : \\ &\neg a^B_{s_k,s_l} \end{aligned} \tag{11}$$

This also implies that each node has only one predecessor with lower index. In combination with axiom 4, which excludes any predecessor with higher index, each node can only have exactly one preceding action.

7. *Path Cost and Heuristic Value* The transmitted path- and heuristic cost $g_s,\ h_s$ that are transmitted in the messages must align with the values computed by the observer.

$$\forall s_k \; : \; h(s_k) = h_{s_k}, \;\; g(s_k) = g_{s_k} \tag{12}$$

While the last axiom may seem obvious, it illustrates an important rule concerning the properties of a privacy preserving algorithm: A MAP algorithm can only preserve privacy if the functions $g(\cdot)$ and $h(\cdot)$ only depend on public variables.

We won't formally prove that the axiomisation is sound according to definition 4. In the following, we will assume that definition 4 holds and based on this assumption prove that $A_{\mathrm{A}^*}$ is a complete axiomisation according to definition 5.

Consider the set $\mathbf{PT}_{\mathrm{ax}}(\Pi)$ of all planning trees based on problem $\Pi$ which satisfy the axioms $A_{\mathrm{A}^*}$. We assume that the axiomisation $A_{\mathrm{A}^*}$ is sound, so $\mathbf{PT}_{\mathrm{ax}}(\Pi)$ must contain $\mathbf{PT}_{\mathrm{A}^*}(\Pi)$. Since secure-MAFS is a deterministic algorithm, the set $\mathbf{PT}_{\mathrm{A}^*}(\Pi)$ contains only one entry $PT_\Pi$.

$$\mathbf{PT}_{\mathrm{A}^*}(\Pi) = \{PT_\Pi\} \subseteq \mathbf{PT}_{\mathrm{ax}}(\Pi)$$

If we can prove that $\mathbf{PT}_{\mathrm{ax}}(\Pi)$ is also singular, so the axiomisation is only consistent with exactly one planning tree $PT_\Pi$ for any problem $\Pi$, then $\mathbf{PT}_{\mathrm{A}^*}(\Pi) = \mathbf{PT}_{\mathrm{ax}}(\Pi)$ and the axiomisation is complete. We will prove this by induction over the nodes of the planning tree.

*Lemma 1: Uniqueness*
*A planning tree $PT_\Pi$ that fulfills the axiomisation $A_{\mathrm{A}^*}$ is uniquely defined.*
*Induction start: $s_1$*
The valuation of the initial state $s_1$ is defined by the problem $\Pi$.

*Induction hypothesis:*
The partial tree $PT_\Pi^{k-1}$ defined by the states $s_1, \ldots, s_{k-1}$ and all applied actions between them is uniquely defined.

*Induction step: $s_k$*
Let $S_k^{\mathrm{pre}} = \{s_1, \ldots, s_{k-1}\}$ be the set of all states in $PT_\Pi^{k-1}$. Per induction hypothesis, all those states and their valuations are uniquely defined.

Axiom 2 defines that $s_k$ must be the successor of another state $s_j$ via an action $a^{s_k}$. Due to axiom 4, the preceding state must have a lower index, so we know that $s_j \in S_k^{\mathrm{pre}}$. Consequently, $s_k$ must be reachable from a state in $S_k^{\mathrm{pre}}$ by application of one action. We call $S_k^{\mathrm{pot}} = \bar{s}_1, \ldots, \bar{s}_n$ the set of potential states reachable from $S_k^{\mathrm{pre}}$ in this manner. We sort the potential states by $f$-value, with $f(\bar{s}_1) < \cdots < f(\bar{s}_n)$. This is possible because we defined $f(\cdot)$ to be an injective function by including the tie-breaking function in it.

Consider a state $\bar{s}_i \in S_k^{\mathrm{pre}}$ as candidate for $s_k$. Axiom 5 demands that all reachable states with lower $f$-value - in this case the states $\bar{s}_1, \ldots, \bar{s}_{i-1}$ - have a lower index. All states with an index lower than $k$ are in the set $S_k^{\mathrm{pre}}$. It follows that $\{\bar{s}_1, \ldots, \bar{s}_{i-1}\} \subseteq S_k^{\mathrm{pre}}$.

