Privacy-Preserving Policy Iteration for Decentralized POMDPs

Feng Wu[†] **Shlomo Zilberstein**[‡] **Xiaoping Chen**[†]

[†]School of Computer Science and Technology, University of Science and Technology of China, CHN
[‡]College of Information and Computer Sciences, University of Massachusetts Amherst, USA
wufeng02@ustc.edu.cn, shlomo@cs.umass.edu, xpchen@ustc.edu.cn

Abstract

We propose the first privacy-preserving approach to address the privacy issues that arise in multi-agent planning problems modeled as a Dec-POMDP. Our solution is a distributed message-passing algorithm based on trials, where the agents' policies are optimized using the cross-entropy method. In our algorithm, the agents' private information is protected using a public-key homomorphic cryptosystem. We prove the correctness of our algorithm and analyze its complexity in terms of message passing and encryption/decryption operations. Furthermore, we analyze several privacy aspects of our algorithm and show that it can preserve the agent privacy of non-neighbors, model privacy, and decision privacy. Our experimental results on several common Dec-POMDP benchmark problems confirm the effectiveness of our approach.

Introduction

Many real-world applications require a group of agents to work together towards a common goal, which may be impossible or complicated for a single agent to achieve. In critical domains such as health care and e-commerce, privacy is a major concern because the agents may carry sensitive and private data that should not be accessed without authorization. Privacy is also a crucial issue if the agents in a team are from different parties with different interests. For example, the network infrastructure of a communication network may belong to multiple providers; sensor stations of a sensor network may be owned by several organizations; robots in disaster response may be designed by different companies and operated by different rescue teams (Wu et al. 2015; Ramchurn et al. 2016). They may have constraints on sharing the information about agent models, decision rules, sensor data, etc. In such cases, agents from different parties must collaborate with each other while preserving privacy.

We assume in this work that while agents may have privacy concerns, they otherwise operate cooperatively on the task at hand. The *Decentralized Partially Observable Markov Decision Process* (Dec-POMDP) (Bernstein et al. 2002) offers a rich framework for planning under uncertainty in such settings. The model extends the single-agent POMDP model to cooperative multi-agent settings. Over the past decade, numerous optimal and approximate Dec-POMDP algorithms have been developed (Amato, Dibangoye, and Zilberstein 2009; Wu, Zilberstein, and Chen 2010b; Pajarinen and Peltonen 2011; Oliehoek et al. 2013; Kumar, Zilberstein, and Toussaint 2015; Dibangoye et al. 2016; Wu, Zilberstein, and Chen 2017). However, the associated privacy issues have not been addressed. Specifically, researchers usually assume that the Dec-POMDP model is constructed by domain experts and then solved by a planner in a centralized manner. In the modeling and planning phases, the experts and the planner generally have complete information about the problem and the agents. However, this assumption is limiting in domains where the agents belong to different parties and privacy is a concern. To date, privacy has been extensively studied for multi-agent systems in other contexts such as agent negotiation (Zhang and Makedon 2005), multi-agent reinforcement learning (Sakuma, Kobayashi, and Wright 2008), distributed constraint optimization (DCOP) (Léauté and Faltings 2013; Grinshpoun and Tassa 2014; Tassa, Zivan, and Grinshpoun 2015), and multi-agent classical planning (Brafman 2015), but has not been tackled in the more complex setting captured by Dec-POMDPs.

We propose *Privacy-Preserving Policy Iteration* (P3I) — a novel privacy-preserving planning algorithm for solving Dec-POMDPs. We consider three types of privacy that are relevant to multi-agent systems, that is, *agent privacy*, *model privacy*, and *decision privacy*. To preserve privacy, we devise a distributed message-passing algorithm based on trials. Specifically, our solution is model-free where each agent's policy is optimized iteratively with the local information collected in several trials by that agent. In more detail, we optimize each agent's policy using a variation of the *Cross-Entropy* (CE) method (Oliehoek, Kooij, and Vlassis 2008; Omidshafiei et al. 2016; Clark-Turner and Amato 2017). Like most of the existing algorithms for Dec-POMDPs, privacy issues are not concerned in the vanilla CE method (Oliehoek, Kooij, and Vlassis 2008). To address that, we use a *public-key homomorphic cryptosystem* to encrypt the messages and apply techniques such as *random masking* and *random permutation* to protect the agents' private information. Hence, our method is much more efficient than the general *secure function evaluation* (Goldreich 2009), which is impractical given a large number of trials.

We advance the state-of-the-art techniques of solving Dec-POMDPs with the following main contributions: (1)

We propose the *first* privacy-preserving algorithm for solving Dec-POMDPs *without* a third party, prove the correctness of the algorithm, and analyze its encryption/decryption and communication complexity; (2) We perform analysis of our algorithm with respect to three types of privacy and prove that it preserves agent privacy for non-neighbors, model privacy, and decision privacy; and (3) We conduct experiments on six common benchmark problems, which show that the time is mainly consumed by the encryption and decryption operations and can be significantly reduced by running the operations concurrently.

All in all, we show how privacy issues can be addressed with our approach for multi-agent problems modeled as Dec-POMDPs. This extends the applicability of Dec-POMDPs to domains where preserving privacy is required.

Preliminaries

This section briefly describes the Dec-POMDP model and the cryptographic tools used by our algorithm.

