Secure Multi-agent Planning via Sharemind

Radek Bumbálek, Michal Štolba and Antonín Komenda
Department of Computer Science, Czech Technical University in Prague,
Karlovo náměstí 13, 121 35, Prague, Czech Republic
{*bumbarad, michal.stolba, antonin.komenda*}*@fel.cvut.cz*

Keywords: Multi-agent Planning, Secure Multiparty Computation, Privacy-preserving Planning.

Abstract: Classical planning provides models and algorithms for solving problems of finding a sequence of actions that transforms the initial state of the world into a state of the world with the desired properties. In classical planning, we assume that the solution plan entails all actors in the world and thus it can be computed centrally. In multi-agent planning, this assumption is dropped in favor of situations where there is multitude of actors with individual capabilities, goals, and objectives, called agents. In this work, we propose a novel technique for multi-agent planning which combines a state-of-the-art planner called Planning State Machine (PSM) Planner with a framework for mutli-party secure computation, Sharemind. This allows the agents to find a cooperative plan while preventing the leakage of private information in a practical and scalable way.

1 INTRODUCTION

Automated (classical) planning is a long-studied technique of Artificial Intelligence allowing for efficient solution of complex combinatorial problems. The planning problem consists of a logical description of the world, the actions or operators which can be used to modify the world and a set of conditions which are desirable to hold in the goal world state. The task is to find a plan, that is, a sequence of actions, which modifies the initial world state into a world state which fulfills the goal conditions. The most common planning paradigms are heuristic state space search and plan space search. Both techniques are based on intelligent exploration of the combinatorially expanding space of reachable world states (in the first case) and possible partial plans (in the second case).

In modern real-world large-scale personal, corporate or military applications, the whole environment is often not under control of one entity. An complex interplay of more or less cooperative entities is often present. Such entities may need to cooperate in the plan synthesis, while still wanting to protect the privacy of their input data and internal processes. Multi-agent and privacy-preserving multi-agent planning allow the definition of factors of the global planning problem private to the respective entities (i.e., agents) in order to improve the efficiency of planning and/or to maintain the privacy of the information. Similarly to classical planning, the most prevalent planning paradigms in multi-agent planning are multi-agent (distributed) variants of state-space and plans-space search.

In (Tožička et al., 2017), the authors have shown that it is not possible for a multi-agent planner based on either of the planning paradigms mentioned above to be strong privacy-preserving. A strong privacy-preserving planner is a multi-agent planner such that after the execution of the planning process, none of the agents learn any private information they could not have learned from the planning problem input and the public part of the solution plan. Nevertheless, the authors have proposed a theoretical strong-privacy preserving planner (albeit not complete) based on an existing multi-agent planner, the Planning State Machine (PSM) Planner.

In this work, we utilize a Secure Mutliparty Computation framework Sharemind (Bogdanov et al., 2008; Bogdanov, 2013) to implement the most crucial part of the PSM planner and thus draw nearer to a provably and truly strong privacy-preserving planner. We leave less important aspects of the planner for future work, thus we cannot claim full strong privacy preservation yet. We have thoroughly evaluated the new PSM planner variant and we have shown that it is still competitive in terms of planning performance while bringing the aspect of secure computation.

2 MULTI-AGENT PLANNING

In this section, we present the formalism used throughout the paper. First of all, we define a general (that is single-agent) planning task in the form of Multi-Valued Planning Task (MPT). The MPT is a tuple

$$\Pi = \langle \mathcal{V}, O, s_I, s_\star \rangle$$

where $\mathcal{V}$ is a finite set of finite-domain variables, O is a finite set of operators, s_I is the initial state, and $s_\star$ is the goal condition. An *operator* o from the finite set O has a precondition $\mathsf{pre}(o)$ and effect $\mathsf{eff}(o)$ which are both partial variable assignments. An operator o is applicable in a state s if $\mathsf{pre}(o)$ is consistent with s. Application of operator o in a state s results in a state s' such that all variables in $\mathsf{eff}(o)$ are assigned to the values in $\mathsf{eff}(o)$ and all other variables retain the values from s, formally $s' = o \circ s$. A solution to MPT Π is a sequence $\pi = (o_1, ..., o_k)$ of operators from O (a plan), such that o_1 is applicable in $s_I = s_0$, for each $1 \leq l \leq k$, o_l is applicable in s_{l-1} and $s_l = o_l \circ s_{l-1}$ and s_k is a goal state (i.e., $s_\star$ is consistent with s_k).

