2023 62nd IEEE Conference on Decision and Control (CDC)
December 13-15, 2023. Marina Bay Sands, Singapore

Oblivious Markov Decision Processes: Planning and Policy Execution

Murtadha Alsayegh[1], Jose Fuentes[1], Leonardo Bobadilla[1], and Dylan A. Shell[2]

***Abstract*— We examine a novel setting in which two parties have partial knowledge of the elements that make up a Markov Decision Process (MDP) and must cooperate to compute and execute an optimal policy for the problem constructed from those elements. This situation arises when one party wants to give a robot some task, but does not wish to divulge those details to a second party—while the second party possesses sensitive data about the robot's dynamics (information needed for planning). Both parties want the robot to perform the task successfully, but neither is willing to disclose any more information than is absolutely necessary. We utilize techniques from secure multi-party computation, combining primitives and algorithms to construct protocols that can compute an optimal policy while ensuring that the policy remains opaque by being split across both parties. To execute a split policy, we also give a protocol that enables the robot to determine what actions to trigger, while the second party guards against attempts to probe for information inconsistent with the policy's prescribed execution. In order to improve scalability, we find that basis functions and constraint sampling methods are useful in forming effective approximate MDPs. We report simulation results examining performance and precision, and assess the scaling properties of our Python implementation. We also describe a hardware proof-of-feasibility implementation using inexpensive physical robots, which, being a small-scale instance, can be solved directly.**

***Index Terms*— Linear programming, Markov Decision Process, Robotics, Control Systems Privacy.**

"There are many things of which a wise man might wish to be ignorant." — Ralph Waldo Emerson

I. Introduction

Autonomous robots that help with household labor, chores, and routine or repetitious tasks in the domestic sphere remain frustratingly out of reach—service robots are far from ubiquitous, having failed to penetrate markets in any meaningful way. Indeed, the last five years have been a tougher time for consumer robots than many —including seasoned technology entrepreneurs— had anticipated [1]. Perhaps one solution to these issues is *Robots-as-a-Service* (RaaS), an alternative business model in which service providers lease robots to users, reducing the consumer's initial outlay and alleviating uncertainty surrounding maintenance and repair.

Now add to this an important recent trend: the upsurge in concerns voiced regarding privacy and confidentiality, including specific objections to companies selling data collected by robots [2]. This paper addresses the question of how to operate a robot, as a resource shared between parties, without divulging sensitive information. We imagine a user who, leasing a robot from a service provider, has some tasks she wishes the robot to perform, yet does not want to share details of the work she has the robot carry out. (See Fig. 1.) The service provider deploys the robot and monitors its general condition, pushing software upgrades as appropriate. To boost their market share and attract users who prize privacy or have concerns over data confidentiality, the service provider guarantees that they will never track the robot, nor permit people snooping on network communications to learn about the robot's activities. Further, the service provider has a number of proprietary aspects of the robot they wish to protect and avoid becoming public. The problem becomes: (i) how to plan by combining some of the consumer's information (the task itself and the robot's current state) and some of the service provider's (aspects of robot performance) without transmitting that information to the other party, and (ii) having found a plan, how to execute it.

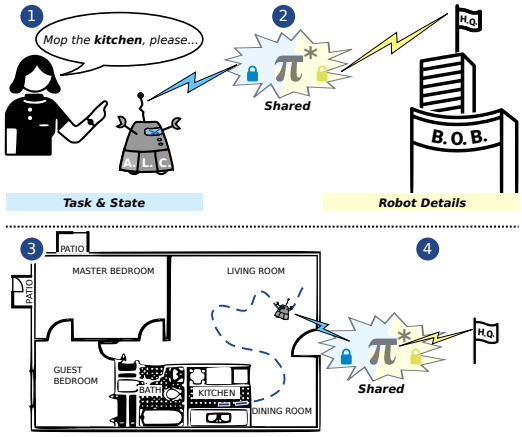


Fig. 1: Oblivious MDPs illustrated. The user gives her robot, A.L.C. (pronounced Alice), high-level commands (step 1). From this, the robot constructs a policy $\pi^\star$ by interacting with the manufacturer (step 2). Both keep their respective informative private: the manufacturer does not learn the task (i.e., reward function); the robot owner does not obtain sensitive proprietary details of the robot design (i.e., transition dynamics). The pair of parties keep interacting in order to use the policy (3 & 4), the robot executing $\pi^\star$ without disclosing states, even while subject to uncertain transitions, and the manufacturer verifying that the queries reflect a plausible execution.

This paper treats the planning problem via the Markov Decision Processes (MDP) framework. The user places the robot in her home and specifies the task via a reward function (we imagine an interface where a goal is selected, putting a unit reward at the goal state, zero elsewhere). The robot, which we refer to as Alice, has some control programs that it can trigger; the hardware and control programs are opaque

[1]M. Alsayegh, J. Fuentes, and L. Bobadilla are with the School of Computing and Information Sciences, Florida International University, Miami, FL 33199, USA {malsa061@,jfuent099@,bobadilla@cs.}fiu.edu

[2]D. A. Shell is with the Department of Computer Science & Engineering at Texas A&M University, College Station, TX, USA dshell@tamu.edu

This material is based upon work supported by the NSF under Grants HRD-1547798 and HRD-2111661. These grants were awarded to FIU as part of the CREST Program, grants IIS-2034123, IIS-2024733, IIS-2034097, and DHS grant 2017-ST-062000002. This is contribution #1608 from IoE.

and treated as actions in the MDP. The service provider, Bots-of-Boston™ (or B.O.B., for short) has conducted extensive system identification on these controllers; they consider the transition dynamics to be sensitive information. (Also, these may change as the control programs are updated to incorporate new features.) Some of the information in the MDP is available to one party, and some is available to the other. The innovation in this paper is to use secure multi-party computation (SMPC) [3], [4] to compute and execute a policy that is shared. After the computation, we propose an execution protocol in which neither Alice nor B.O.B. have access to the policy itself, but each has only pieces that are meaningless without the other. Having completed the planning phase, the robot uses this shared policy: it localizes itself and has Alice and B.O.B. compute the action to execute (i.e., controller to trigger). On Alice's side, the action is determined (since the robot must know what to do), but this occurs without B.O.B. knowing what state the robot is in. This process is repeated as the robot executes, with B.O.B. ensuring that the state sequence is plausible, using knowledge of the transition dynamics without ever learning the states or reward function that encodes Alice's task.

II. Related Work

The need for privacy and security in robot applications has garnered recent attention in the robotics community [5], [6]. Our approach to planning fits within the family of approaches that use cryptography-based methods in estimation and control. Several cryptographic techniques have been used, such as homomorphic encryption for a privacy-preserving Linear Quadratic Gaussian (LQG) controller in a multi-party setting [7] and Reinforcement Learning [8] and garbled circuits for Model Predictive Control [9]. Researchers have also used alternative methods to achieve privacy guarantees in control and automation, including differential privacy [10], [11] and federated learning [12], and models with worst-case nondeterminism [13], [14], [15].