Decentralized POMDPs

Formally, a infinite-horizon *Decentralized Partially Observable Markov Decision Process* (Dec-POMDP) is defined as a tuple $\langle I, S, b^0, \{A_i\}, P, \{\Omega_i\}, O, R, \gamma\rangle$, where:

- I is a set of n agents.
- S is a set of states and b^0 is the initial state distribution.
- A_i is a set of actions for each agent $i \in I$ and $\vec{A} = \times_{i\in I} A_i$ is the set of joint actions.
- $P : S \times \vec{A} \times S \to [0, 1]$ is the transition function where $P(s'|s, \vec{a})$ is the probability of transiting to next state s' when taking joint action $\vec{a}$ in state s.
- Ω_i is a set of observations for each agent $i \in I$ and $\vec{\Omega} = \times_{i\in I}\Omega_i$ is the set of joint observations.
- $O : S \times \vec{A} \times \vec{\Omega} \to [0, 1]$ is the observation function where $O(\vec{o}|\vec{a}, s')$ is the probability of observing $\vec{o}$ after taking joint action $\vec{a}$ with outcome state s'.
- $R : S \times \vec{A} \to \mathbb{R}$ is the reward function where $R(s, \vec{a})$ is the reward after taking joint action $\vec{a}$ in state s.
- $\gamma \in (0, 1)$ is the discount factor.

A local policy $q_i : \bar{\Omega}_i \to A_i$ of agent $i \in I$ is a mapping from its local observation histories $\bar{\Omega}_i = (o_i^1, o_i^2, \cdots, o_i^t)$ to its actions A_i and a joint policy is a collection of local policies $\vec{q} = \langle q_1, q_2, \cdots, q_n\rangle$, one for each agent. The goal of solving a Dec-POMDP is to find a joint policy $\vec{q}^*$ that maximizes the expected accumulated rewards.

For infinite-horizon Dec-POMDPs, a local policy is usually represented as a *Finite State Controller* (FSC) where each controller node is associated with an action and has out-going links, one for each observation. Here, we consider a stochastic FSC as $\theta_i = \langle Q_i, \nu_{q_i}, \pi_{a_i q_i}, \lambda_{q_i' q_i o_i}\rangle$, where:

- Q_i is a finite set of controller nodes.
- $\nu_{q_i} \in \Delta(Q_i)$ is the initial node distribution.
- $\pi_{a_i q_i} \in [0, 1]$ is the probability of selecting action $a_i \in A_i$ in controller node $q_i \in Q_i$.
- $\lambda_{q_i' q_i o_i} \in [0, 1]$ is the probability of transiting to the next controller node $q_i' \in Q_i$ when observing $o_i \in \Omega_i$ in the current controller node $q_i \in Q_i$.

Given a joint FSC $\vec{\theta} = \langle\theta_1, \theta_2, \cdots, \theta_n\rangle$, the expected value at state s and joint controller node $\vec{q}$ can be computed recursively by the following Bellman equation:

$$V(s, \vec{q}) = (\prod_{i\in I} \pi_{a_i q_i})[R(s, \vec{a}) + \gamma \sum_{s'\in S, \vec{o}\in\vec{\Omega}} P(s'|s, \vec{a}) \; O(\vec{o}|\vec{a}, s') \sum_{\vec{q}'\in\vec{Q}} (\prod_{i\in I} \lambda_{q_i' q_i o_i}) V(s', \vec{q}')] \quad (1)$$

where $\vec{a}$ is the joint action selected by $\vec{q}$ and $\vec{q}'$ is the next joint controller node after observing $\vec{o}$ in joint controller node $\vec{q}$. Then, the goal is to find a joint FSC $\vec{\theta}^* = \text{argmax}_{\vec{\theta}} V(\vec{\theta})$ that maximizes the value function:

$$V(\vec{\theta}) = \sum_{s\in S, \vec{q}\in\vec{Q}} b^0(s)(\prod_{i\in I} \nu_{q_i}) V(s, \vec{q}) \quad (2)$$

Public-Key Homomorphic Cryptosystems

We make use of several existing cryptographic tools as the building blocks of our approach. Specifically, in our algorithm, we use a *public-key additive homomorphic cryptosystem*, which allows the agents to encrypt the message with a shared public key and perform the addition of encrypted values without requiring their decryption.

In a *public-key cryptosystem*, encryption of messages uses a public key that can be known to everyone, while decryption requires knowledge of the corresponding private key. Specifically, given a pair of (sk, pk) of private and public keys and a plaintext m, we denote $c = e_{pk}(m)$ an encryption of m and $m = d_{sk}(c)$ the decryption.

In an *additive homomorphic cryptosystem*, the addition computations are allowed on encrypted values without knowledge of the secret key. Specifically, there is some operation $\cdot$ without the knowledge of sk such that for any plaintexts m_1 and m_2, we have $e_{pk}(m_1 + m_2) = e_{pk}(m_1) \cdot e_{pk}(m_2)$. Based on this property, given a constant k and the encryption $e_{pk}(m)$, the multiplications by k can be computed via repeated application of the operator $\cdot$, i.e., $e_{pk}(km) = e_{pk}(m)^k$. Examples for such ciphers are Benaloh (Benaloh 1994) and Paillier (Paillier 1999) ciphers.

Privacy-Preserving Policy Iteration

We aim at optimizing each agent's policy while preserving their *privacy*. Indeed, the term *privacy* is quite broad, a fact that gave rise to several categories of privacy. Here, we consider three types of privacy that are relevant to multi-agent systems (Léauté and Faltings 2013).

Definition 1 (Agent privacy)**.** The requirement that no agent should be able to discover the *identity*, or even the *existence* of another agent.

Agent privacy is essential when some of the agents should remain anonymous. For example, some enterprises may not want their competitors or the general public to know they are parts of certain activities for business reasons.

Definition 2 (Model privacy)**.** The requirement that no agent should be able to discover the *model* of another agent, including its local states, actions, observations, local transition and observation probabilities, and local rewards.

Model privacy is an important issue, especially when agents represent different parties, so that an agent cannot gain more information about the other agents beyond what has been revealed from its own observations.

Definition 3 (Decision privacy)**.** The requirement that no agent should be able to discover the *decision rules* used by another agent.

Decision privacy is related to the agents' private strategies that should not be shared with their competitors or adversaries for business or security reasons. Otherwise, such information may give adversaries a competitive advantage.