All states in $S_k^{\mathrm{pre}}$ are already part of the partial planning tree $PT_\Pi^{k-1}$. Consequently, $s_k$ must be the first entry among $S_k^{\mathrm{pot}}$ (as sorted by $f$-value) that is not already part of $PT_\Pi^{k-1}$. This uniquely defines both $s_k$ and its preceding action $a^{s_k}$.

Axiom 6 implies that each state has only one preceding action. Therefore the new partial tree $PT_\Pi^k$ generated by

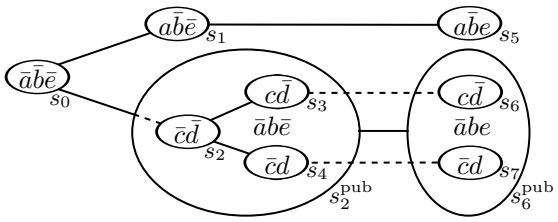


Figure 3: Public and private search tree as processed by secure-MAFS. The propositions $a, b$ and $e$ are public, $c$ and $d$ are private.

adding $s_k$ and $a^{s_k}$ to $PT_{\Pi}^{k-1}$ doesn't allow the addition of any further actions between the states. Therefore, all states and actions of $PT_{\Pi}^{k}$ are uniquely defined. □

It follows from the lemma that $\mathbf{PT}_{\mathrm{A}^*}(\Pi) = \mathbf{PT}_{\mathrm{ax}}(\Pi)$. Therefore, $A_{\mathrm{A}^*}$ is a complete axiomisation of the $A^*$ algorithm.

## Public Self-Sufficiency

The secure-MAFS algorithm only sends valuations of states in the messages, not which actions were used to get to these states. So at the end of the execution, the transmitted graph consists of an unconnected set of states.

Secure-MAFS is an extension of MAFS that was developed to give better privacy guarantees. Instead of just keeping a list of open states it maintains a list of the public projections of these states. Each of these open public states has two lists attached: a list of open private states that have this same public projection and a list of public states $s'$ which are successors of any state in the first list. Figure 3 illustrates this: The private states $s_3, s_4, s_5$ and $s_7, s_8$ are merged to two single public states $t_1$ and $t_2$. As $t_2$ is the successor of $t_1$, the open list of this problem would look as follows:

| public states: | $\bar{a}\bar{b}\bar{e}$ | $a\bar{b}\bar{e}$ | $\bar{a}b\bar{e}$ | $a\bar{b}e$ | $\bar{a}be$ |
|---|---|---|---|---|---|
| private states: | $\bar{c}\bar{d}$ | $\bar{c}\bar{d}$ | $\bar{c}\bar{d}$<br>$c\bar{d}$<br>$\bar{c}d$ | $\bar{c}\bar{d}$ | $c\bar{d}$<br>$\bar{c}d$ |
| successors: | $a\bar{b}\bar{e}$<br>$\bar{a}b\bar{e}$ | $a\bar{b}e$ | $\bar{a}be$ | - | - |

Instead of broadcasting every state to the other agents, the secure-MAFS algorithm adds new private states to the list and only sends a new message when a new public state is added to the open list.

As discussed earlier, an informed search algorithm can only be privacy preserving if the path cost $g(\cdot)$ and the heuristic $h(\cdot)$ only depend on public properties; therefore we will assume this for this analysis.

To prove or disprove strong privacy, we will go through the axioms and show whether they can stand in conflict to the transmitted graph. In order to keep the clarity of the prove, we will only consider the interactions of each axiom with the previous axioms. We call $s_k$ a global state with public and private propositions, $s_k^{\text{priv}}$ a purely private state that is not transmitted during the application of MAFS and $s_k^{\triangleright}$ the public projection of $s_k$. Equivalently, $a^{\triangleright}$ is the public projection of an action $a$ that is transmitted to the other agents.

1. *Action Definition*

$$a^{\triangleright}_{s_j^{\triangleright}, s_k^{\triangleright}} \rightarrow \mathrm{pre}_{a^{\triangleright}}(s^{\triangleright}) \wedge \mathrm{apply}_{a^{\triangleright}}(s_k^{\triangleright}, s_j^{\triangleright})$$

Since the transmitted graph initially contains no actions, this axiom cannot stand in contrast do it.

