Privacy-Preserving Reinforcement Learning

Jun Sakuma JUN@FE.DIS.TITECH.AC.JP
Shigenobu Kobayashi KOBAYASI@DIS.TITECH.AC.JP
Tokyo Institute of Technology, 4259 Nagatsuta-cho, Midoriku, Yokohama, 226-8502, Japan

Rebecca N. Wright REBECCA.WRIGHT@RUTGERS.EDU
Rutgers University, 96 Frelinghuysen Road, Piscataway, NJ 08854, USA

Abstract

We consider the problem of distributed reinforcement learning (DRL) from private perceptions. In our setting, agents' perceptions, such as states, rewards, and actions, are not only distributed but also should be kept private. Conventional DRL algorithms can handle multiple agents, but do not necessarily guarantee privacy preservation and may not guarantee optimality. In this work, we design cryptographic solutions that achieve optimal policies without requiring the agents to share their private information.

1. Introduction

With the rapid growth of computer networks and networked computing, a large amount of information is being sensed and gathered by distributed agents physically or virtually. Distributed reinforcement learning (DRL) has been studied as an approach to learn a control policy thorough interactions between distributed agents and environments—for example, sensor networks and mobile robots. DRL algorithms, such as the distributed value function approach (Schneider et al., 1999) and the policy gradient approach (Moallemi & Roy, 2004), typically seek to satisfy two types of physical constraints. One is constraints on communication, such as an unstable network environment or limited communication channels. The other is memory constraints to manage the huge state/action space. Therefore, the main emphasis of DRL has been to learn good, but sub-optimal, policies with minimal or limited sharing of agents' perceptions.

Appearing in *Proceedings of the 25th International Conference on Machine Learning*, Helsinki, Finland, 2008. Copyright 2008 by the author(s)/owner(s).

In this paper, we consider the privacy of agents' perceptions in DRL. Specifically, we provide solutions for privacy-preserving reinforcement learning (PPRL), in which agents' perceptions, such as states, rewards, and actions, are not only distributed but are desired to be kept private. Consider two example scenarios:

Optimized Marketing (Abe et al., 2004): Consider the modeling of the customer's purchase behavior as a Markov Decision Process (MDP). The goal is to obtain the optimal catalog mailing strategy which maximizes the long-term profit. Timestamped histories of customer status and mailing records are used as state variables. Their purchase patterns are used as actions. Value functions are learned from these records to learn the optimal policy. If these histories are managed separately by two or more enterprises, they may not want to share their histories for privacy reasons (for example, in keeping with privacy promises made to their customers), but might still like to learn a value function from their joint data in order that they can all maximize their profits.

Load Balancing (Cogill et al., 2006): Consider a load balancing among competing factories. Each factory wants to accept customer jobs, but in order to maximize its own profit, may need to redirect jobs when heavily loaded. Each factory can observe its own backlog, but factories do not want to share their backlog information with each other for business reasons, but they would still like to make optimal decisions.

Privacy constraints prevent the data from being combined in a single location where centralized reinforcement algorithms (CRL) could be applied. Although DRL algorithms work in a distributed setting, they are designed to limit the total amount of data sent between agents, but do not necessarily do so in a way that guarantees privacy preservation. Additionally, DRL often sacrifices optimality in order to learn with low communication. In contrast, we propose solutions

that employ cryptographic techniques to achieve optimal policies (as would be learned if all the information were combined into a centralized reinforcement learning (CRL) problem) while also explicitly protecting the agents' private information. We describe solutions both for data that is "partitioned-by-time" (as in the optimized marketing example) and "partitioned-by-observation" (as in the load balancing example).

Related Work. Private distributed protocols have been considered extensively for data mining, pioneered by Lindell and Pinkas (Lindell & Pinkas, 2002), who presented a privacy-preserving data-mining algorithm for ID3 decision-tree learning. Private distributed protocols have also been proposed for other data mining and machine learning problems, including k-means clustering (Jagannathan & Wright, 2005; Sakuma & Kobayashi, 2008), support vector machines (Yu et al., 2006), boosting (Gambs et al., 2007), and belief propagation (Kearns et al., 2007).

Agent privacy in reinforcement learning has been previously considered by Zhang and Makedon (Zhang & Makedon, 2005). Their solution uses a form of average reward reinforcement learning that does not necessarily guarantee an optimal solution; further, their solution only applies partitioning by time. In contrast, our solutions guarantee optimality under appropriate conditions and we provide solutions both when the data is partitioned by time and by observation.

In principle, private distributed computations such as these can be carried out using secure function evaluation (SFE) (Yao, 1986; Goldreich, 2004), which is a general and well studied methodology for evaluating any function privately. However, although asymptotically polynomially bounded, these computations can be too inefficient for practical use, particular when the input size is large. For the reinforcement learning algorithms we address, we make use of existing SFE solutions for small portions of our computation in order as part of a more efficient overall solution.