Definition 4 (Semi-honest)**.** The assumption that all agents *honestly* follow the specified protocol, but might *curiously* use their records of intermediate computations in order to attempt to learn other agents' private information.

In this paper, we adopt a common assumption in private distributed computation (Goldreich 2009), that our agents behave *semi-honestly* (a.k.a., *honest but curious*).

Remark 1. *The agent privacy, model privacy, and decision privacy are not preserved in state-of-the-art planning algorithms for Dec-POMDPs.*

Generally, state-of-the-art planning algorithms for Dec-POMDPs do not explicitly consider privacy issues. Most of the algorithms assume that the Dec-POMDP model is given as prior information, before the planning phase, and the policies are computed in a centralized manner. In that case, the agents' identities, the model, and the agents' decisions are known by any agent who runs the planning algorithms. Therefore, no privacy is preserved in the planning phase. In fact, it is non-trivial to preserve privacy in those algorithms because the full knowledge about the model is required as an input and the agents' policies must be jointly evaluated.

Recently, model-free methods (Wu, Zilberstein, and Chen 2010a; Wu, Zilberstein, and Jennings 2013; Liu et al. 2015) have been proposed to solve Dec-POMDPs without the full knowledge of the model. However, they still record a set of samples of the agents' policy executions. The samples may leak the agents' private information, from which a third party can learn the agents' identities, the model, and even the agents' decisions. Indeed, our method is also model-free but with several modifications to protect the agents' private information and preserve the agent privacy, model privacy, and decision privacy as detailed in the next sections.

Algorithm Overview

We propose the *Privacy-Preserving Policy Iteration* (P3I) algorithm to preserve agent privacy, model privacy, and decision privacy when solving Dec-POMDPs. Similar to model-free approaches, our method computes the policies based on multiple trials instead of the fully specified model. In each trial, each agent simply executes its current policy in the environment and records its local information about the execution. The local information at each time step is a tuple $\langle q_i, a_i, o_i, r_i \rangle$, consisting of the visited controller node of the agent's policy q_i, the action taken by the agent a_i, the observation received from the environment o_i, and the local reward r_i. Specifically, in the k-th trial, we denote

$$h_i^k = (\langle q_i^{k,1}, a_i^{k,1}, o_i^{k,1}, r_i^{k,1} \rangle, \cdots, \langle q_i^{k,l}, a_i^{k,l}, o_i^{k,l}, r_i^{k,l} \rangle)$$

the local information recorded by agent i where l is the trial length and $H = \{\langle h_1^1, \cdots, h_n^1 \rangle, \cdots, \langle h_1^N, \cdots, h_n^N \rangle\}$ a set of information collected from N trials.

The main steps of our algorithm are outlined as follow:

1. Each agent i initializes its local policy θ_i^0 and $t \leftarrow 0$;
2. All agents use their current policies θ^t to perform N trials in the environment and records the results in H;
3. Each agent updates its current policy $\theta_i^{t+1} \leftarrow update(\theta_i^t)$ privately with the information sent by the other agents;
4. If no improvement for all agents' policies, then the process terminates; Otherwise $t \leftarrow t + 1$ and goto Step 2.

Starting from Step 1, each agent initializes its local policy without information shared with others. This can be done either by the agent's prior knowledge or just randomly. Running the trials in Step 2 is for each agent to execute its policy in the environment. Note that each agent only requires its local observation to execute the policy in the Dec-POMDP settings. Thus, no private information needs to be shared among agents during the execution.

Now, the key step determines how each agent updates its current policy privately given the information collected from the trials. In principle, private distributed computations can be carried out using *secure function evaluation* (Goldreich 2009), which is a general and well studied methodology for evaluating any function privately. However, although asymptotically polynomially bounded, these computations (e.g., Yao's garbled circuit protocol (Yao 1986)) can be too inefficient (i.e., exponential in time and space) for practical use, particularly when the input size is large. In the next section, we introduce a much simpler and more efficient procedure that the agents can execute for carrying out the secure computation.

Secure Policy Update

To update the policy, we need to solve the following optimization problem: $\vec{\theta}^{t+1} = \text{argmax}_{\vec{\theta}} V^H(\vec{\theta})$ with information H collected by executing the current policies in the environment. Generally, this involves two major procedures: value estimation and policy improvement, detailed below.

Value Estimation. The first step is to compute the value of the k-th trial by the sum of the discounted rewards:

$$V^k = \sum\nolimits_{t=1}^{l_k} [\gamma^t \cdot (\sum\nolimits_{i \in I} r_i^{k,t})] \tag{3}$$

Note that we assume that the rewards are shared in a privacy-preserving manner among the agents and therefore the total reward is the sum of the private shares of all the agents.

To ensure that value estimation preserves privacy, we encrypt the sum of the discounted rewards of each agent:

$$e_{pk}(V^k) = e_{pk}(\sum_{i \in I} V_i^k) = \prod_{i \in I} e_{pk}(V_i^k) \tag{4}$$

where $V_i^k = \sum_{t=1}^{l_k} \gamma^t r_i^{k,t}$ and the sum can be correctly carried out with the additive property of the additive homomorphic cryptosystem. Note that the discount factor γ is known to all agents and the value V_i^k can be computed by agent i privately, using only its own information h_i^k of the k-th trial.

Now, we can conduct the secure value estimation for each trial k as follow: (1) each agent i computes its part of the trial value: $V_i^k = \sum_{t=1}^{l_k} \gamma^t r_i^{k,t}$, (2) each agent i encrypts the value with a shared public key pk and sends $e_{pk}(V_i^k)$ to a computation agent[1], and (3) the computation agent sums up the encrypted values from all the agents and produces: $e_{pk}(V^k) = \prod_{i \in I} e_{pk}(V_i^k)$. Here, we assume that the computation agent does not have the private key sk so that it cannot decrypt the value $e_{pk}(V^k)$ and know $V^k, \forall k \in 1..N$.