Similarly as MA-STRIPS (Brafman and Domshlak, 2008) is an extension of STRIPS (Fikes and Nilsson, 1971) towards privacy and multi-agent planning, MA-MPT is a multi-agent extension of the Multi-Valued Planning Task. For n agents, the MA-MPT problem $\mathcal{M} = \{\Pi^i\}_{i=1}^n$ consists of a set of n MPTs. Each MPT for an agent $\alpha_i \in \mathcal{A}$ is a tuple

$$\Pi^i = \left\langle \mathcal{V}^i = \mathcal{V}^{\mathsf{pub}} \cup \mathcal{V}^{\mathsf{priv}_i}, O^i = O^{\mathsf{pub}_i} \cup O^{\mathsf{priv}_i}, s_I^{\triangleright i}, s_\star^{\triangleright i} \right\rangle$$

where $\mathcal{V}^{\mathsf{priv}_i}$ is a set of private variables, $\mathcal{V}^{\mathsf{pub}}$ is a set of public variables shared among all agents $\mathcal{V}^{\mathsf{pub}} \cap \mathcal{V}^{\mathsf{priv}_i} = \emptyset$ and for each $i \neq j$, $\mathcal{V}^{\mathsf{priv}_i} \cap \mathcal{V}^{\mathsf{priv}_j} = \emptyset$ and $O^i \cap O^j = \emptyset$. All variables in $\mathcal{V}^{\mathsf{pub}}$ and all values in their respective domain are public, that is known to all agents. All variables in $\mathcal{V}^{\mathsf{priv}_i}$ and all values in their respective domains are private to agent α_i which is the only agent aware of such V and allowed to modify its value.

The set O^i of operators of agent α_i consists of private and public operators such that $O^{\mathsf{pub}_i} \cap O^{\mathsf{priv}_i} = \emptyset$. The precondition $\mathsf{pre}(o)$ and effect $\mathsf{eff}(o)$ of private operators $o \in O^{\mathsf{priv}_i}$, are partial assignments over $\mathcal{V}^{\mathsf{priv}_i}$, whereas in the case of public operators $o \in O^{\mathsf{pub}_i}$ the assignment is over $\mathcal{V}^i$ and either $\mathsf{pre}(o)$ or $\mathsf{eff}(o)$ assigns a value to at least one public variable from $\mathcal{V}^{\mathsf{pub}}$. Because $\mathcal{V}^{\mathsf{pub}}$ is shared, public operators can influence (or be influenced by) other agents.

The global problem is the actual problem the agents are solving. We define a *global problem* (MPT) as a union of the agent problems, that is

$$\Pi^{\mathsf{G}} = \left\langle \bigcup_{i \in 1..n} \mathcal{V}^i, \bigcup_{i \in 1..n} O^i, s_I, s_\star \right\rangle$$

An *i-projected problem* is a complete view of agent α_i on the global problem Π^{G}. The i-projected problem of agent α_i contains i-projections of all operators of all agents. Formally, an i-projection $o^{\triangleright i}$ of $o \in O^i$ is o. For a public operator $o' \in O^{\mathsf{pub}_j}$ of some agent α_j s.t. $j \neq i$, an i-projected operator $o'^{\triangleright i}$ is o' with precondition and effect restricted to the variables of $\mathcal{V}^i$, that is $\mathsf{pre}(o'^{\triangleright i})$ is a partial variable assignment over $\mathcal{V}^i$ consistent with $\mathsf{pre}(o')$ ($\mathsf{eff}(o')$ treated analogously). The set of i-projected operators is $O^{\triangleright i}$ and an i-projected *problem* is $\Pi^{\triangleright i}$. The set of all i-projected problems is then $\mathcal{M}^{\triangleright} = \{\Pi^{\triangleright i}\}_{i=1}^n$.

A public plan $\pi_j^{\triangleright}$ is i-extensible, if by adding private operators from O_i^{priv} to $\pi_j^{\triangleright}$ and replacing all projections $o^{\triangleright}$ s.t. $o \in O_i^{\mathsf{pub}}$ by o, we obtain a local solution to Π_i. According to (Tožička et al., 2016), a public plan which is i-extensible by all $i \in 1, ..., n$ is a global solution to $\mathcal{M}$ and can be extended by all agents to form $\{\pi_i\}_{i=1}^n$.

3 PRIVACY-PRESERVING MULTI-AGENT PLANNING

The Privacy-Preserving Multi-Agent Planning (PP-MAP) is based on concepts from the field of Secure Multiparty Computation and their application to MAP, as described in the following sections.