Our work connects to techniques using Secret Sharing Schemes [16] for privacy-preserving machine learning [17], [18], [19], [20], optimization [21], [22], [23], sensing [24], and networking [25]. In our previous work, we proposed an approach based on garbled circuits and homomorphic encryption for privacy-preserving rendezvous and collision avoidance [26] and multi-robot task allocation [27] through Shamir's Secret Sharing. Secure Linear Programming [18], [19] gives a useful coordination primitive. In this paper, we use it to compute optimal policies for MDPs, which—unlike the previous two [27],[26]—has a probabilistic model of uncertainty. The work by Mangasarian et al. [28] proposes a method to solve a linear program by dividing a matrix among n agents. While their approach is relevant for seeking a secure solution, but it is not directly applicable to our setting because it gives Alice more information than needed.

III. Preliminaries and Problem Formulation

To recap to story so far: we have two parties, Alice and B.O.B., the former has a navigation task to execute but needs the latter's help to do so. This two party setting arises in the *secure rental problem*: a person considers using one of B.O.B.'s robots in order to perform useful work for her, but she is reticent to share either the task specifics or the information gleaned during task execution. B.O.B. only obtains a fee if the user is persuaded, via guarantees of privacy, to rent the robot. Moreover, B.O.B. wishes to constrain what others can learn about the design and specific performance parameters of the robot. Naturally, everyone wishes protection against third-party attacks during the rental.

Formally speaking, the robot is modeled using a Markov Decision Process (MDP) a 5-tuple: (S, A, T, R, γ) given by the state space S; the action space A; the transition function $T : S \times A \times S \to [0, 1]$, when evaluated as $T(s, a, s')$ gives the probability of arriving to state s' given the agent was in state s and applied action a. The function T is owned by B.O.B. as it contains the knowledge about the robot's dynamics; the reward function $R : S \times A \to \mathbb{R}$, where $R(s, a)$ is the reward obtained by executing action a at state s; and $0 < \gamma < 1$ is the discount factor. Alice owns the reward function and discount, as she knows the task requirements.

Secure Decentralized MDP: *Given the parties Alice, who owns the reward function and the discount factor, and B.O.B., who owns the transition function, build, solve and, execute an MDP securely, using multi-party computation.*

IV. Secure Multi-Party Computation Framework

This section aims to make the paper self-contained, describing the elements of secure multi-party computation needed. More detail appears in standard texts [3], [4], [29].

A. Shamir's Secret Sharing

In this paper, we use Shamir's Secret Sharing (SSS)-based multi-party computation [16]. This scheme splits a secret $\mathcal{S}$ across ℓ parties, by creating a set of ℓ shares indicated by square brackets: $\{[\mathcal{S}]_1, [\mathcal{S}]_2, \ldots, [\mathcal{S}]_\ell\}$ with the property that secret $\mathcal{S}$ can be recovered if and only if a subset of at least t parties combine their shares.

The SSS scheme depends on the fact that t points suffice to uniquely define a polynomial of degree less or equal to $t-1$ on a finite field denoted by $GF(p) = \mathbb{Z}_p$, with prime p. To construct polynomial $f(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{t-1} x^{t-1}$, we represent the secret $\mathcal{S}$ as an element $a_0 \in GF(p)$ and the $a_1, \ldots, a_{t-1}$ are randomly chosen from $GF(p)$. After the polynomial has been constructed, ℓ points are obtained from it and one is given to each participant. Specifically, for participant i, $1 \le i \le \ell$, the pair $(i, f(i))$ is derived from the polynomial as illustrated in Protocol 1 (for more detail, see [17]).

Protocol 1 *ShamirShare*$(\mathcal{S}, \ell, t, p)$

Inputs: $\mathcal{S} \in GF(p)$ the secret, ℓ parties, t reconstruction threshold
Output: Shares $[\mathcal{S}]_j$ for each of the ℓ parties

1 Party P_i selects $a_1, a_2, \ldots, a_{t-1}$ from $GF(p)$
2 **foreach** $j \in \{1, 2, \ldots, \ell\}$
3 $\quad [\mathcal{S}]_j \leftarrow \mathcal{S} + \sum_{k=1}^{t-1} a_k j^k$
4 $\quad$ Send $[\mathcal{S}]_j$ to party P_j

Reconstruction is via *OpenShare*$(A, [\mathcal{S}])$, where P is a set of parties with $|P| \geq t$ and $[\mathcal{S}]$ are shares of the secret. The routine *OpenShare* interpolates the polynomial to obtain $f(0) = \mathcal{S}$, and implements Lagrange interpolation [17].

B. Secure Operations on Secrets

One critical aspect of SSS is that it enables each party to perform locally linear computations of secrets and public values. Mostly, we employ basic operations such as addition, multiplication, comparison, and dot product between two vectors. (Implementation and other details appear in [30].) These are building blocks for the more complex protocols in this work: our initial focus will be a secure solution for Linear Programming (LP) problems through the simplex method [31], [22], as the optimal value function of an MDP can be obtained by solving an LP.

V. Methods

We chose Linear Programming (LP) over alternatives like value or policy iteration due to its inherent benefits. LP provides greater flexibility to customize the objective function and reduce constraints, and it allows us to use basis functions, as explained in subsection VI-C. Moreover, LP requires fewer runtime iterations, enhancing efficiency.

The entire execution consists of 4 parts: *A.*) converting a given MDP into an LP through its tableau representation; *B.*) solving the result; *C.*) converting the optimal value function to a policy for Alice, and *D.*) execution via interaction.

A. From MDP problems to LP problems

The initial phase involves Protocol 2, takes as input the elements of the 5-tuple representing the MDP problem using the components owned by Alice and B.O.B. and converts it to an LP problem. The state space S and the action set A are known by both parties. The transition function T is known by B.O.B. only, while the reward function R, and the discount factor γ are known solely by Alice.

In this formulation, the value function $V : S \to \mathbb{R}$ associated with the MDP problem satisfies the following [32]:

$$\begin{aligned} \min_{s\in S} \quad & \sum_{s\in S} \mu(s)V(s) \\ \text{subject to} \quad & V(s) \geq R(s,a) + \gamma \sum_{s'\in S} T(s,a,s')V(s'), \\ & V(s) \geq 0, \quad \text{for } s \in S \text{ and } a \in A. \end{aligned} \tag{1}$$

Here $\mu(s) > 0$ for $s \in S$ is a probability distribution over the state space for the initial state. This is converted into its canonical form, thusly:

$$\begin{aligned} \min_{v} \quad & \mu^\top v \\ \text{subject to} \quad & r = Mv,\ v \geq 0,\ r \geq 0, \end{aligned} \tag{2}$$

which provides a shorthand for the different functions involved. To begin, we need to identify a *basic feasible solution*, i.e. a vector v located on a vertex of the constraint polyhedron. However, the simplex method employed in [22] is tailored to specific constraint formats, simplifying basic feasible solution derivation via a single tableau. As (1) lacks the requisite format, a two-phase simplex algorithm becomes necessary. The first phase determines a basic feasible solution by solving the *artificial problem* derived from (1):

$$\begin{aligned} \min_{v,\mathrm{a}} \quad & \mathbb{1}^\top \mathrm{a} \\ \text{subject to} \quad & r = Mv + \mathrm{a},\ v \geq 0,\ r \geq 0, \text{ and } \mathrm{a} \geq 0. \end{aligned} \tag{3}$$