Policy Improvement. We use the *Cross-Entropy* (CE) method, a probabilistic approach for stochastic optimization, for its simplicity. Several non-private variations of the CE method (Oliehoek, Kooij, and Vlassis 2008; Omidshafiei et al. 2016; Clark-Turner and Amato 2017) have been successfully applied to optimizing Dec-POMDP policies.

The basic idea of the CE method is an iterative two-phase process: (1) generating a set of samples according to some parameters, and (2) selecting the best N_b of the generated samples and using those to update the parameters. This process repeats until some stopping condition is met. In our algorithm, the sampling phase is done by the trials with the recorded information as samples. Below, we explain how to select the best N_b samples from the recorded trial information and how to use them to update policies parameters, while preserving privacy.

After running the secure value estimation, we have the encrypted values for all N trials: $\{e_{pk}(V^k), k \in 1..N\}$. Selecting the best N_b trials is trivial if we know $V^k, \forall k \in 1..N$ because we can simply sort V^k in descending order and choose the trials with the top N_b values. Unfortunately, in our algorithm, only the encrypted values $e_{pk}(V^k), \forall k \in 1..N$ are available. If we directly send all the encrypted values to some agent who has the private key sk, it may decrypt $e_{pk}(V^k)$ and have the private information of V^k, something which we are trying to avoid.

To address this, we randomly select a masking scalar $m \in \Re^+$, a standard cryptographic technique (Goldreich 2009), used to protect private information V^k as $e_{pk}(V^k)^m = e_{pk}(mV^k)$. Given this, we send $e_{pk}(V^k)^m$ to a computation agent[1] instead of $e_{pk}(V^k)$. After decryption, the computation agent gets the value of mV^k instead of V^k. Then, the computation agent sorts the values based on mV^k and returns the order, which is identical to the one based on V^k. To further protect the indices k, we randomly permute them so that the computation agent is not able to associate the value mV^k with any particular trial. Specifically, we map each k to a new index k' and recover it after the computation agent returns the sorted order. Given the order, we select the best N_b trials and use them to update the policy of each agent.

[1]The procedure of selecting the computation agent will be described in the implementation section.

Algorithm 1: Secure Value Estimation

```
Agent 1 runs the following procedure:
    (sk, pk) ← init the private and public keys once
    for k = 1 to N do
        V_1^k ← Σ_{t=1}^{l_k} γ^t r_1^{k,t}
        e_pk(V^k) ← e_pk(V_1^k)
    Send message ⟨pk, {e_pk(V^k)}⟩ to Agent 2
Once Agent i ∈ 2..n − 1 receives the message from Agent i − 1, Agent i runs the following procedure:
    for k = 1 to N do
        V_i^k ← Σ_{t=1}^{l_k} γ^t r_i^{k,t}
        e_pk(V^k) ← e_pk(V^k) · e_pk(V_i^k)
    Send message ⟨pk, {e_pk(V^k)}⟩ to Agent i + 1
Once Agent n receives the message from Agent n − 1, Agent n runs the following procedure:
    for k = 1 to N do
        V_n^k ← Σ_{t=1}^{l_k} γ^t r_n^{k,t}
        e_pk(V^k) ← e_pk(V^k) · e_pk(V_n^k)
```

Similar to the process above, we can reject trials with value less than the worst-performing trial from the previous iteration's best N_b trials. This is usually required by the CE method to discourage convergence to a local optimum. Specifically, we maintain $e_{pk}(V^{\min})$ as the encrypted value of the last one in the best N_b trials of the previous iteration. We send $e_{pk}(mV^{\min})$ to the computation agent and ask it to remove all the elements with $mV^k < mV^{\min}$ in the returned order. Hence the number elements of the returned order may be less than N_b. After that, we update $e_{pk}(V^{\min})$ with the value of the last one in the current best N_b trials, which corresponds to the value of the worst-performing trial obtained by the current iteration.

With the best N_b trials, the policy of each agent is updated by a *Maximum-Likelihood Estimate* (MLE) of the parameters as in the standard CE method (Oliehoek, Kooij, and Vlassis 2008). Notice that the MLE update can be conducted independently by each agent without requiring any private information of the other agents. To further discourage convergence to a local optimum, we also apply smoothed update to the policy parameter vector of each agent as follows: $\theta_i^{t+1} \leftarrow \alpha\theta_i^{t+1} + (1-\alpha)\theta_i^t$, where θ_i^{t+1} is the new parameter vector, θ_i^t the old one, and $\alpha \in (0, 1]$ is the learning rate.

Implementation. We propose a distributed implementation of the aforementioned privacy-preserving policy update process based on message passing among the agents. All of the computations are performed merely by the agents, without any third party. This is useful because the third party may introduce new privacy issues. Specifically, we arrange the agents in a chain structure ordered by their indices (i.e., $1, 2, \ldots, n$). We assume that each agent i can only exchange messages with its neighboring agents (i.e., agents $i-1$ and $i+1$, if existing) in the chain. With this structure, each agent only needs to know its neighbors instead of all the other

Algorithm 2: Secure Policy Improvement

Agent n runs the following procedure:
  $m \leftarrow$ select at random a positive masking scalar
  $e_{pk}(V^{\min})^m \leftarrow$ mask $e_{pk}(V^{\min})$ with m
  $\{e_{pk}(V^k)^m\} \leftarrow$ mask $\{e_{pk}(V^k)\}$ with m
  $\{e_{pk}(V^{k'})^m\} \leftarrow$ randomly permutes the indices
  Send $\langle e_{pk}(V^{\min})^m, \{e_{pk}(V^{k'})^m\}\rangle$ to Agent $n-1$

Once Agent $i \in n-1..2$ receives the message from Agent $i+1$, Agent i runs the following procedure:
  Forward message $\langle e_{pk}(V^{\min})^m, \{e_{pk}(V^{k'})^m\}\rangle$ received from Agent $i+1$ to Agent $i-1$