3.1 Secure-multiparty Computation

Secure multiparty computation (MPC) (Yao, 1986) is a subfield of cryptography, which studies computing a function f by a set of n parties $p_1, \ldots, p_n$ such that each p_i knows part of the input of f. The goal of MPC is to compute f in a way that no party p_i learns more information about the inputs of other parties than what can be learned from the output of f. Clearly, PP-MAP is an instance of MPC, where the respective problems of the agents are the inputs and the global plan is the desired output.

In an ideal world, assuming secure communication channels, a trusted third-party could receive the inputs from the parties, perform the needed computation, and return the solutions to the respective parties. Secure MPC studies whether and how such computations can be done in the real world without the trusted third-party, and alternatively how much private information leaks in comparison to the ideal execution (recently applied to PP-MAP in (Štolba et al., 2019)). In some cases, the trusted third-party can be replaced by a relatively small number (e.g., three) computation agents which can be trusted not to collude as each of them is

controlled by a different party. Such approach is taken by some real-world secure MPC solutions, such as Sharemind (Bogdanov et al., 2008) used in this work.

In MPC, assumptions are typically placed on the participating parties (agents in our case) and their communication and computation capabilities. The assumptions we consider in ths work are the following: (i) There is no trusted third-party. (ii) The planning agents are semi-honest (or honest but curious). (iii) The computation power of the agents is either unbounded (information-theoretic security), or polynomial-time bounded (computational security).

The assumption of semi-honest agents means, as opposed to malicious agents, that every agent follows the rules of the computation protocol based on its input data, but after the computation is finished, it is allowed to use any information it has received during the protocol to compromise the privacy. The computation power of the agents (which can be used to infer additional knowledge from the executed protocol) is typically seen either as unbounded, in which case we are talking about information-theoretic security, or polynomial-time bounded, which is the case of computational security.

When applied to PP-MAP, the notion of polynomial-time bounded adversary may seem somewhat less suitable, as the planning itself is not polynomial (but PSPACE-complete (Bylander, 1994)). Nevertheless, the computation power of the agents is still polynomial, thus allowing to solve either polynomial instances or small instances. For such instances of planning problems which can be practically solved, the cryptographic assumptions (such as that the factoring of large integers is hard), for which the polynomial-time bound is typically used are still valid.

There are basically two approaches to multi-agent planning based on the MPC techniques. The first approach is to encode planning in some general MPC technique such as cryptographic circuits (Yao, 1986). For example, the cryptographic circuits encode the whole computation of a function into a boolean or algebraic circuit, which can be then securely evaluated using some of the existing secure protocols. The problem related to MAP is, that the worst-case scenario has to be encoded, that is, the complete exploration of the search space, which itself is exponential in the size of the MAP input (e.g., MA-STRIPS). Therefore, it is not clear how exactly PP-MAP would be encoded in such general model, whether it is even feasible, and what the overhead of such encoding would be.

The second approach is to devise a specific PP-MAP algorithm based on MPC primitives, such as private set intersection (Li and Wu, 2007). There is a number of solutions for a related problem, shortest path in a graph (Brickell and Shmatikov, 2005). Such techniques solve the shortest path problem for an explicit graph, typically represented by a matrix. In classical planning and subsequently in MAP, the explicit graph (that is, the transition system) is exponential in the problem size, which, for practical problem sizes, makes it impossible to use such explicit (e.g., matrix) representation. So far, the only published (theoretical) MAP planner using MPC primitives is (Tožička et al., 2017) which uses the secure set intersection as part of the planning process. In this work we implement the theoretical concept by implementing the set intersection using the Sharemind framework.

3.2 Weak and Strong Privacy

In the MAP and PP-MAP literature, the concept of privacy has been mostly reduced to the idea of weak privacy, as stated in (Nissim and Brafman, 2014). Here, we rephrase the (informal) definition:

> We say that an algorithm is *weak privacy-preserving* if, during the whole run of the algorithm, the agent does not communicate (unencrypted) private parts of the states, private actions and private parts of the public actions. In other words, the agent openly communicates only the information in $\mathcal{M}^{\triangleright} = \{\Pi^{\triangleright i}\}_{i=1}^{n}$.

Obviously, the weak privacy does not give any guarantees on privacy whatsoever, as the adversary may deduce private knowledge from the communicated public information. Nevertheless, not all weak privacy-preserving algorithms are equal in the amount of privacy leaked as shown in (Štolba et al., 2019).