Here, $\mathbb{1} = (1, \ldots, 1)^\top$, and $(v, \mathrm{a}) = (0, r)$ consistently forms a basic feasible solution, facilitating the standard simplex method. This strategy deploys two concurrent tableaux, $\mathcal{T}$ for (3) and $\mathcal{U}$ for (2),

$$\mathcal{T} = \begin{bmatrix} \mathbb{1} & \mathbb{1}^\top r \\ M & r \end{bmatrix}, \quad \text{and} \quad \mathcal{U} = \begin{bmatrix} \mu & 0 \\ M & r \end{bmatrix}. \tag{4}$$

The second phase of the simplex procedure uses $\mathcal{T}$ to select the entering and exiting variables. As detailed shortly, the standard Gauss pivoting procedure acts on both matrices. Protocol 2 gives *MDPToLP* to generate $\mathcal{T}$ and $\mathcal{U}$ from (S, A, T, R, γ). The parentheses $\langle\cdot\rangle_{Alice}$, $\langle\cdot\rangle_{Bob}$ denote elements that are owned by Alice and B.O.B respectively; $[\cdot]$ denotes secure elements shared between them.

Protocol 2 *MDPToLP*$(S, A, \langle T\rangle_{Bob}, \langle R\rangle_{Alice}, \langle\gamma\rangle_{Alice}, \mu)$

Inputs: The MDP (S, A, T, R, γ) with transitions, rewards, and discount; also μ an initial probability distribution

Output: Shared tableaux $[\mathcal{T}]$ and $[\mathcal{U}]$

Alice shares $[\gamma] \leftarrow$ *ShamirShare*$(\langle\gamma\rangle_{Alice})$
foreach $(s, a) \in S \times A$ **do locally in parallel**
  Alice shares $[R(s, a)] \leftarrow$ *ShamirShare*$(\langle R(s, a)\rangle_{Alice})$
foreach $(s, a, s') \in S \times A \times S$ **do locally in parallel**
  B.O.B. shares $[T(s, a, s')] \leftarrow$ *ShamirShare*$(\langle T(s, a, s')\rangle_{Bob})$
foreach $(s, a, s') \in S \times A \times S$ **do**
  $[N_{sas'}] \leftarrow \text{sign}([-R(s, a)])(1 - [\gamma][T(s, a, s')])$
foreach $(s, a) \in S \times A$ **do**
  $[\mathcal{M}_{(s,a)}] \leftarrow ([N_{sas_1}], \ldots, [N_{sas_{|S|}}])$
  $[\hat{e}_{f(s,a)}] \leftarrow \text{sign}([-R(s, a)])e_{f(s,a)}]$
  $[\mathcal{T}(f(s, a), \cdot)] \leftarrow ([\mathcal{M}_{f(s,a)}], [\hat{e}_{f(s,a)}], |[R(s, a)]|)$
  $[\mathcal{U}(f(s, a), \cdot)] \leftarrow ([\mathcal{M}_{f(s,a)}], [\hat{e}_{f(s,a)}], |[R(s, a)]|)$
$[\mathcal{T}(1, \cdot)] \leftarrow (\mathbb{1}, [\mathbb{1}^\top r])$ ▷ $r = \big(R(f^{-1}(2)), \ldots, R(f^{-1}(|S||A|+1))\big)^\top$
$[\mathcal{U}(1, \cdot)] \leftarrow [(\mu, 0)]$
return $([\mathcal{T}], [\mathcal{U}])$

Some useful conventions will be adopted in presentation of subsequent the protocols: if M is a matrix we will refer to $M(i, \cdot)$ and $M(\cdot, j)$ to its i^{th} rows and j^{th} columns respectively. Also, if x and y are vectors, vector (x, y) will mean the concatenation of those vectors. In the above protocol for *MDPToLP*, $f : S \times A \to \{2, \ldots, |S||A| + 1\}$ is any bijection between $S \times A$ and $\{2, \ldots, |S||A| + 1\}$; also e_i denotes the i^{th} canonical vector whose size is $|A||S|$. (Note that colors are used to help indicate visually when a protocol is defined herein, or is leveraging the prior work of others.)

B. Solving the LP problem

After obtaining the tableaux, $[\mathcal{T}]$ and $[\mathcal{U}]$, next, we securely solve the LP. In essence, an implementation is just successive application of the basic operations mentioned before, since selecting the entering/exiting variables, and performing Gaussian elimination on the tableaux can be achieved through element comparisons and dot products.

Our approach makes heavy use of the secure LP solver implemented in [22]. The main difference is that, in the

extension *TwoPhaseIteration* (below), we handle the two tableaux $[\mathcal{T}]$ and $[\mathcal{U}]$, by using the protocol *Iteration* as a subroutine (see call on line 13), it being responsible for a single simplex iteration. The protocol performs iterations in a two-phase manner, which required the addition of a flag to indicate failure in locating, through (3), any basic feasible solution. The primitives *GetPivCol*, *GetPivRow*, *SecReadRow*, *SecRead*, *UpdateTab*, *UpdateVar* and *Iteration* perform the basic pivoting and reduction operations in a single iteration of the simplex algorithm, and are detailed in [22].

Protocol 3 *TwoPhaseIteration*$([\mathcal{T}], [\mathcal{U}], [B], [N_b])$

Inputs: Shared tableaux $[\mathcal{T}]$ and $[\mathcal{U}]$, and initialized vectors corresponding to the basic and non basic variables $[B]$ and $[N_b]$

Output: Tableau $[\mathcal{U}]$, vector $[B]$ with the basic variables and a flag $\in \{\text{Opt}, \text{Unb}, \text{Unf}\}$ indicating the state of the LP problem (Opt: problem solved, Unb: it is unbounded, Unf: no feasible solution)

1 **while True do**
2 $\quad([V], a_v) \leftarrow$ *GetPivCol*$([\mathcal{T}(1, 1, \ldots, n)])$
3 $\quad$**if** $a_v = 0$ **then break**
4 $\quad[C] \leftarrow$ *SecReadCol*$(\mathcal{T}, [V])$
5 $\quad([W], a_v) \leftarrow$ *GetPivRow*$([\mathcal{T}(1, \ldots, m, n+1)], [C])$
6 $\quad$**if** $a_v = 0$ **then** flag $\leftarrow$ Unf **break**
7 $\quad[R] \leftarrow$ *SecReadRow*$([\mathcal{T}], [W])$
8 $\quad[p] \leftarrow$ *SecRead*$([R], [V])$
9 $\quad[\mathcal{T}] \leftarrow$ *UpdateTab*$([\mathcal{T}], [C], [R], [V], [W], [p])$
10 $\quad[\mathcal{U}] \leftarrow$ *UpdateTab*$([\mathcal{U}], [C], [R], [V], [W], [p])$
11 $\quad([B], [N_b]) \leftarrow$ *UpdateVar*$([B], [N_b], [V], [W])$
12 **if** flag $=$ Unf **then return** $([\mathcal{U}], [B],$ flag$)$
13 $([\mathcal{U}], [B],$ flag$) \leftarrow$ *Iteration*$([\mathcal{U}], [B], [N_b])$
14 **return** $([\mathcal{U}], [B],$ flag$)$