Once Agent 1 receives the message from Agent 2, Agent 1 runs the following procedure:
  $mV^{\min} \leftarrow d_{sk}(e_{pk}(V^{\min})^m)$
  $list \leftarrow \emptyset$
  for $k' = 1$ **to** N **do**
    $mV^{k'} \leftarrow d_{sk}(e_{sk}(V^{k'})^m)$
    if $mV^{k'} \geq mV^{\min}$ **then**
      $list \leftarrow list \cup \{\langle k', mV^{k'}\rangle\}$
  $order \leftarrow$ sort $list$ in descending order based on $mV^{k'}$
  $[k'_{N_b}] \leftarrow$ select indices of top N_b pairs in $order$
  Send message $[k'_{N_b}]$ to Agent 2

Once Agent $i \in 2..n-1$ receives the message from Agent $i-1$, Agent i runs the following procedure:
  Forward message $[k'_{N_b}]$ to Agent $i+1$

Once Agent n receives the message from Agent $n-1$, Agent n runs the following procedure:
  $[k_{N_b}] \leftarrow$ recover the best N_b trials from $[k'_{N_b}]$
  $e_{pk}(V^{\min}) \leftarrow \min_{k \in [k_{N_b}]} e_{pk}(V^k)$
  $\theta'_n \leftarrow$ MLE of θ_n using $\{h_n^k \in H_n, k \in [k_{N_b}]\}$
  $updated \leftarrow (||\theta'_n - \theta_n|| \geq \epsilon_n)$
  $\theta_n \leftarrow \alpha\theta'_n + (1-\alpha)\theta_n$
  Send message $\langle updated, [k_{N_b}]\rangle$ to Agent $n-1$

Once Agent $i \in n-1..2$ receives the message from Agent $i+1$, Agent i runs the following procedure:
  $\theta'_i \leftarrow$ MLE of θ_i using $\{h_i^k \in H_i, k \in [k_{N_b}]\}$
  $updated \leftarrow updated \wedge (||\theta'_i - \theta_i|| \geq \epsilon_i)$
  $\theta_i \leftarrow \alpha\theta'_i + (1-\alpha)\theta_i$
  Forward message $\langle updated, [k_{N_b}]\rangle$ to Agent $i-1$

Once Agent 1 receives the message from Agent 2, Agent 1 runs the following procedure:
  $\theta'_1 \leftarrow$ MLE of θ_1 using $\{h_1^k \in H_1, k \in [k_{N_b}]\}$
  $updated \leftarrow updated \wedge (||\theta'_1 - \theta_1|| \geq \epsilon_1)$
  $\theta_1 \leftarrow \alpha\theta'_1 + (1-\alpha)\theta_1$
  if $updated = false$ *or max-iters exceeded* **then**
    Send message $finished$ to Agent 2
    Terminate the procedure

Once Agent $i \in 2..n$ receives the message from Agent $i-1$, Agent i runs the following procedure:
  if $finished$ **then**
    Send $finished$ to Agent $i+1$ if $i \neq n$
    Terminate the procedure

agents during the message-passing procedures.

In practice, when there is a given interaction graph among the agents such as in ND-POMDPs (Nair et al. 2005), we can make use of that graph by finding a path in the graph that starts with an arbitrary agent and covers all the agents. Messages can be exchanged between agents along these links. By doing so, no additional agent identities are exposed and agents communicate only with their neighbors in the graph. Note that some agents may appear more than once on the path. In that case, special tokens can be used to ensure that each agent only does its computation once.

Algorithm 1 outlines the main procedures in our message-passing implementation of the *secure value estimation*. It starts with Agent 1 generating the private and public key pair (sk, pk) if the keys have not been initialized earlier. The public key pk is used by all the agents to encrypt messages while the private key sk is kept privately by Agent 1. In other words, only Agent 1 knows how to decrypt the message encrypted using the public key pk. Along the chain, each agent i first computes its share V_i^k of the values for every trial $k \in 1..N$, encrypts them with the public key pk, and then multiplies them with the values $\{e_{pk}(V^k)\}$ received from Agent $i-1$, if any. Given the *additive homomorphic cryptosystem*, this is equivalent to the sum of all values from Agent 1 to Agent i without the encryption. After that, Agent i sends the updated values $\{e_{pk}(V^k)\}$ and the public key pk to Agent $i+1$ when it exists. Finally, Agent n gets the encrypted sum of values for all trials.

Proposition 1 (Correctness)**.** *Algorithm 1 correctly computes the encrypted sum of the agents' values for all trials.*

Proof. This is can be proved directly using the homomorphic multiplication property of the cryptosystem:

$$\forall k \in 1..N, e_{pk}(V^k) = \prod_{i \in I} e_{pk}(V_i^k) = e_{pk}(\sum_{i \in I} V_i^k) \quad (5)$$

Starting from Agent 1, we apply the multiplication operator $\cdot$ incrementally, one by one for all agents. Finally, Agent n has the encrypted sum of all agents' values for all trials. Thus, we conclude the correctness of our algorithm. □

Proposition 2 (Complexity)**.** *Algorithm 1 requires a total of nN encryptions and the overall exchange of $n-1$ messages of size of $N+1$ per message.*

Note that the nN encryptions of V_i^k can be carried out concurrently because they are independent of each other.