Our Contribution. We introduce the concepts of partitioning by time and partitioning by observation in distributed reinforcement learning (Section 2). We show privacy-preserving solutions for SARSA learning algorithms with random action selection for both kinds of partitioning (Section 4). Additionally, these algorithms are expanded to Q-learning with greedy or ϵ-greedy action selection (Section 5). We provide experimental results in Section 6.

Table 1 provides a qualitative comparison of variants of reinforcement learning in terms of efficiency, learning accuracy, and privacy loss. We compare five approaches: CRL, DRL, independent distributed reinforcement learning (IDRL, explained below), SFE, and our privacy-preserving reinforcement learning solutions (PPRL). In CRL, all the agents send their perceptions to a designated agent, and then a centralized reinforcement is applied. In this case, the optimal convergence of value functions is theoretically guaranteed when the dynamics of environments follow a discrete MDP; however, privacy is not provided, as all the data must be shared.

	comp.	comm.	accuracy	privacy
CRL	good	good	good	none
DRL	good	good	medium	imperfect
IDRL	good	good	bad	perfect
PPRL	**medium**	**medium**	**good**	**perfect**
SFE	bad	bad	good	perfect

Table 1. Comparison of different approaches

On the opposite end of the spectrum, in IDRL (independent DRL), each agent independently applies CRL only using its own local information; no information is shared. In this case, privacy is completely preserved, but the learning results will be different and independent. In particular, accuracy will be unacceptable if the agents have incomplete but important perceptions about the environment. DRL can be viewed as an intermediate approach between CRL and IDRL, in that the parties share only some information and accordingly reap only some gains in accuracy.

The table also includes the direct use of general SFE and our approach of PPRL. Both PPRL and SFE obtain good privacy and good accuracy. Although our solution incurs a significant cost (as compared to CRL, IDRL, and DRL) in computation and communication to obtain this, it does so with significantly improved computational efficiency over SFE. We provide a more detailed comparison of the privacy, accuracy, and efficiency of our approach and other possible approaches along with our experimental results in Section 6.

2. Preliminaries

2.1. Reinforcement Learning and MDP

Let S be a finite *state set* and A be a finite *action set.* A *policy* π is a mapping from state/action pair (s, a) to the probability $\pi(s, a)$ with which action a is taken at state s. At time step t, we denote by s_t, a_t, and r_t, the state, action, and *reward* at time t, respectively.

A *Q-function* is the expected return $Q^\pi(s, a) = E_\pi \left\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s, a_t = a \right\}$, where γ is a discount factor ($0 \leq \gamma < 1$). The goal is to learn the optimal policy π maximizing the Q-function: $Q^*(s, a) =$

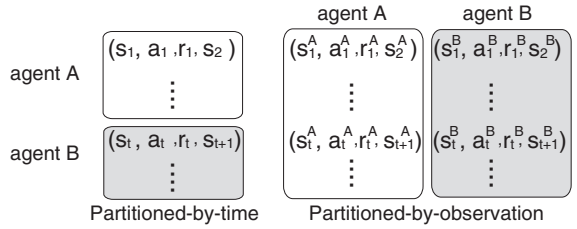

Figure 1. Partitioning model in the two-agent case

$\max_\pi Q(s,a)$ for all (s,a). In SARSA learning, Q-values are updated at each step as:

$$\begin{aligned} \Delta Q(s_t,a_t) &\leftarrow \alpha(r_t+\gamma Q(s_{t+1},a_{t+1})-Q(s_t,a_t)), \\ Q(s_t,a_t) &\leftarrow \Delta Q(s_t,a_t)+Q(s_t,a_t), \end{aligned} \quad (1)$$

where α is the learning rate. Q-learning is obtained by replacing the update of ΔQ by:

$$\Delta Q(s_t,a_t) \leftarrow \alpha(r_t+\gamma \max_a Q(s_{t+1},a)-Q(s_t,a_t)).$$

Iterating these updates under appropriate conditions, optimal convergence of Q-values is guaranteed with probability 1 in discrete MDPs (Sutton & Barto, 1998; Watkins, 1989); the resulting optimal policy can be readily obtained.

2.2. Modeling Private Information in DRL

Let $h_t=(s_t,a_t,r_t,s_{t+1},a_{t+1})$, let $H=\{h_t\}$, and suppose there are m agents. We consider two kinds of partitioning of H (see Fig. 1).

Partitioned-by-Time. This model assumes that only one agents interacts with the environment at any time step t. Let T^i be the set of time steps at which only ith agent has interactions with the environment. Then $T^i \cap T^j = \emptyset, (i \neq j)$ and the set $H^i=\{h_t \mid t \in T^i\}$ is considered the private information of the ith agent.