Lastly, *TwoPhaseSimplex* (below as Protocol 4) calls protocols *InitVar*, *GetSolution* from [22] and protocol *TwoPhaseIteration* to perform the entire two-phase simplex to obtain the optimal value function $[V^\star(s)]$. The first phase will locate some feasible basic solution (the artificial problem (3) condensed in the tableau $\mathcal{T}$), and the second phase will find the optimal solution to (2) (in the tableau $\mathcal{U}$).

Protocol 4 *TwoPhaseSimplex*$([\mathcal{T}], [\mathcal{U}], (m, n))$

Inputs: Shared tableaux $[\mathcal{T}]$, $[\mathcal{U}]$ and (m, n) the dimensions M

Output: Shared optimal value function $[V(s)]$

1 $([B], [N_b]) \leftarrow$ *InitVar*(m, n)
2 $([\mathcal{T}], [B],$ flag$) \leftarrow$ *TwoPhaseIteration*$([\mathcal{T}], [\mathcal{U}], [B], [N_b])$
3 **if** flag $=$ Unf **then return** Unf
4 **if** flag $=$ Unb **then return** Unb
5 $[X] \leftarrow$ *GetSolution*$([\mathcal{U}(\cdot, n+1)], [N])$
6 $[V^\star(s)] \leftarrow [X(1, \ldots, |S|)]$
7 **return** $[V^\star(s)]$

C. Obtaining the optimal policy

For an MDP, a deterministic optimal policy is a mapping $\pi^\star : S \to A$ that maximizes each state's value. In our case, we are interested in obtaining the optimal policy $\pi^\star(s)$ from an optimum state value function $V^\star(s)$, both are related by

$$\pi^\star(s) = \arg\max_{a \in A} \sum_{s' \in S} \underbrace{T(s, a, s')}_{\text{Owned by B.O.B.}} \; (\underbrace{R(s, a) + \gamma}_{\text{Owned by Alice}} \, V^\star(s')). \quad (5)$$

ValueToPolicy obtains the policy using the *ArgMax* function [30] where *ArgMax*$([Q(s, \cdot)]) = \arg\max_{a \in A}[Q(s, a)]$.

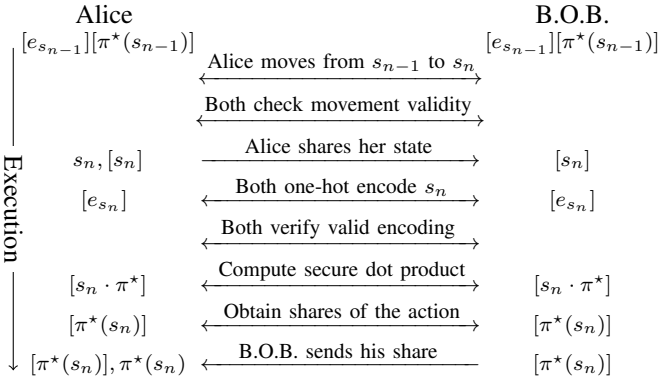


Fig. 2: Policy Execution. An interactive scheme where, after the iteration, Alice and B.O.B. will possess shares $[e_{s_n}]$ and $[\pi^\star(s_n)]$. When both check valid encoding it means to make sure that e_{s_n} is indeed a canonical vector.

Protocol 5 *ValueToPolicy*$([V^\star(s)], [T], [R], [\gamma])$

Inputs: Value function $[V^\star(s)]$

Output: Shared policy $[\pi^\star(s)]$ associated with $[V^\star(s)]$

1 **foreach** $(s, a) \in S \times A$
2 $\quad[Q(s, a)] \leftarrow \sum_{s' \in S} [T(s, a, s')]([R(s', a)] + [\gamma][V^\star(s')])$
3 **foreach** $s \in S$
4 $\quad[\pi^\star(s)] \leftarrow$ *ArgMax*$([Q(s, \cdot)])$
5 **return** $[\pi^\star(s)]$

D. Policy execution

After completing the preceding steps, an optimal policy has been computed, but neither party can query it alone.

1) Protecting Alice: During the execution phase, each time she performs an action, Alice has to retrieve the policy value at her current state. To reconstruct it, she needs to ask B.O.B. about his part of the policy *in her state*. But she never wishes to reveal to him which state she occupies or has occupied. To guarantee that Alice's state will not be disclosed, she creates secure shares of her position using the *ShamirShare* primitive, as a one-hot vector representation, i.e., state s_a is encoded as the canonical vector e_{s_a}. Then, with a secure dot product, the parties compute $e_{s_a} \cdot (\pi^\star(s_1), \ldots, \pi^\star(s_{|S|})) = \pi^\star(s_a)$. Finally, B.O.B. sends his share of the result to Alice, who can reconstruct the value and execute the action. Alice's state is not disclosed to B.O.B. Even if B.O.B. compares his shared part of the policy with his shared part of $\pi^\star(s_a)$, he gains no information as his shared part of $\pi^\star(s_a)$ needs Alice's half of $\pi^\star(s_a)$, and her half of vector e_{s_a}. This argument relies on the fact that one agent cannot manipulate their shares to change the shared value as a change in shares will result in a random change to the shared value (refer to protocol *ShamirShare*); therefore no agent can manipulate this computation individually.

2) Protecting B.O.B.: On the other hand, B.O.B. wants to ensure that Alice follows the optimal policy, seeking to protect his robot from misuse. To do so, without disclosing the transition function or learning Alice's state, suppose we are at the n^{th} step during the execution of the policy. At this juncture, Alice will have shared her state s_{n-1} through the shared canonical vector $[e_{s_{n-1}}]$ when she queried the action $[\pi^\star(s_{n-1})]$. If that action *was* executed as it should have been, it leads to her next state. To ascertain veracity, we

determine whether the state Alice's claims to be her current state, s_n, could indeed be achieved from prior state s_{n-1} after execution of action $\pi^\star(s_{n-1})$. Thus, we wish to verify $T(s_{n-1}, \pi^\star(s_{n-1}), s_n) > 0$ securely. Observe that to extract a specific element $[A]_{ij}$ from $A \in \mathbb{R}^{n\times m}$, we perform the matrix product $e_i^\top A e_j$. Define matrix $T_a \in \mathbb{R}^{|S|\times|S|}$ where $a \in A$ and $[T_a]_{ss'} = [T(s, a, s')]$ for $(s, s') \in S \times S$, and, finally, let $[e_{s_{n-1}}]$, $[e_{s_n}] \in \mathbb{R}^{|S|}$ and $[e_{\pi^\star(s_{n-1})}] \in \mathbb{R}^{|A|}$ be, respectively, one-hot encoded vectors of the Alice's last state, current state, and the action given by the policy at Alice's last state. Then, compute $[T(s_{n-1}, \pi^\star(s_{n-1}), s_n)] > 0$ via

$$\left[e^\top_{s_{n-1}}\right]\left(\sum_{a\in A}\left[e^{(a)}_{\pi^\star(s_{n-1})}\right][T_a]\right)\left[e_{s_n}\right] > 0, \tag{6}$$

where $\left[e^{(a)}_{\pi^\star(s_{n-1})}\right]$ is the a^{th} component of $\left[e_{\pi^\star(s_{n-1})}\right]$. This value certifies the consistency of Alice's movement, which we call *checking movement validity*.