Algorithm 2 shows our implementation of the *secure policy improvement* based on the CE method. Since Agent n has the encrypted sum $\{e_{pk}(V^k)\}$ after the secure value estimation, we start the computation from Agent n. It first selects at random a masking scalar m and uses it to protect $e_{pk}(V^{\min})$, the worst value of the best N_b trials from the previous iteration, and $\{e_{pk}(V^k)\}$, which produces $e_{pk}(V^{\min})^m$ and $\{e_{pk}(V^k)^m\}$. The indices in $\{e_{pk}(V^k)^m\}$ are randomly permuted so $\{e_{pk}(V^{k'})^m\}$ do not relate to the trial identities. Then, Agent n sends the secured messages $e_{pk}(V^{\min})^m$ and $\{e_{pk}(V^{k'})^m\}$ through the chain, which will finally reach Agent 1. After receiving the messages from Agent n, Agent 1 decrypts the messages with the private key

sk. As aforementioned, only Agent 1 owns the private key. Next, it makes a list with elements that have better value than the threshold $mV^{\min}$. Then, the list is sorted in a descending order and the indices $[k'_{N_b}]$ of the top N_b elements are selected, where $[\cdot]$ denotes an ordered list. Note that the size of $[k'_{N_b}]$ may be less than N_b because the elements in the list are filtered by $mV^{\min}$. At the end, the message $[k'_{N_b}]$ is sent by Agent 1 through the chain to Agent n.

When Agent n receives the message originally sent by Agent 1, it recovers $[k'_{N_b}]$ and obtains the correct identities $[k_{N_b}]$ of the best N_b trials. Then, the worst value $e_{pk}(V^{\min})$ is updated with the value of the last element in $[k_{N_b}]$. With the best N_b trials, all the agents are ready to update their policies. This process starts from Agent n. It first computes the MLE of the policy parameters using the best N_b trials, checks if there is any change made to the parameters, and applies a smoothed update to its policy parameters. The indices $[k_{N_b}]$ as well as the flag $updated$ are passed to the neighbor on the left side of the chain. One by one, each agent updates its policy parameters with the process similar to Agent n. When Agent 1 finally updates its policy, it takes one more step to check if any updates were been made to the agents' policies. If not, it indicates that the algorithm has converged because no improvement of the policy is possible for any of the agents. Then, Agent 1 passes the token $finished$ to the agents on the right side and terminates. All the other agents that receive the token forward it to their right-side neighbor, if any, and also terminate. At that point, the whole process of secure policy improvement is complete.

Proposition 3 (Correctness)**.** *Algorithm 2 correctly improves each agent's policy parameters based on the CE method.*

Proof. Notice that the key procedure is to select the best N_b trials where trials with value V^k less than the worst value $V^{\min}$ from the previous iteration's best N_b trials are rejected. In Algorithm 2, the computation is done by Agent 1 because it is the only agent who owns the private key sk. Hence it can decrypt the message and do the comparisons. Here, we use the masking values mV^k and $mV^{\min}$ to protect the original values V^k and $V^{\min}$ respectively. Given that m is a positive value, the comparisons can be carried out correctly using the masking values. Therefore, Agent 1 still can correctly select he best N_b trials using the masking values instead the true values. Finally, the true indices of the best N_b are recovered by Agent n, who maintain a mapping between the original indices and the permuted indices.

Given the best N_b trials, the policy improvement is done independently by each agent using the standard procedure of the CE method. This consists of computing the MLE of the parameters and applying the smoothed update with some learning rate. Thus, we conclude that our algorithm correctly improves each agent's policy parameters. □

Proposition 4 (Complexity)**.** *Algorithm 2 requires a total of N decryptions and exchanges at most $4(n-1)$ messages with a maximum size of $N+1$ per message.*

Here, the N decryptions can also be done concurrently.

Privacy Analysis

We perform an analysis of our algorithms with respect to the three types of privacy described earlier, i.e., agent privacy, model privacy, and decision privacy.

Proposition 5 (Agent privacy)**.** *The proposed algorithms preserve privacy about the identity or the existence of the non-neighboring agents in the chain.*

Proof. In the proposed algorithms, the agents are organized in a chain structure and each agent can only communicate with the neighboring agents in the chain. All the computations are run on the chain with message passing between two neighboring agents. All the messages used in the algorithms contain no information about the identity or the existence of the agents in the chain. Specifically, in Algorithm 1, the message is $\langle pk, \{e_{pk}(V^k)\}\rangle$, where pk is a public key and $\{e_{pk}(V^k)\}$ is a set of encrypted values, one for each trial. Algorithm 2 relies on passing the following messages: several encrypted values $\langle e_{pk}(V^{\min})^m, e_{pk}(V^{k'})^m\rangle$, a set of indices $[k'_{N_b}]$, a boolean flag and a set of indices $\langle updated, [k_{N_b}]\rangle$, and a boolean token $finished$. None of them leak the private information about the identity or the existence of the agents. After running the algorithms, agents only know the neighboring agents in the chain. Thus, we conclude that the algorithms preserve privacy about the identity or the existence of the non-neighboring agents. □

Proposition 6 (Model privacy)**.** *The proposed algorithms preserve privacy about the model of the other agents.*

Proof. In the proposed algorithms, each agent i only records its local information $\langle q_i, a_i, o_i, r_i\rangle$ about trials. Throughout the algorithms, no information about the states, the transition function, and the observation function is required. The agents know the index and the total time steps of each trial. The information about each agent i's controller node q_i, action a_i, and observation o_i is kept privately by the agent itself. Only the local reward r_i is used by the algorithms, but it is securely protected with encryption.

Specifically, in Algorithm 1, the sum of discounted local reward V_i^k is protected by encryption $e_{pk}(V_i^k)$ using the public key pk. The private key is owned only by Agent 1. In other words, no other agents except Agent 1 can decrypt the value $e_{pk}(V_i^k)$. The algorithm does not require that the other agents directly send the value $e_{pk}(V_i^k)$ back to Agent 1. Base on the *semi-honest* assumption, all the other agents will not do so because this is not allowed by the algorithm. Furthermore, only Agent 2 knows the identity and the existence of Agent 1. On the one hand, Agent 2 will not send the intermediate message $e_{pk}(V_1^k) \cdot e_{pk}(V_2^k)$ back to Agent 1 as it will leak its own private information V_2^k. One the other hand, Agent 1 will not tell Agent 2 the private key sk as this will leak the private information V_1^k of Agent 1.