In (Nissim and Brafman, 2014), the authors define also strong privacy, which is in accordance with the cryptographic and secure MPC model. Here we informally rephrase the definition of Nissim&Brafman:

> A *strong privacy-preserving* algorithm is such a distributed algorithm that no agent α_i can deduce an isomorphic (that is differing only in renaming) model of a private variable, a private operator and its cost, or private precondition and effect of a public operator belonging to some other agent α_j, beyond what can be deduced from the public input ($\mathcal{M}^{\triangleright} = \{\Pi^{\triangleright i}\}_{i=1}^{n}$) and the public output (projection of the solution plan $\pi^{\triangleright}$).

A more precise formal definition can be stated based on the definition of privacy in secure MPC, where privacy is typically defined with respect to the ideal world in which a trusted third party exists.

Definition 1. (Strong privacy MPC) Let $p_1, ..., p_n$ be n parties computing an algorithm P which takes n pri-

vate inputs $\mathsf{in}_1, ..., \mathsf{in}_n$ respective to the parties, and produces n private outputs $\mathsf{out}_1, ..., \mathsf{out}_n$ respective to the parties. Let T be a trusted third-party. An algorithm P is *strong privacy-preserving* if the parties $p_1, ..., p_n$ do not learn more information from executing P without the third-party (in a distributed way), than by sending their respective private inputs in_i via a secure channel to T and receiving their respective outputs out_i via a secure channel.

Formally, the definition is bound to the existence of a simulator which, based solely on the input and output of a party i, can simulate the execution of the protocol so that the simulated execution (its distribution, in the probabilistic case) is indistinguishable from the execution of the protocol by the party i. See (Goldreich, 2009) for detailed formal definitions. Now we rephrase the definition in the context of MAP, where the difference is that there is a public part of the input and also a public part of the output.

Definition 2. (Strong privacy MAP) Let $\mathcal{M} = \{\Pi_i\}_{i=1}^n$ be a MAP problem for a set $\mathcal{A}$ of n agents. Let T be a trusted third-party. A MAP planner P is strong privacy preserving, if the agents $\alpha_1, ..., \alpha_n$ do not learn more information from solving $\mathcal{M}$ with P without the third-party (in a distributed way), than by sending their respective agent planning problems Π_i via a secure channel to T and receiving the i-projections $\pi^{\triangleright i}$ of the global plan via a secure channel.

The Definition 2 is compatible with the definition of Nissim&Brafman published in (Nissim and Brafman, 2014; Brafman, 2015).

4 ONE-SHOT SECURE PSM PLANNER

In this section, we describe the main contributions of this work. We start by describing the PSM planner, its variants, and the necessary modifications we made. Then, we describe the implementation of set intersection in Sharemind and its use in the novel One-Shot Secure PSM Planner.

4.1 PSM Planner

The PSM was originally introduced in (Tožička et al., 2016) together with the Planning State Machine (PSM) data structure it is based on. The PSM structure is used to compactly represent a (possibly infinite) set of plans. In this work we somewhat simplify the planner to use only sets of plans and set intersection. The main idea of the PSM Planner is the following generate and test planning paradigm:

1. For each agent, generate a set S_i of solutions of Π_i consisting of valid plans for agent i.
2. For each agent, construct a set $S_i^{\triangleright} = \{\pi^{\triangleright} | \pi \in S\}$ of projected plans.
3. Construct an intersection $S^{\triangleright} = \bigcap_{i \in \mathcal{A}} S_i^{\triangleright}$ of all projected plans.
4. Based on (Tožička et al., 2016) any plan $\pi^{\triangleright} \in S^{\triangleright}$ is publicly extensible and thus is a solution to the global planning problem $\mathcal{M}$.

Based on instantiations of the above steps, multiple variants of the planner can be constructed as proposed in (Tožička et al., 2017):

One-Shot-PSM Planner: generates a proper random subset of all solutions in Step 1. and terminates in Step 3, even if a solution is not found (incomplete).

Iterative-PSM Planner: repeats steps 1.-3. until the intersection $S^{\triangleright} = \bigcap_{i \in \mathcal{A}} S_i^{\triangleright}$ is nonempty, or all agents have generated all possible solutions, in which case if the intersection is empty, there is no global solution. In each iteration of Step 1., new plans are added systematically (e.g., ordered by length).

Full-PSM Planner: each agent first generates all possible solutions in Step 1 (might not terminate due to infinite number of possible solutions). If the problem has a global solution, all solutions are found in the first iteration of Step 4.