The steps of the interactive procedure just described have been summarized in Fig. 2. It allows B.O.B. to verify that the sequence of queries Alice generates is consistent with the execution of the policy, doing so without disclosing her states at each point or revealing the transition dynamics. Alice needn't know the transition dynamics but need only act faithfully in a world with those dynamics. Finally, to prevent excessive policy leakage B.O.B. will limit the number of policy queries, for instance to $\frac{3}{2}\sqrt{|S|}$ ([33] has a similar quantification). For queries exceeding this number, B.O.B. will conclude that the policy is being probed and will, thus, cease communication with Alice.

E. Complexity

1) Computational Complexity: The computational cost of the planning procedure, before executing the the policy is dominated by the protocols *TwoPhaseIteration* and *TwoPhaseSimplex*, which are the simplex method. Although the algorithm has an exponential time worst-case, several works report that those cases are rare and even minuscule modifications to the constraints can make them polynomial-time solvable, e.g., cf. [34]. Moreover, [31] and [34] agree that algorithms solving LP problems with n variables and m constraints, including simplex, have average complexity $\mathcal{O}(nm^2) = \mathcal{O}(|A|^2|S|^3)$. During the execution phase, we need to calculate one dot product for extracting the action from the policy, $|A|$ matrix sums, and $|S|+2$ dot products to evaluate the movement validity, for a total cost of $\mathcal{O}(|A||S|^2)$ per iteration.

2) Communication Complexity: The usual bottleneck for interactive protocols is the communication between agents. To be useful, a metric for complexity should consider the data transmissions: one measure is the number of invocations of primitives that require information transmission; another is round complexity, which tallies the number of invocations done sequentially and evaluates the delay between messages. The simplex procedure, during planning, dominates the entire procedure. Extrapolating the analysis of [22], protocols *TwoPhaseIteration* and *TwoPhaseSimplex* are performed in $\mathcal{O}(m\log(nm)) = \mathcal{O}(|A||S|\log(|A||S|^2))$ rounds and $\mathcal{O}(nm^2) = \mathcal{O}(|A|^2|S|^3)$ invocations in the average case.

During policy execution, each agent can perform matrix additions locally (so no triggering of a round or invocation is needed). However, $|A||S|^2$ single products and $|S| + 2$ dot products do have to be computed as shown in Fig.2. An efficient dot product implementation requires only one round and one invocation each [22]. In total, execution steps require $\mathcal{O}(|A||S|^2)$ rounds and invocations per query.

F. Correctness and Convergence

The protocols are sequential so termination follows from the termination of the schemes employed. Protocols *MDP-ToLP* and *ValueToPolicy* directly apply the definitions to construct the LP problem and extract the optimal policy from the optimal value function—these procedures converge in a finite number of steps. On the other hand, *TwoPhaseIteration* and *TwoPhaseSimplex* perform the simplex method; their convergence depends upon Dantzig's pivot rule [31], which is guaranteed to converge whether it finds an optimal solution or determines no solution exists (either being unbounded or infeasible). Finally, the execution phase complexity is bounded by the the number of queries Alice is permitted.

G. Security Analysis

The SSS framework provides strong guarantees: perfect or statistical security [30], [22]. We use secure blocks in order to take advantage of Canetti's Universal Composition framework [35], which ensures that composing different secure primitives yields a more complex protocol but maintains its security characteristics.

VI. Experimental Results

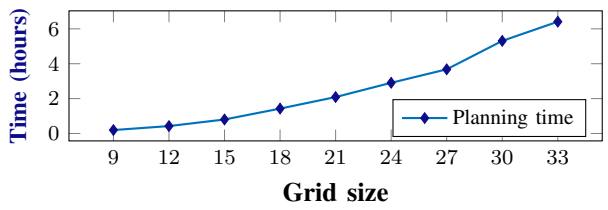
We tested our approach on simulation experiments and using physical platforms.

A. Simulation Experiments

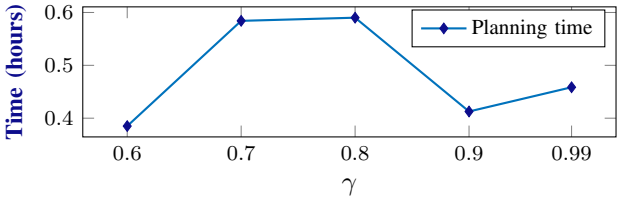
We conducted our simulation experiments using a single machine with an Intel i7 3.60 GHz CPU and 16 GB RAM, running Ubuntu 18.04 LTS, Python 3, and Robot Operating System (ROS). Each run has random starting and goal positions, and a random *obstacle region*. We used the Secure Multiparty Computation framework (MPyC) to enable both parties to perform computations on secret-shared values. Both Alice and B.O.B. interact by exchanging messages via peer-to-peer connections, each with a virtual network interface used for MPyC configured for asynchronous operation, as detailed in [36].

Each experimental run consists of two steps. First, both agents solve the MDP and construct the policy shares. Secondly, Alice queries the policy using the procedure mentioned in Section V-D.1, while B.O.B. checks movement validity through the scheme in Section V-D.2.

We studied the time needed to compute the policy in worlds of increasing size and varying the discount factor γ. During this process, we fixed the number of bits needed for the computation's secure (secret-shared) fixed-point numerical representation to 200 bits. On one hand, we increased



(a) Planning time and precision vs MDP size, the latter expressed in grids ranging in size as $3 \times y$ with $y \in \{3, 4, \ldots, 11\}$, and $\gamma = 0.99$.



(b) Planning time and precision vs discount factor γ within $3 \times 4 = 12$ grid.

Fig. 3: Planning time and needed numerical precision plotted as a problem instance size (top) and MDP discount factor (bottom) vary while fixing the number of bits to 200.

the world size to range across $3 \times \{3, 4, \ldots, 11\}$ and kept $\gamma = 0.99$ fixed. The impact on running time is shown in the solid blue curve in the plot in Fig. 3a. We observed that the running time increases at a polynomial rate. On the other hand, we also examined how the discount factor plays a role. Fixing the 3×4 grid, we varied γ between 0.6 and 0.99, which yields the result in Fig 3b. Unlike value iteration techniques (as we mentioned above), running time is usually nearly constant regardless of γ for LP-based approaches.