2. *Reachability*

$$\forall s_k^{\triangleright} \text{ with } k > 1 \; \exists a^{\triangleright}, s_j^{\triangleright} \text{ with } j < k \; : \; a^{\triangleright}_{s_j^{\triangleright}, s_k^{\triangleright}} \qquad (13)$$

Every state except the initial state has to be the successor of another state with lower index. In combination with axiom 1 this means that the public projection $s_k^{\triangleright}$ of a state $s_k$ needs a public predecessor $s_j^{\triangleright}$, $j < k$ and a public action $a^{\triangleright}$ such that $\mathrm{pre}_a^{\triangleright}(s_j^{\triangleright})$ and $\mathrm{apply}_{a^{\triangleright}}(s_j^{\triangleright}, s_k^{\triangleright})$ are true.
The last action that leads to a transmitted state is always a public action. So the public projection of the postconditions of that action are satisfied in $s_k$.

$$\forall s_k \; \exists a \; : \; \mathrm{post}_{a^{\triangleright}}(s_k^{\triangleright})$$

Before every public action can come an arbitrary number of private actions. By definition, a private action only has private variables in its effect. Accordingly, between two transmitted states $s_j$ and $s_k$ are a number of untransmitted private states $(\bar{s}_1^{\text{priv}}, \ldots, \bar{s}_n^{\text{priv}})$ which all have the same public projection as the public predecessor $s_j$. Since the public projection of the public predecessor is the same as the projection of the actual predecessor $s_j^{\triangleright} = \bar{s}_n^{\text{priv}\triangleright}$, and the public projection of the precondition $\mathrm{pre}_a^{\triangleright}$ is applicable in $\bar{s}_n^{\text{priv}}$, it must also be applicable in $s_j$.

$$\forall s_k \; \exists a, s_j \; : \; \mathrm{pre}_{a^{\triangleright}}(s_j^{\triangleright}) \wedge \mathrm{post}_{a^{\triangleright}}(s_k^{\triangleright})$$

Since the only thing that affects the public projection of the states between $s_j$ and $s_k$ is the public action $a$, we can conlude that all public propositions not mentioned in $\mathrm{post}_{a^{\triangleright}}$ stay the same.

$$\forall s_k \; \exists a, s_j \; : \; \mathrm{pre}_{a^{\triangleright}}(s_j^{\triangleright}) \wedge \mathrm{apply}_{a^{\triangleright}}(s_j^{\triangleright}, s_k^{\triangleright})$$

In conclusion, the *single root state* axiom is also satisfied in any execution of secure-MAFS.

3. *Singleton States*

$$\forall s_k, s_j \text{ with } k < j, \; f(s_k) \leq f(s_j) :$$
$$\exists p \in P^{\text{pub}} : (p_{s_k} \wedge \neg p_{s_j}) \vee (\neg p_{s_k} \wedge p_{s_j})$$

Whenever secure-MAFS extends a state, it checks whether that another state with the same public projection was already transmitted. If so, the state is not transmitted to the other agents. This guarantees that no public state is transmitted twice and therefore that this axiom always holds.

4. *No Backward Actions*

$$\forall s_k, s_j, k \leq j \; : \; \neg a_{s_j, s_k} \qquad (14)$$

All actions implied by axiom 2 lead from a previously expanded state to one that is expanded later on. Therefore, this axiom is always satisfied.

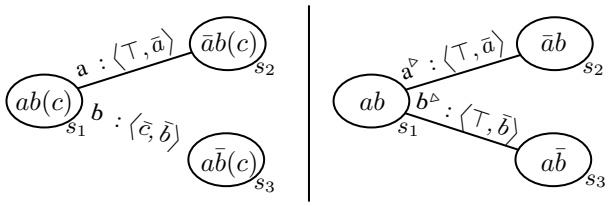


Figure 4: Left: global states; right: public projection of the same states. In the public projection, the state $(a\bar{b})$ appears to be reachable, while in fact it is not.