In Algorithm 2, the private information of the encrypted values $\{e_{pk}(V^k)\}$ is protected by random masking and random permutation before they are sent to Agent 1. The protections are conducted by Agent n. Note that the random masking scalar m and the mapping of the random permutations are privately owned by Agent n. Therefore, after Agent

1 decrypts $e_{pk}(V^{k'})^m$ and obtains $mV^{k'}$, it does not know the true value $V^{k'}$ because of random masking and the relation with the trials due to random permutation. Indeed, it leaks some *excessive* information about the true values but that excessive information is benign and of no practical use for Agent 1. More specifically, agents may know that some values are non-zero, positive, or negative after random masking. However, they cannot know the true values or even numerical relationship between the values. With random permutation, they cannot associate the values with any specific trials. Therefore, agents cannot infer the private information of the other agents' model or behavior with these numbers.

Thank to the *semi-honest* assumption, Agent 1 will not leak $mV^{k'}$ to Agent n and Agent n will not share information about the random masking and random permutation with Agent 1. In fact, Agents 1 and n do not know the identity or the existence of each other and cannot directly communicate with each other without all the other agents between them. The computation of MLE and smoothed update are done by each agent independently without referring to the model knowledge of the other agents. Therefore, we conclude that the algorithms preserve model privacy. □

Proposition 7 (Decision privacy)**.** *The proposed algorithms preserve privacy about the decision of the other agents.*

Proof. This is straightforward because agents do not share their policies with each other and the policies are optimized locally by each agent in the algorithms. Additionally, the local information recorded in the trials is either kept privately by each agent (e.g., actions, observations, etc.) or protected by encryption (i.e., rewards) throughout the algorithms. Hence the agents are not able to learn the other agents' policies from the trials. Thus, we conclude that the proposed algorithms preserve decision privacy. □

Experiments

We implemented our algorithm and tested it on 6 common benchmark problems[2] for Dec-POMDPs (i.e., Dec-Tiger, Broadcast Channel, Meeting in a 3×3 Grid, Box Pushing, Recycling Robots, and Mars Rovers). For each problem instance, we first ran our algorithm to generate policies and then evaluated the policies by simulation. In the algorithm, we set the number of trials $N = 1000$ and the size of best trials $N_b = 10$. To simulate the private settings, we randomly split reward values among the agents so each agent owned its local portion of the rewards. Note that an agent's local rewards should be hidden from the other agents as this is an important aspect of model privacy. The encryption and decryption operations were performed by the common Java implementation of the Paillier cryptosystem[3].

Since P3I is a privacy-preserving variation of the CE method, we compared it with DICE (Oliehoek, Kooij, and Vlassis 2008) — the vanilla CE method for Dec-POMDPs. The goal is to empirically evaluate the performance of our privacy-preserving techniques. As expected, both P3I and DICE produced policies with identical quality for all the tested problems. This confirms the correctness of P3I. In what follows, we mainly analyze the runtime of P3I on the 6 benchmark problems as shown in Table 1.

Table 1: Runtime Results of Benchmark Problems

Problem	**Runtime (sec.)**		
	DICE	P3I	Parallel P3I
Dec-Tiger	3.9 ± 0.2	7350 ± 260	424 ± 9
Broadcast Channel	4.5 ± 0.1	7070 ± 214	433 ± 12
Meeting in Grid	11.3 ± 1.0	6424 ± 180	381 ± 11
Box Pushing	6.1 ± 0.3	6907 ± 254	425 ± 12
Recycling Robots	4.3 ± 0.3	7404 ± 277	433 ± 12
Mars Rovers	11.8 ± 1.0	6897 ± 158	443 ± 10

As expected, the most time-consuming operations of P3I are the encryption and decryption. This is not surprising since the Paillier cryptosystem requires much more complex operations than the CE method. For instance, in the Box Pushing problem, the encryption and decryption operations took more than 99% of the total time while the CE method only required less than 1% of the time for sampling and policy updating. As we pointed out above, the time complexity of P3I for encryption and decryption depends on the number of trials. This is confirmed by our results. As Table 1 shows, the runtime of P3I is around 7000 seconds for all the tested problems as we used 1000 trials for all instances. We also observed that the runtime of P3I only increased linearly as the number of trials grows.

Fortunately, the encryption and decryption operations in P3I can be run concurrently as the encryption and decryption of the trial values are independent of each other. The runtime results of the parallel P3I on a machine with 40 cores is shown in Table 1 as Parallel P3I. As we can see, parallel P3I achieved *one order of magnitude* improvement in runtime. The runtime can be further reduced if more CPU cores are available. As with P3I, the runtime of parallel P3I is about 400 seconds across all problem instances.

In summary, we can see that privacy in P3I comes at a cost in runtime as the encryption and decryption operations are time-consuming. The runtime can be improved by running these operations concurrently on a multi-core machine.

Conclusions

We present P3I — the first privacy-preserving planning algorithm for solving Dec-POMDPs based on message passing among the agents and a public-key homomorphic cryptosystem to encrypt the messages. We proved the correctness and analyzed the complexity of P3I. Most importantly, we analyzed the privacy properties of P3I and proved that it preserves agent privacy of non-neighbors, model privacy, and decision privacy. Experimental results on six Dec-POMDP benchmark problems confirm our propositions and show that running the time-consuming encryption and decryption operations concurrently can substantially improve the runtime of P3I. In the future, we plan to integrate our privacy-preserving techniques with other Dec-POMDP solvers that are more sample efficient and investigate model-based ap-

[2] http://masplan.org/problem_domains

[3] https://www.csee.umbc.edu/∼kunliu1/research/Paillier.html

proaches where private and public actions are explicitly specified (Brafman 2015).