B. Mobile Robot Experiments

We conducted physical experiments with two Raspberry Pi-embedded computers: one situated on a mobile robot (Alice), the other being statically placed (for B.O.B.). The robot is an iRobot Create device running Robot Operating System (ROS), with the *AprilTag* visual fiducial system [37] used for localization purposes. Both parties communicate through a wireless router using ROS messages. In the physical world, we considered specific obstacles positioned at particular locations. When the robot (Alice) encounters an obstacle, it automatically returns to its previous position. The communication between robots is facilitated through ROS, inheriting the associated limitations in communication capabilities. A video of the physical experiments can be found at `https://youtu.be/QvtoHtYEkhQ?si=1mX7L-_2fxm2hOkT`

Fig. 4 shows a policy computed and executed to navigate a 3×3 grid world. Initially, Alice and B.O.B. start by computing the MDP's policy. Then, Alice queries and makes her first move according to the policy. B.O.B. validates every query, helping to ensure there is no misuse of the robot.

A more interesting execution is shown in Fig. 5, wherein Alice takes the first action as before, but (modeling transition uncertainty) has been shifted to move to the diagonally adjacent grid cell. B.O.B. validates the move's outcome as permissible under the probabilistic transition dynamics, and then Alice queries again en route the final state.

In contrast, if Alice breaches the policy by moving to an

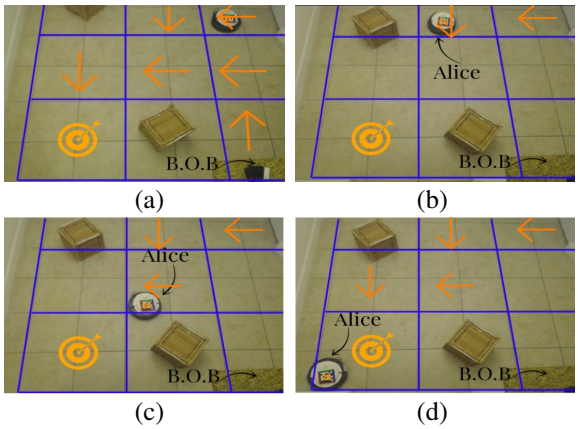


Fig. 4: Navigation from the initial to the goal position: (a) Alice and B.O.B. begin the planning step and, after 4.36 min, the optimal policy is computed; (b) Alice makes her first query and moves (the action is to move West) in 21 sec; (c) At 22 sec, Alice encoded and shared her action and state with B.O.B. who verified the query to validate that Alice had executed the correct action; (d) Following the policy successfully, Alice arrives at the goal.

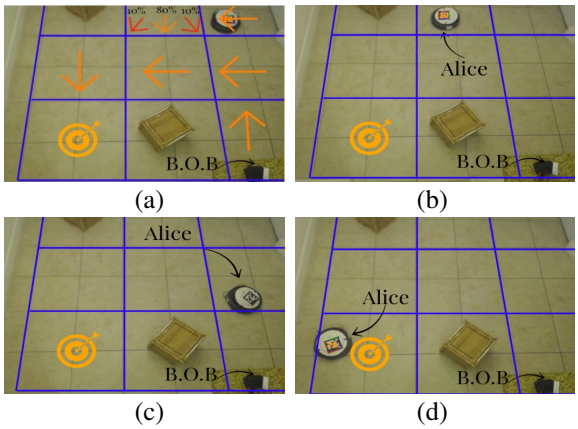


Fig. 5: Navigation that arrives in a diagonal cell: (a) Alice and B.O.B. begin solving the MDP; (b) Alice queries and the action is to move West; (c) her action outcome has been a diagonal motion to a state that is South-East; B.O.B. determines that Alice could have arrived there under the transition model while executing the policy honestly; (d) Alice reaches the goal.

incompatible state or querying the policy from such a state, B.O.B. terminates communication (Fig. 6). Milder options could be triggering a request for Alice to reaffirm the EULA, or by throttling the user by delaying policy execution.

C. Scalability

The experiments so far have shown the approach to be feasible for small-sized scenarios. Some further innovation appears to be needed in order to address larger MDPs, as the required computation rises dramatically. Our approach for large state spaces, following [38], is to express the value function as a sum of a set of basis functions $h_i(s) : S \to \mathbb{R}$:

$$V(s) = \sum_{i=1}^{k} w_i h_i(s), \text{ where } k \ll |S|. \tag{7}$$

Problem (1) then becomes

$$\begin{aligned} \min_{w_1,\ldots,w_k} \quad & \sum_{i=1}^{k} w_i \sum_{s \in S} \mu(s) h_i(s) \\ \text{subject to} \quad & \sum_{i=1}^{k} w_i h_i(s) \geq R(s,a) + \gamma \sum_{i=1}^{k} w_i \sum_{s' \in S} T(s,a,s') h_i(s'), \\ & w_i \geq 0, \quad \text{for } s \in S,\ a \in A \text{ and } i = 1, \ldots, k \end{aligned} \tag{8}$$

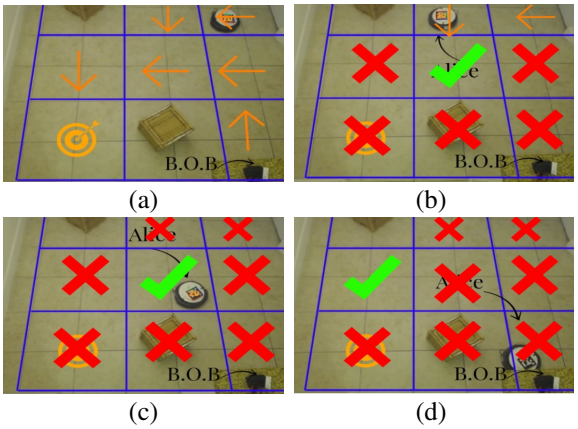


Fig. 6: Alice's attempted misuse of the policy: (a) Alice and B.O.B. start computing the MDP; (b) Alice queries and follows the policy in her first move (West); (c) Alice again queries, but now misbehaves, executing a different action causing her to be in a state with zero probability, shown in (d). Here Alice remains because B.O.B. ceases to engage in the protocol.

for some bounded k. This technique reduces computational and communication complexities by $\mathcal{O}(|S|)$. Still, this does not emend the requirement of $|S||A|$ constraints in the LP, which remains challenging when the number of states is not small. To surmount this difficulty, we further employ the *constraint sampling scheme* championed by [39]: when the number of variables is much smaller than the number of constraints ($k \ll |S||A|$), this suggests that there are likely many redundant constraints. The idea is that sampling a reduced number of constraints can produce solutions with a high probability of satisfying all but a small number of constraints. The hope is that omitted constraints are redundant, so will not significantly impact the solution. Let

$$m = \frac{4}{\epsilon}\left(k \ln\left(\frac{12}{\epsilon}\right) + \ln\left(\frac{1}{\delta}\right)\right). \tag{9}$$

Then taking m to be the number of samples in the constraint sampling scheme, one determines that the probability that a reduced set of that size will be enough to satisfy almost every constraint. Specifically, with a probability of at least $1-\delta$, sampling m constraints uniformly and independently at random will lead to at most $\epsilon(|S||A|)$ unsatisfied constraints. Importantly, the value of m does not depend on the size of the state or action spaces — hence the scheme reduces the computational and communication complexity by $\mathcal{O}(|S||A|)$, at the cost of a small error margin of ϵ.