5. *Expansion Order*

$$\begin{aligned}&\forall \text{ public } s_j, s_k, \text{ with } j < k \ : \\ &\forall a \text{ with } \mathrm{pre}_{a^{\triangleright}}(s_j^{\triangleright}) \text{ and } f(\mathrm{apply}_{a^{\triangleright}}(s_j)) < f(s_k) \ : \\ &\exists \text{ public } s_l \text{ with } l < k \ : \ a_{s_j,s_l}\end{aligned} \tag{15}$$

Consider a situation as shown in figure 4. While $s_3^{\triangleright}$ appears to be reachable in the public projection of the problem, it is in fact not because the private variable $c$ prohibits it.
If $f(s_3) < f(s_2)$, the axiom requires $s_3^{\triangleright}$ to be expanded before $s_2^{\triangleright}$. But since $s_3$ can't be expanded, this situation leads to a violation of axiom 5.

6. *Shortest Path*

$$\begin{aligned}&\forall s_j, s_k, s_l \text{ with } j, k < l \text{ and} \\ &a^A, a^B \text{ with } \mathrm{pre}_{a^{A\triangleright}}(s_j^{\triangleright}) \wedge \mathrm{pre}_{a^{B\triangleright}}(s_k^{\triangleright}) \text{ and} \\ &\mathrm{apply}_{a^{A\triangleright}}(s_j^{\triangleright}) = \mathrm{apply}_{a^{B\triangleright}}(s_k^{\triangleright}) = V(s_l^{\triangleright}) \text{ and} \\ &f\big(\mathrm{apply}_{a^{A\triangleright}}(s_j^{\triangleright})\big) < f\big(\mathrm{apply}_{a^{B\triangleright}}(s_k^{\triangleright})\big) \ : \\ &\neg a^B_{s_k,s_l}\end{aligned} \tag{16}$$

The same reasoning as for axiom 5 applies. If the public projection $a^{A\triangleright}$ is applicable, but the actual action $a^A$ is not, then $a^B_{s_k,s_l}$ is the only path that connects $s_l$ to the rest of the planning tree. Consequently, axiom 6 is not guaranteed to hold.

7. *Path Cost and Heuristic Value*

$$\forall s_k \ : \ h\big(V(s_k)\big) = h_{s_k}, \ \ g\big(V(s_k)\big) = g_{s_k} \tag{17}$$

Since we assumed that the path cost and heuristic value depend only on public propositions, this is by construction always fulfilled.

Axiom 5 and 6 aren't guaranteed to hold. This means that secure-MAFS is not a public self-sufficient algorithm.

## Logistics Example

We will illustrate this with an example from the logistics domain. A truck needs to deliver two packages `a` and `b` from their respective locations `A` and `B` to a third location `X`. The problem has the following public propositions

`A` : truck is at location `A`
`B` : truck is at location `B`
`X` : truck is at location `X`
`a` : package `a` is loaded in the truck

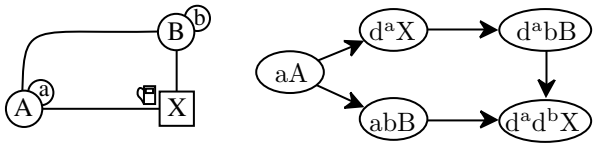


Figure 5: Logistics example. Left: the different locations and paths; right: public search tree.

`b` : packabge `b` is loaded in the truck
$\mathtt{d}^{\mathtt{a}}$ : package `a` is delivered
$\mathtt{d}^{\mathtt{b}}$ : package `b` is delivered

Figure 5 shows the paths between the locations and the corresponding public search tree of the truck. To keep the example readable we omit negative propositions in all state descriptions. The truck has the following public actions:

$$\begin{aligned}&\mathtt{drive}(\mathtt{L}_1,\mathtt{L}_2) &&: \langle \mathtt{L}_1 \mid \bar{\mathtt{L}}_1\mathtt{L}_2\rangle \\ &\mathtt{pick\_up}(\mathtt{L},\mathtt{p}) &&: \langle \mathtt{L}\bar{\mathtt{p}}\bar{\mathtt{d}}^{\mathtt{p}} \mid \mathtt{p}\rangle \\ &\mathtt{deliver}(\mathtt{p}) &&: \langle \mathtt{Xp} \mid \bar{\mathtt{p}}\mathtt{d}^{\mathtt{p}}\rangle\end{aligned}$$