By using a secure set intersection (e.g., PSI (Li and Wu, 2007)), both One-Shot-PSM and Full-PSM can be made strong privacy preserving in the information-theoretic sense (without any assumptions), although Full-PSM would not terminate in the presence of infinite number of plans (i.e., including loops). Another promising feature of the One-Shot-PSM planner is that there is a trade-off between completeness and efficiency, which can be exploited. The more plans are generated, the more time it takes, but also the higher is the chance of success in the one shot secure set intersection.

The original Iterative-PSM planner starts with the preparation of a problem and a domain. The preprocessing phase does not only prepare data for the planner but also contains a distributed relaxed reachability analysis algorithm (Štolba and Komenda, 2013) and sets initial landmarks. Landmarks are used to guide the plan generation process towards the solution as the planner is trying to primarily use landmark actions in the generated solutions. Preprocessing does not share private actions and facts directly. However, there is a possibility of privacy information leakage because of public fact sharing during relaxed planning. Determining the amount of information leaked is beyond

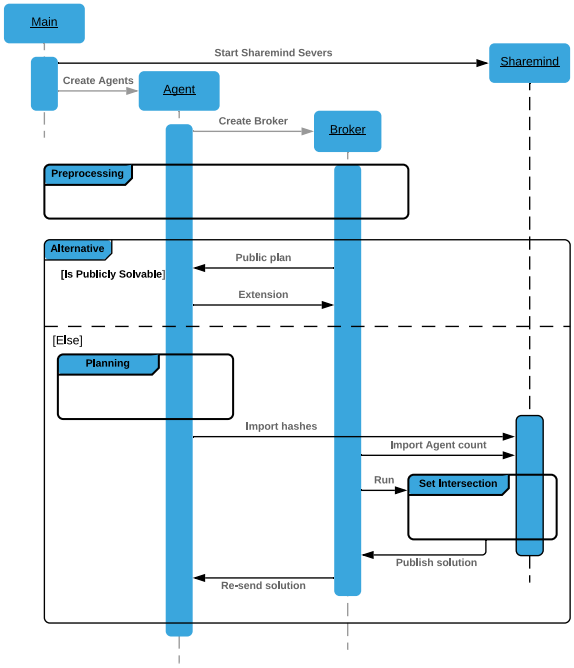

Figure 1: One-shot PSM Planner UML diagram.

this work, see e.g. (Štolba et al., 2019). We leave the relaxed reachability in place for this work and postpone its replacement or secure re-implementation for future work. This means that the presented planner is not fully strong privacy-preserving, but nevertheless serves as a valuable stepping stone towards this ultimate goal.

4.2 One-Shot PSM Planner

Let us now describe the modifications necessary to implement the One-Shot PSM Planner which is then combined with the Sharemind Secure MPC framework to obtain the Secure One-Shot PSM Planner. As already mentioned before, the initial preprocessing remains unchanged in the current version of the planner (see Figure 1). Because the preprocessing is not changed, in some cases planner solves the problem even without even calling the One-Shot algorithm. After grounding One-Shot agents call Fast-downward in a loop. Each call returns several plans which are saved for future use. When the time limit expires, all Fast-downward threads are killed. Agents then make public projections of plans they have found. Different plans may share the same public projection. These duplicates are removed, keeping the plan with the shortest private part.

To justify the next step, we first need to explain how the Sharemind framework processes input. Sharemind itself does not have a simple input, the language secreC 2 does not allow scripts to be run with additional arguments. Sharemind was implemented for operations over databases, therefore we need to convert the generated plans into a database. The iterative algorithm makes use of knowledge of how similar plans are. Based on similarity and usage of public actions, landmarks are set. However, in the One-Shot algorithm, only one comparison is made. Any additional information could not be used anyway. Therefore we only need a simple comparison, whether the plans are the same, or not. Therefore we can use encryption over the whole plans. We decided for hashing our public projections with SHA256. Because public plans are strings of variable lengths usually much longer than is a hash size, it is possible that different plans would be encrypted with the same hash. However, it is extremely improbable. In test situations we usually have 2-8 agents, each generating roughly 30-100 original public projections. Counting the probability leads to the so-called birthday problem. Rough estimation via Taylors polynomial leads to $p(n,d) \approx 1 - e^{\frac{-n^2}{2d}}$. Where d is the size of hash $d = 2256$ and n is the number of plans. Even if we use a ridiculously huge estimation such as $n = 10^{10}$ we still obtain the probability of collision $1 - e^{\frac{-10^{20}}{2257}} = 4.31 \times 10^{-68}$. Hashed plans are then saved into the table database as sets of four 64bit Longs. Hashes are much easier to compare than sets of strings of variable length. Also hashing increase security in case of eavesdropping on the input or the output of the Sharemind script. The table database is created via Sharemind CSV importer. Each agent creates a CSV file containing hashes and XML file with a description of tables to be imported. Then they call the import command. The broker also creates CSV and XML files with simple information about the number of agents for the Sharemind script. Name conventions are based on the numbering of agents, therefore knowledge of their count is crucial. After all the agents imported their hashed plans, the broker calls set intersection script. The script returns a hash of a common plan if there is at least one. The broker then resends it back to agents so every agent could reconstruct the original plan based on the hash and create an output.