Acknowledgments

This work was supported in part by the National Natural Science Foundation of China (grant No. 61603368), the Youth Innovation Promotion Association of CAS (grant No. 2015373), the Natural Science Foundation of Anhui Province (grant No. 1608085QF134) and the US National Science Foundation (grant No. IIS-1405550).

References

Amato, C.; Dibangoye, J. S.; and Zilberstein, S. 2009. Incremental policy generation for finite-horizon DEC-POMDPs. In *Proceedings of the International Conference on Automated Planning and Scheduling*, 2–9.

Benaloh, J. 1994. Dense probabilistic encryption. In *Workshop on Selected Areas of Cryptography*, 120–128.

Bernstein, D. S.; Givan, R.; Immerman, N.; and Zilberstein, S. 2002. The complexity of decentralized control of Markov decision processes. *Mathematics of Operations Research* 27(4):819–840.

Brafman, R. I. 2015. A privacy preserving algorithm for multi-agent planning and search. In *Proc. of the International Joint Conference on Artificial Intelligence*, 1530–1536.

Clark-Turner, M., and Amato, C. 2017. COG-DICE: An algorithm for solving continuous-observation Dec-POMDPs. In *Proc. of the International Joint Conference on Artificial Intelligence*, 4573–4579.

Dibangoye, J. S.; Amato, C.; Buffet, O.; and Charpillet, F. 2016. Optimally solving Dec-POMDPs as continuous-state MDPs. *Journal of Artificial Intelligence Research* 55:443–497.

Goldreich, O. 2009. *Foundations of Cryptography: Volume 2, Basic Applications*. Cambridge University Press.

Grinshpoun, T., and Tassa, T. 2014. A privacy-preserving algorithm for distributed constraint optimization. In *Proc. of the International Conference on Autonomous Agents and Multiagent Systems*, 909–916.

Kumar, A.; Zilberstein, S.; and Toussaint, M. 2015. Probabilistic inference techniques for scalable multiagent decision making. *Journal of Artificial Intelligence Research* 53:223–270.

Léauté, T., and Faltings, B. 2013. Protecting privacy through distributed computation in multi-agent decision making. *Journal of Artificial Intelligence Research* 47:649–695.

Liu, M.; Amato, C.; Liao, X.; Carin, L.; and How, J. P. 2015. Stick-breaking policy learning in Dec-POMDPs. In *Proc. of the International Joint Conference on Artificial Intelligence*, 2011–2018.

Nair, R.; Varakantham, P.; Tambe, M.; and Yokoo, M. 2005. Networked distributed POMDPs: A synthesis of distributed constraint optimization and POMDPs. In *Proc. of the AAAI Conference on Artificial Intelligence*, 133–139.

Oliehoek, F. A.; Spaan, M. T. J.; Amato, C.; and Whiteson, S. 2013. Incremental clustering and expansion for faster optimal planning in Dec-POMDPs. *Journal of Artificial Intelligence Research* 46:449–509.

Oliehoek, F. A.; Kooij, J. F. P.; and Vlassis, N. 2008. The cross-entropy method for policy search in decentralized POMDPs. *Informatica* 32(4).

Omidshafiei, S.; Agha-Mohammadi, A.-A.; Amato, C.; Liu, S.-Y.; How, J. P.; and Vian, J. 2016. Graph-based cross entropy method for solving multi-robot decentralized POMDPs. In *Proc. of the IEEE International Conference on Robotics and Automation*, 5395–5402.

Paillier, P. 1999. Public-key cryptosystems based on composite degree residuosity classes. In *Proc. of the International Conference on the Theory and Applications of Cryptographic Techniques*, volume 99, 223–238.

Pajarinen, J. K., and Peltonen, J. 2011. Periodic finite state controllers for efficient POMDP and DEC-POMDP planning. In *Proc. of Neural Information Processing Systems*, 2636–2644.

Ramchurn, S. D.; Huynh, T. D.; Wu, F.; Ikuno, Y.; Flann, J.; Moreau, L.; Fischer, J. E.; Jiang, W.; Rodden, T.; Simpson, E.; Reece, S.; Roberts, S.; and Jennings, N. R. 2016. A disaster response system based on human-agent collectives. *Journal of Artificial Intelligence Research* 57:661–708.

Sakuma, J.; Kobayashi, S.; and Wright, R. N. 2008. Privacy-preserving reinforcement learning. In *Proc. of the International Conference on Machine Learning*, 864–871.

Tassa, T.; Zivan, R.; and Grinshpoun, T. 2015. Max-sum goes private. In *Proc. of the International Joint Conference on Artificial Intelligence*, 425–431.

Wu, F.; Ramchurn, S. D.; Jiang, W.; Fischer, J. E.; Rodden, T.; and Jennings, N. R. 2015. Agile planning for real-world disaster response. In *Proc. of International Joint Conference on Artificial Intelligence*, 132–138.

Wu, F.; Zilberstein, S.; and Chen, X. 2010a. Rollout sampling policy iteration for decentralized POMDPs. In *Proc. of the Conference on Uncertainty in Artificial Intelligence*, 666–673.

Wu, F.; Zilberstein, S.; and Chen, X. 2010b. Trial-based dynamic programming for multi-agent planning. In *Proc. of the AAAI Conference on Artificial Intelligence*, 908–914.

Wu, F.; Zilberstein, S.; and Chen, X. 2017. Multi-agent planning with baseline regret minimization. In *Proc. of the International Joint Conference on Artificial Intelligence*, 444–450.

Wu, F.; Zilberstein, S.; and Jennings, N. R. 2013. Monte-carlo expectation maximization for decentralized POMDPs. In *Proc. of the International Joint Conference on Artificial Intelligence*, 397–403.

Yao, A. C.-C. 1986. How to generate and exchange secrets. In *Proc. of IEEE Symposium on Foundations of Computer Science*, 162–167.

Zhang, S., and Makedon, F. 2005. Privacy preserving learning in negotiation. In *Proceedings of ACM Symposium on Applied Computing*, 821–825.