The placeholders $\mathtt{L}_1, \mathtt{L}_2, \mathtt{L} \in \{\mathtt{A}, \mathtt{B}\}$ describe locations and $\mathtt{p} \in \{\mathtt{a}, \mathtt{b}\}$ describes a package. The planning logic $\mathcal{L}_{\Pi^{\triangleright}}$ of the public problem $\Pi^{\triangleright}$ consists of the propositions

$$\begin{aligned}P_{\mathcal{L}_{\Pi^{\triangleright}}} = \{ \\ A_{s_k}, B_{s_k}, X_{s_k}, a_{s_k}, b_{s_k}, d^a_{s_k}, d^b_{s_k}, \\ dr^{L_1L_2}_{s_j s_k}, p^{Lp}_{s_j s_k}, d^p_{s_j s_k}, \\ k, j = 1, \ldots, |S|;\ p = a, b;\ L, L_1, L_2 = A, B, X\}\end{aligned} \tag{18}$$

The public search tree as shown in figure 5 can be transcribed as the following logic formular

$$a_{s_1} A_{s_1} \cdot d^a_{s_2} X_{s_2} \cdot a_{s_3} b_{s_3} B_{s_3} \cdot d^a_{s_4} b_{s_4} B_{s_4} \cdot d^a_{s_5} d^b_{s_5} X_{s_5} \tag{19}$$

again we omit all negative propositions for better readability.

The amount of fuel in the tank of the truck is considered private and has the following possible states:

$\mathtt{t}^{\mathtt{f}}$ : tank is full
$\mathtt{t}^{\mathtt{h}}$ : tank is half full
$\mathtt{t}^{\mathtt{e}}$ : tank is empty

As it happens, driving between locations `A` and `B` uses up a full tank, while between `A` and `X` or `B` and `X` only uses half a tank. The truck can only refuel at location `X`. The private actions of the truck are as follows:

$$\begin{aligned}&\mathtt{drive\_l}(\mathtt{L}_1,\mathtt{L}_2) &&: \langle \mathtt{L}_1\mathtt{t}^{\mathtt{f}} \mid \bar{\mathtt{L}}_1\mathtt{L}_2\mathtt{t}^{\mathtt{e}}\rangle \\ &\mathtt{drive\_s}(\mathtt{L}_1,\mathtt{L}_2) &&: \langle \mathtt{L}_1\bar{\mathtt{t}}^{\mathtt{e}} \mid \bar{\mathtt{L}}_1\mathtt{L}_2(\mathtt{t}^{\mathtt{f}} \triangleright \mathtt{t}^{\mathtt{h}})(\mathtt{t}^{\mathtt{h}} \triangleright \mathtt{t}^{\mathtt{e}})\rangle \\ &\mathtt{refuel} &&: \langle \mathtt{X} \mid \mathtt{t}^{\mathtt{f}}\rangle\end{aligned}$$

The action `drive_l` denotes the long road between `A` and `B`, `drive_s` denotes the short road from or to location `X`.

We apply the secure-MAFS algorithm to this problem. We define the path cost function $g(s)$ as the number of `drive` actions necessary to reach a state. As heuristic function $h(s)$ we use the optimal heuristic $h^*$. The initial state of the truck is ($\mathtt{Aat}_{\mathtt{h}}$): The truck is at location `A` with only package `a` and a half empty tank.

Figure 5 shows two different paths to the goal state. The upper path via state ($\mathtt{d}^{\mathtt{a}}\mathtt{X}$) has a path cost of 3, the lower path

via the state (abB) a cost of 2. Since we assume a perfect heuristic $h^*$, those are also the $f$-values of the mentioned public states. Expanding the cheaper, lower path would require the agent to first expand the action drive_l(A, B). Because the agent has only half a tank of gas, this action is not applicable in the initial state. The agent can only expand the more expensive upper path and transmit the corresponding message ($\mathtt{d}^{\mathtt{a}}\mathtt{X}$). This violates axiom 5, because the state (abB) has a lower $f$-value. This allows the observer to deduce that some private variable must exist that doesn't allow the expansion of the lower path.

# Conclusions

## Optimality vs. Privacy

The problem of secure-MAFS we presented is a general one: If an observer has perfect knowledge of the applied algorithm, it can generate the public search tree of the sender and find the optimal path in it. If an action in that path is not applicable because of a private variable, the transmitted optimal path differs from the expected one. This allows the observer to deduce that private variables exist and affect the execution of the algorithm.