4.3 Plan Intersection via Sharemind

Sharemind is operated via built-in commands and created scripts. These scripts as already mentioned are written in the secreC 2 language. This language is rather limited and resembles the C language. The specific part of this language is privacy types, which are annotated with a privacy domain. Private variables are being treated in a very specific way. Each variable is divided into three parts, one for each of the Sharemind

servers. Every operation over these variables is done in means of multi-party computation. Declassification can be done only by a declassify expression.

We compute a set intersection by a simple comparison each element of a set A with each element of a set B. The set created by comparison of two previous sets is then used for comparing with the next set until the complete intersection is made. Function setIntersection makes an intersection of given arguments, by comparing each member of the first argument, with each member of the second argument. Because of the encryption, we can not directly declare whether two numbers are equal. However, we can apply basic arithmetic operations such as subtraction and retype the variable from a number to a boolean. By a subtraction and retyping it is guaranteed that the declassification of the variable will not expose original values. The output of the script is the first hashed plan in the resultant set or an empty set in case the algorithm will end with an empty intersection. The output is read by the broker and resent to agents. In case of an empty intersection, the broker terminates the program.

5 EVALUATION

The evaluation consists of two main parts, testing of the planner itself and testing of the sharemind set intersection algorithm. Because of the character of the One-Shot algorithm, we can not easily compare its planning speed, because it will always use the maximum time given for planning. However, we can observe efficiency with changing time. Also, we want to measure a time consumption of the sharemind set intersection script. We need to account this time when we start our planner in order to assure that the algorithm will finish before the time limit.

5.1 Planner Evaluation

For the planner evaluation we use the benchmarks and restrictions of the CoDMAP Competition (Komenda et al., 2016). Our main concern is the number of problems solved (coverage) in the given time limit, which is 30 minutes. The Sharemind framework supposed to run on three independent servers to assure privacy preservation. In our testing scenarios, we emulate such environment in the virtualization tool VirtualBox. VirtualBox allows us to strictly set properties of the virtual machine. We used settings from CoDMAP, which allows 4 thread processor and 8GB of RAM.

Table 2, column "Pre." shows the domains where a public plan solution was found by the reachability analysis and the One-Shot algorithm itself was not started. The One-Shot algorithm was applied in the following domains: Elevators, Rovers, Satellites, and Zenotravel (where it does not solve any problems). We tested the algorithm with three settings, giving the planning part 100, 300 and 1740 seconds. We assume one-minute time reserve for the preprocessing and the set intersection. Even though the planner did not solve even half of the problems, the efficiency has to be put in the context by comparing it with the original planner. Virtualization and running all agents on a single device may lower efficiency significantly when compared with the results of the CoDMAP competition. Therefore we tested the iterative algorithm as well, the comparison is shown in Table 2. We can see that in some domains (Elevators, Zenotravel) our solution offers unconvincing results. However, domains such as Rovers and Satellites were solved successfully. In some domains, the planner succeeded even with a small amount of planning time, but efficiency boost with a long time is rather limited. This is caused by the rapidly growing state space during a search.

Table 1: Problems solved by One-shot algorithm.

	Elevators			Rovers			Satellites		
	100	300	1740	100	300	1740	100	300	1740
1	ok	ok	ok	ok	ok	ok	ok	ok	ok
2	x	x	x	ok	ok	ok	ok	ok	ok
3	x	x	x	ok	ok	ok	ok	ok	ok
4	x	x	x	ok	ok	ok	ok	ok	ok
5	x	x	x	ok	ok	ok	ok	ok	ok
6	x	x	x	ok	ok	ok	ok	ok	ok
7	x	x	x	ok	ok	ok	ok	ok	ok
8	x	x	x	ok	ok	ok	ok	ok	ok
9	x	x	x	x	ok	ok	x	x	x
10	x	x	x	x	x	ok	ok	ok	ok
11	x	x	x	x	x	x	ok	ok	ok
12	x	x	x	x	ok	ok	x	x	x
13	x	x	x	x	x	x	ok	ok	ok
14	x	x	x	x	x	x	x	x	x
15	x	x	x	x	x	x	x	x	x
16	x	x	x	x	x	ok	x	x	x
17	x	x	x	x	x	x	x	x	x
18	x	x	x	x	x	x	x	x	x
19	x	x	x	x	x	x	x	x	x
20	x	x	x	x	x	x	x	x	x

5.2 Evaluation of the Set Intersection Algorithm

We generated sets of hashes to simulate problems that the Sharemind script will be solving. Because of the character of used data (hashes), artificially manufactured sets are indistinguishable from the original.