In attempting to scale to larger problem instances, both techniques were employed together. The value function was assumed to be a plane, i.e., $V(s) = h_1(s)w_1 + w_2h_2(s) + w_3h_3(s) = V(x,y) = w_1 + w_2x + w_3y$ where s, interpreted as the pair (x,y), indicates a grid position. Taking $\epsilon = 0.22$, $\delta = 0.1$, and $k = 3$ expression (9) gives $m \approx 256$. We conducted simulation experiments and measured the running time for larger grids to assess the overall effectiveness. Fig. 7 shows that the reduction in running time is dramatic, at the cost of having assumed the value function's specific form.

To appraise the loss in accuracy this incurs, we used two metrics from [39]: L_p norm and Hamming distance. We applied these metrics to measure the distance between the real optimal value function $V^\star(s)$ and the value function obtained by solving (8), which we denote as $\hat{V}^\star(s)$. The L_p norm weighted by the vector $c \geq 0$ is defined as

$$||x||_{p,c} = \Big(\sum_{i=1}^{n} c_i |x_i|^p\Big)^{1/p}. \tag{10}$$

We used $||\cdot||_{p,c}$ to portray the distance between $V^\star(s)$ and $\hat{V}^\star(s)$, taking $p = 1$ and c as μ, the initial distribution.

To compare the quality of the policies produced by these value functions, denoted $\pi^\star(s)$ and $\hat{\pi}^\star(s)$ respectively, we counted the number of states where $\pi^\star(s) \neq \hat{\pi}^\star(s)$ and weighted this by the vector μ. Known as the Hamming distance, it corresponds also to the norm $||\cdot||_{0,\mu}$.

To aid in interpreting the results, the distances are reported as normalized, given by ratios of relative L_1 distance and relative Hamming distances:

$$\frac{||V^\star(s) - \hat{V}^\star(s)||_{1,\mu}}{||V^\star(s)||_{1,\mu}}, \quad \frac{||\pi^\star(s) - \hat{\pi}^\star(s)||_{0,\mu}}{|S|}. \tag{11}$$

The ratios give an idea of the accuracy relative to the optimal functions, independent of the grid size. In Fig. 7, see how, though the relative L_1 increases, the effect on the resulting policy is minimal (i.e., the relative Hamming distance remains bounded). The approximated value function, thus, manages to represent the overall behavior of the MDP problem even with few basis functions (qualitatively, this can be seen more directly within the visualization in Fig 8).

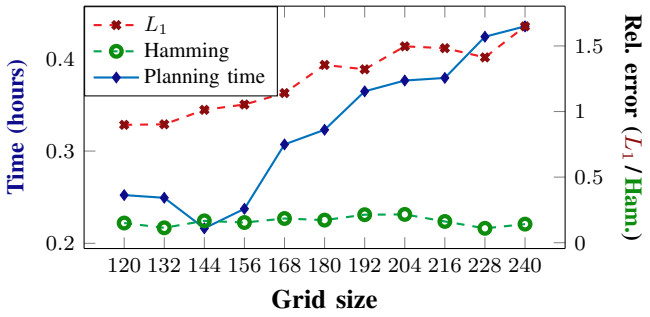


Fig. 7: Planning time when employing features to improve scalability: basis functions and constraint sampling. This also allows the numerical precision to be kept constant at 80 bits. The relative error in value functions and policies, expressed via L_1 and Hamming metrics, share the rightmost axis.

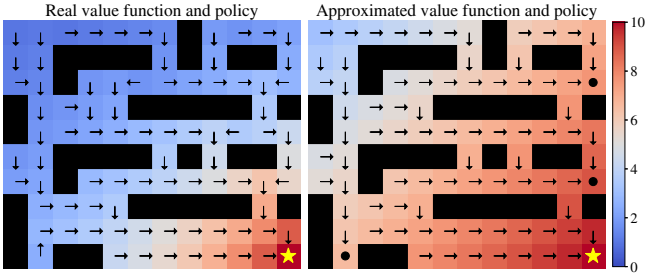


Fig. 8: Visualizations of $V^\star(s)$ and $\hat{V}^\star(s)$, plotted as heatmaps; policies $\pi^\star(s)$ and $\hat{\pi}^\star(s)$ show their actions as arrows (and dots for the stay action). The goal point is marked as the yellow star.

VII. Conclusions and Future Work

This paper explores how two parties can jointly compute and execute an optimal policy for an MDP while protecting their inputs. It focuses on Robots-as-a-Service scenarios

where privacy-conscious customers consider renting a robot. The study demonstrates that planning and execution are feasible but only for small-scale problems. To address this limitation, the paper proposes methods for approximating solutions, significantly increasing the scale of feasible problem instances. Being aware of the algorithm's execution time can disrupt methods like policy or value iteration. The simplex algorithm maintains a steady average running time. To introduce variability, B.O.B. and Alice can add random positive numbers in the bit representation (see Section VI-A).

An interesting aspect of the protocol we have described is that while one party knows (in the sense of owning a mathematical description of) the transition dynamics, the other party represents the executor which 'experiences' those dynamics. When there is mild noise, the execution will mostly follow the nominal trajectory, and so any queries outside of this path (if that agent is curious and, hence, deviates) will be easily found out; thus, the physical transitions somehow affect what can be protected. Future work should explore the ramifications of the physical system's noise obscuring actual utilization, along with the prospects of non-optimal policies (and as well as other means) to constrain how much experience the robot is permitted, e.g., to limit data gathered on its own dynamics.

REFERENCES

[1] G. Nichols, "Another one bites the dust: Why consumer robotics companies keep folding." ZDNET/Innovation, 1 May 2019. https://www.zdnet.com/article/another-one-bites-the-dust-why-consumer-robotics-companies-keep-folding/, Last access: 5 Sept. 2022.

[2] L. Vaas, "Privacy dust-up as Roomba maker mulls selling maps of users' homes," *Naked Security*, July 2017. https://nakedsecurity.sophos.com/2017/07/26/privacy-dust-up-as-roomba-maker-mulls-selling-maps-of-users-homes/, Last access: 5 Sept. 2022.

[3] D. Evans, V. Kolesnikov, M. Rosulek, *et al.*, "A pragmatic introduction to secure multi-party computation," *Foundations and Trends® in Privacy and Security*, vol. 2, no. 2–3, pp. 70–246, 2018.