It was shown before that a multi-agent planning algorithm can't simultaneously be strongly private, optimal and efficient (Tožička, Štolba, and Komenda 2017). We can use this knowledge and trade in some optimality or efficiency for improved privacy. A private, but non-optimal variant of secure-MAFS would expand the search tree using a random walk. Instead of expanding the node with the lowest $f$-value, it would choose a random node from the open list. The axiomisation $A_{\mathrm{rMAFS}}$ would consist of axioms 1-4 and 7 from above; the axioms *Expansion Order* and *Shortest Path* would become obsolete. The axioms of rMAFS describe a set of plannig trees, all of which could be generated by the algorithm. The observer can't distinguish whether an action is omitted because it isn't applicable or just because a different action was randomly chosen.

## Comparison to $\beta$-indistinguishability

We will compare our privacy definition to *PST-indistinguishability* (Beimel and Brafman 2018). *PST-indistinguishability* claims that the messages $R$ receives during the application of a MAP algorithm are uniquely defined by the public search tree of the problem and the initial state of $R$.
In a more general setting, *$\beta$-indistinguishability* considers not the public search tree but the output of a leakage function $\beta(\Pi)$. The exact definition of $\beta$-indistinguishability as given in (Beimel and Brafman 2018) is as follows:

***Definition 7*** *Let $\beta : \{0,1\}^* \to \{0,1\}^*$ be a (leakage) function. We say that a deterministic algorithm is $\beta$-indistinguishable if there exists a simulator* Sim *such that for every set $T$ of agents and for every input $x = (x_1, ..., x_n)$ the view of $T$ is the same as the output of the simulator that is given $(x_i)_{i\in T}$ and $\beta(x)$, i.e.,* $\mathrm{Sim}(T, (x_i)_{i\in T}, \beta(x)) = \mathrm{view}_T(x)$.

Instead of working with the public search tree, we define a different construction that we will call the *transmitted graph*. The transmitted graph consists of the states and actions that are transmitted by the algorithm. Since a privacy preserving algorithm only transmits the public projections of actions and states, the transmitted graph consists of a subset of the states and actions in the public search tree.
We call an algorithm *public self-sufficient* if the transmitted graph is consistent with the model a foreign agent has of the algorithm.

If a receiving agent $R$ has perfect knowledge of the sender i.e. if the model $\mathrm{M}_S$ is an exact representation of the applied algorithm including possible heuristics, we can regard the model as a simulator Sim from definition 7. The leakage function $\beta$ is then only given by the transmitted messages. An unsatisfiable model and transmitted graph is then equivalent with the simulator being inconsistent with the transmitted messages.

## Further Work

Further work is needed to evaluate the implications of public self-sufficiency. New algorithms are needed that comply to the stricter rules of public self-sufficiency. One interesting approach might be to check before transmitting a message, whether this message might compromise privacy and to adjust it accordingly or expand a different path. Another promising path is that of randomized algorithms. Especially in a larger problem, they might be vunerable to a stochastic approach. It is also important dicuss further, when we want to trade optimality for privacy, and in what domains more relaxed concepts of privacy are sufficent.

# References


Beimel, A., and Brafman, R. 2018. Privacy preserving multi-agent planning with provable guarantees. *ArXiv*.

Brafman, R., and Domshlak, C. 2008. From one to many: Planning for loosely coupled multi-agent systems. *Amerian Association for Artificial Intelligence*.

Brafman, R. 2015. A privacy preserving algorithm for multi-agent planning and search. *International Joint Conference on Artificial Intelligence*.

Faltings, B.; Léauté, T.; and Petcu, A. 2008. Privacy guarantees through distributed constraint satisfaction. 350–358.

Kautz, H., and Selman, B. 1992. Planning as satisfiability. 359–363.

Maliah, S.; Shani, G.; and Stern, R. 2018. Action dependencies in privacy-preserving multi-agent planning. *Autonomous Agents and Multi-Agent Systems* 32(6):779–821.

Tožička, J.; Štolba, M.; and Komenda, A. 2017. The limits of strong privacy preserving multi-agent planning. In *Twenty-Seventh International Conference on Automated Planning and Scheduling*.