Table 2: Comparison of One-Shot and iterative PSM.

Domain	Pre.	Iterative	One Shot	Total
Blocksworld	20	20	20	20
Depot	17	17	17	20
Driverlog	20	20	20	20
Elevators	-	12	1	20
Logistics	20	20	20	20
Rovers	-	14	12	20
Satellites	-	11	11	20
Sokoban	17	17	17	20
Taxi	20	20	20	20
Woodworking	20	20	20	20
Zenotravel	-	8	0	20
Σ		179 (81.4%)	158 (71.8%)	220 (100%)

Therefore we can faithfully imitate real problems in a controlled environment. We can control nature of the problem by 3 main attributes: (i) Sizes of sets, (ii) Number of sets, and (iii) Similarity rate. The similarity rate defines a percentage loss of plans in each intersection iteration. For example, with 3 sets of size 100 and the similarity rate 50%, each iteration eliminates 50% of plans. After 1 iteration (comparing the first 2 sets) only 50 plans remain. After the second iteration (the result of the first iteration intersected with the third set) only 25 plans remain. We assume 5 sets, because of the average number of agents. We also assume each set holding 200 members. Higher set sizes allow us to better observe similarity rate impact, because with a lower number of members in each set, low similarity rate would cause intersection to contain only one member in a small number of iterations.

We can see in Figure 2 that the growth is unstable. This is caused by the random shuffling of set members. In some occasions, sets intersect very fast with a comparison of the only the first couple of members. The role of chance is more noticeable when a difference between member count of sets is higher. This difference lowers with higher similarity rate. For example, with similarity rate 1.0, member count of each set 75, we com- pare 5 identical sets in 4 iterations. When all set members are present in both sets, we always make 75 comparisons per iteration, 1560 comparisons in Sharemind experiments total, with no difference between shuffled and unshuffled sets. But with a low similarity rate such as 0.2, we compare 75 with 75 members only in the first iteration. In the second iteration, we will compare only 15 members with 75 and in the third only 3 members of the intersection with 75 members of the fourth set. Based on shuffle, we can make between 6 (1 + 2 + 3) and 222 (73 + 74 + 75) comparisons.

Next, we examined time complexity grow with an

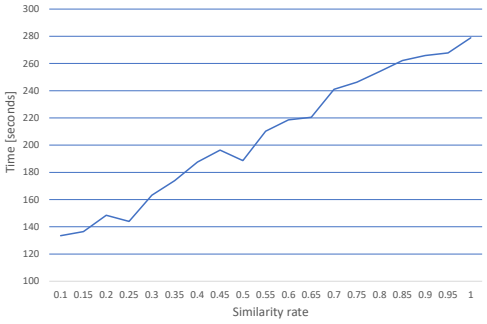

Figure 2: Sharemind script time complexity based on similarity of sets.

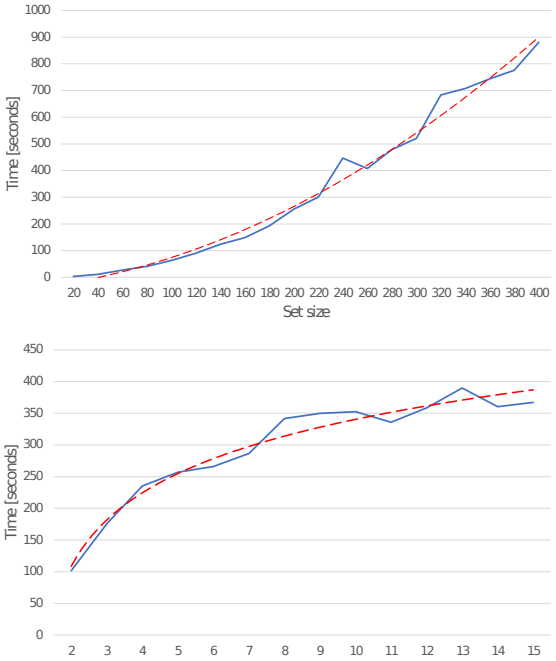

Figure 3: Sharemind script time complexity based on set size (top blue) and number of sets (bottom blue), including polynomial and logarithmic approximation respectively (red).