[4] R. Cramer, I. B. Damgård, and J. B. Nielsen, *Secure Multiparty Computation and Secret Sharing*. Cambridge University Press, 2015.

[5] N. DeMarinis, S. Tellex, V. P. Kemerlis, G. Konidaris, and R. Fonseca, "Scanning the Internet for ROS: A View of Security in Robotics Research," in *Proc. IEEE Conf. on Robotics and Automation*, pp. 8514–8521, 2019.

[6] S. Eick and A. I. Antón, "Enhancing privacy in robotics via judicious sensor selection," in *Proc. IEEE Conf. on Robotics and Automation*, pp. 7156–7165, 2020.

[7] K. Tjell, N. Schlüter, P. Binfet, and M. S. Darup, "Secure learning-based mpc via garbled circuit," in *Proc IEEE Conf. on Decision and Control*, pp. 4907–4914, 2021.

[8] J. Park, D. S. Kim, and H. Lim, "Privacy-preserving reinforcement learning using homomorphic encryption in cloud computing infrastructures," *IEEE Access*, vol. 8, pp. 203564–203579, 2020.

[9] K. Tjell, N. Schlüter, P. Binfet, and M. S. Darup, "Secure learning-based MPC via garbled circuit," in *Proc IEEE Conf. on Decision and Control*, pp. 4907–4914, 2021.

[10] S. Han and G. J. Pappas, "Privacy in control and dynamical systems," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 1, no. 1, pp. 309–332, 2018.

[11] V. Pacelli and A. Majumdar, "Robust control under uncertainty via bounded rationality and differential privacy," in *Proc. IEEE Conf. on Robotics and Automation*, pp. 3467–3474, 2022.

[12] S. Savazzi, M. Nicoli, M. Bennis, S. Kianoush, and L. Barbieri, "Opportunities of federated learning in connected, cooperative, and automated industrial systems," *IEEE Communications Magazine*, vol. 59, no. 2, pp. 16–21, 2021.

[13] J. M. O'Kane and D. A. Shell, "Automatic design of discreet discrete filters," in *Proc. IEEE Conf. on Robotics and Automation*, pp. 353–360, 2015.

[14] Y. Zhang and D. A. Shell, "Complete characterization of a class of privacy-preserving tracking problems," *International Journal of Robotics Research*, vol. 38, no. 2–3, pp. 299–315, 2019.

[15] Y. Zhang, D. A. Shell, and J. M. O'Kane, "Finding Plans Subject to Stipulations on What Information They Divulge," in *Proc. Workshop on Algorithmic Foundations of Robotics (WAFR)*, pp. 106–124, 2018.

[16] A. Shamir, "How to share a secret," *Communications of the ACM*, vol. 22, no. 11, pp. 612–613, 1979.

[17] P. Chen, "Secure multiparty computation for privacy preserving data mining," Master's thesis, Eindhoven University of Technology, 2012.

[18] S. de Hoogh, B. Schoenmakers, P. Chen, and H. op den Akker, "Practical secure decision tree learning in a teletreatment application," in *Int'l Conf. on Financial Cryptography and Data Security*, pp. 179–194, 2014.

[19] F. Blom, N. J. Bouman, B. Schoenmakers, and N. de Vreede, "Efficient secure ridge regression from randomized gaussian elimination," in *Int'l Symp. on Cyber Security Cryptography and Machine Learning*, pp. 301–316, 2021.

[20] M. Abspoel, N. J. Bouman, B. Schoenmakers, and N. de Vreede, "Fast secure comparison for medium-sized integers and its application in binarized neural networks," in *Topics in Cryptology (CT-RSA)*, pp. 453–472, 2019.

[21] T. Toft, "Solving linear programs using multiparty computation," in *Financial Cryptography and Data Security* (R. Dingledine and P. Golle, eds.), pp. 90–107, Springer, 2009.

[22] O. Catrina and S. de Hoogh, "Secure Multiparty Linear Programming Using Fixed-Point Arithmetic," in *Computer Security (ESORICS)*, pp. 134–150, 2010.

[23] S. de Hoogh, *Design of large scale applications of secure multiparty computation: secure linear programming*. PhD thesis, Technische Universiteit Eindhoven, Mathematics and Computer Science, 2012.

[24] P. Mainali and C. Shepherd, "Privacy-enhancing fall detection from remote sensor data using multi-party computation," in *Proc. of Intl. Conf. on Availability, Reliability and Security*, pp. 1–10, 2019.

[25] D. Moser, *Modern Attacker Models and Countermeasures in Wireless Communication Systems–The Case of Air Traffic Communication*. PhD thesis, ETH Zurich, 2021.

[26] L. Li, A. Bayuelo, L. Bobadilla, T. Alam, and D. A. Shell, "Coordinated multi-robot planning while preserving individual privacy," in *Proc. IEEE Conf. on Robotics and Automation*, pp. 2188–2194, 2019.

[27] M. Alsayegh, P. Vanegas, A. A. R. Newaz, L. Bobadilla, and D. A. Shell, "Privacy-preserving multi-robot task allocation via secure multi-party computation," in *Proc. of European Control Conference*, pp. 1274–1281, 2022.

[28] O. L. Mangasarian, "Privacy-preserving linear programming," *Optimization Letters*, vol. 5, pp. 165–172, 2011.

[29] O. Goldreich, *Foundations of cryptography: volume 2, basic applications*. Cambridge University Press, 2009.

[30] B. Schoenmakers, "MPyC: Secure multiparty computation in Python." https://github.com/lschoe/mpyc, May 2018.

[31] M. S. Bazaraa, J. J. Jarvis, and H. D. Sherali, *Linear programming and network flows*. John Wiley & Sons, 4th ed., 2008.

[32] M. L. Puterman, *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

[33] H. H. Zhuo, W. Feng, Y. Lin, Q. Xu, and Q. Yang, "Federated deep reinforcement learning," 2020. arXiv: 1901.08277.

[34] D. A. Spielman and S.-H. Teng, "Smoothed analysis of algorithms: Why the simplex algorithm usually takes polynomial time," *Journal of the ACM*, vol. 51, p. 385–463, May 2004.

[35] R. Canetti, "Universally composable security: A new paradigm for cryptographic protocols," in *Proc. IEEE Symp. on Foundations of Computer Science*, pp. 136–145, 2001.

[36] I. Damgård, M. Geisler, M. Krøigaard, and J. B. Nielsen, "Asynchronous multiparty computation: Theory and implementation," in *Public Key Cryptography (PKC)*, Springer, 2009.

[37] E. Olson, "Apriltag: A robust and flexible visual fiducial system," in *Proc. IEEE Conf. on Robotics and Automation*, pp. 3400–3407, 2011.

[38] C. Guestrin, D. Koller, and R. Parr, "Multiagent planning with factored mdps," in *Advances in Neural Inf. Processing Systems*, vol. 14, 2001.

[39] D. P. De Farias and B. Van Roy, "On constraint sampling in the linear programming approach to approximate dynamic programming," *Mathematics of operations research*, vol. 29, no. 3, pp. 462–478, 2004.