increase of set sizes. We were always comparing 5 sets and a similarity rate of 75%. We use a higher similarity rate in order to lower the impact of the random plan shuffle. The intersection of n sets with m members is done by comparing their m x m members n times which leads to O(m 2), as observed from the experiment in Figure 3 (top). Lastly, we observe time complexity with the growing number of sets. We used to set size 200 and 75% similarity rate. As we can see, even with a high amount of agents, time complexity grows slowly, because intersection will get very small after a couple of iterations as seen in Figure 3 (bottom).

6 CONCLUSION AND FUTURE WORK

In this work, we have pioneered the use of Secure Multiparty Computation primitives and tools represented by the Sharemind framework on the problem of Privacy-Preserving Multi-Agent Planning. We have implemented concepts so far described only in theory and shown that they are competitive in terms of planning performance and valuable in therms of privacy preservation. We have not succeeded in implementing a full general strong privacy-preserving planner, but the presented work is a crucial stepping stone towards such ultimate goal. To reach that goal, the main future work lies in either eliminating or, better, replacing the relaxed reachability analysis step with its secure variant. A future work in terms of improving the planner's performance is to use the full PSM structures in the planner and implement their intersection inside the Sharemind framework.

ACKNOWLEDGEMENTS

This research was supported by the Czech Science Foundation (grant no. 18-24965Y).

REFERENCES

Bogdanov, D. (2013). *Sharemind: programmable secure computations with practical applications*. PhD thesis.

Bogdanov, D., Laur, S., and Willemson, J. (2008). Sharemind: A framework for fast privacy-preserving computations. In *Proceedings of the European Symposium on Research in Computer Security*, pages 192–206. Springer.

Brafman, R. I. (2015). A privacy preserving algorithm for multi-agent planning and search. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, (IJCAI'15)*, pages 1530–1536.

Brafman, R. I. and Domshlak, C. (2008). From one to many: Planning for loosely coupled multi-agent systems. In *Proceedings of the 18th International Conference on Automated Planning and Scheduling (ICAPS'08)*, pages 28–35.

Brickell, J. and Shmatikov, V. (2005). Privacy-preserving graph algorithms in the semi-honest model. In *Proceedings of the International Conference on the Theory and Application of Cryptology and Information Security*, pages 236–252. Springer.

Bylander, T. (1994). The computational complexity of propositional STRIPS planning. *Artificial Intelligence*, 69(1-2):165–204.

Fikes, R. and Nilsson, N. (1971). STRIPS: A new approach to the application of theorem proving to problem solving. In *Proceedings of the 2nd International Joint Conference on Artificial Intelligence (IJCAI'71)*, pages 608–620.

Goldreich, O. (2009). *Foundations of cryptography: volume 2, basic applications*. Cambridge university press.

Komenda, A., Stolba, M., and Kovacs, D. L. (2016). The international competition of distributed and multiagent planners (CoDMAP). *AI Magazine*, 37(3):109–115.

Li, R. and Wu, C. (2007). An unconditionally secure protocol for multi-party set intersection. In *Proceedings of Applied Cryptography and Network Security*, pages 226–236. Springer.

Nissim, R. and Brafman, R. I. (2014). Distributed heuristic forward search for multi-agent planning. *Journal of Artificial Intelligence Research*, 51:293–332.

Štolba, M., Fišer, D., and Komenda, A. (2019). Privacy leakage of search-based multi-agent planning algorithms. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 29, pages 482–490.

Štolba, M. and Komenda, A. (2013). Fast-forward heuristic for multiagent planning. In *Proceedings of the 1st ICAPS Workshop on Distributed and Multi-Agent Planning (DMAP'13)*, pages 75–83.

Tožička, J., Jakubuv, J., Komenda, A., and Pechouček, M. (2016). Privacy-concerned multiagent planning. *Knowledge Information Systems*, 48(3):581–618.

Tožička, J., Štolba, M., and Komenda, A. (2017). The limits of strong privacy preserving multi-agent planning. In *Proceedings of the 27th International Conference on Automated Planning and Scheduling (ICAPS'17)*.

Yao, A. (1986). How to generate and exchange secrets. In *Proceedings of the Foundations of Computer Science, 1986., 27th Annual Symposium on*, pages 162–167. IEEE.