Partitioned-by-Observation. This model assumes that states and actions are represented as a collection of state and action variables. The state space and the action space are $S=\prod_i S^i$ and $A=\prod_i A^i$ where S^i and A^i are the space of the ith agent's state and action variables, respectively. Without loss of generality (and for notational simplicity), we consider each agent's local state and action spaces to consist of a single variable. If $s_t \in S$ is the joint state of the agents at time t, we denote by s_t^i the state that ith agent perceives and by a_t^i the action of ith agent. Let r_t^i be the *local reward* of ith agent obtained at time t. We define the *global reward* (or *reward* for short) as $r_t=\sum_i r_t^i$ in this model. Our Q-functions are evaluated based on this global reword. The perception of the ith agent at time t is denoted as $h_t^i=\{s_t^i,a_t^i,r_t^i,s_{t+1}^i,a_{t+1}^i\}$. The private information of the ith agent is $H^i=\{h_t^i\}$.

We note that partitioning by observation is more general than partitioning by time, in that one can always represent a sequence that is partitioned by time by one that is partitioned by observation. However, we provide more efficient solutions in simpler case of partitioning by time.

Let π^c be a policy learned by CRL. Then, informally, the objective of PPRL is stated as follows:

Statement 1. *The ith agent takes H^i as inputs. After the execution of PPRL, all agents learn a policy π which is equivalent to π^c. Furthermore, no agent can learn anything that cannot be inferred from π and its own private input.*

This problem statement can be formalized as in SFE (Goldreich, 2004). This is a strong privacy requirement which precludes consideration of solutions that reveal intermediate Q-values, actions taken, or states visited. We assume our agents behave *semi-honestly*, a common assumption in SFE—this assumes agents follows their specified protocol properly, but might also use their records of intermediate computations in order to attempt to learn other parties' private information.

3. Cryptographic Building Blocks

Our solutions make use of several existing cryptographic tools. Specifically, in our protocol, Q-values are encrypted by an additive homomorphic cryptosystem, which allows the addition of encrypted values without requiring their decryption, as described in Section 3.1. Using the homomorphic properties, this allows encrypted Q-values are updated in the regular RL manner, while unencrypted Q-values are not known to agents. For computations which cannot be treated by the homomorphic property, we use SFE as a primitive, as we describe in Section 3.2.

3.1. Homomorphic Public Key Cryptosystems

In a public key cryptosystem, encryption uses a *public key* that can be known to everyone, while decryption requires knowledge of the corresponding *private key*. Given a corresponding pair of $(\mathsf{sk},\mathsf{pk})$ of private and public keys and a message m, then $c=e_{\mathsf{pk}}(m;\ell)$ denotes a (random) encryption of m, and $m=d_{\mathsf{sk}}(c)$ denotes decryption. The encrypted value c uniformly distributes over $\mathbb{Z}_N$ if ℓ is taken from $\mathbb{Z}_N$ randomly. An *additive homomorphic cryptosystem* allows addition computations on encrypted values without knowl-

edge of the secret key. Specifically, there is some operation $\cdot$ (not requiring knowledge of sk) such that for any plaintexts m_1 and m_2, $e_{\mathsf{pk}}(m_1 + m_2; \ell) = e_{\mathsf{pk}}(m_1; \ell_1) \cdot e_{\mathsf{pk}}(m_2; \ell_2)$, where ℓ is uniformly random provided that at least one of ℓ_1 and ℓ_2 is. Based on this property, it also follows that given a constant k and the encryption $e_{\mathsf{pk}}(m_1; \ell)$, we can compute multiplications by k via repeated application of $\cdot$. This also enables a re-randomization property, which allows the computation of a new random encryption $c' = e_{\mathsf{pk}}(m; \ell')$ of m from an existing encryption $c = e_{\mathsf{pk}}(m; \ell)$ of m, again without knowledge of the private key or of m, as follows: $e_{\mathsf{pk}}(m; \ell) = \mathrm{Enc}_{\mathsf{pk}}(m; \ell_1) \cdot \mathrm{Enc}_{\mathsf{pk}}(0; \ell_2)$. In the rest of the paper, we omit the random number ℓ from our encryptions for simplicity.

In an (m, t)-*threshold cryptosystem*, m agents share a common public key pk while the agents hold different private keys $\mathsf{sk}^1, ..., \mathsf{sk}^n$. Each agent can encrypt any message with the common public key. Decryption cannot be performed by fewer than t agents, and can be performed by any group of at least t agents using a recovery algorithm based on the public key and their *decryption shares* $d_{\mathsf{sk}^1}(c), ..., d_{\mathsf{sk}^n}(c)$. We require a cryptosystem that provides semantic security (under appropriate computational hardness assumptions), re-randomization, the additive homomorphic property, and threshold decryption, such as the generalized Paillier cryptosystem (Dåmgard & Jurik, 2001).

3.2. Private Comparison and Division

As mentioned, secure function evaluation (SFE) is a cryptographic primitive which allows two or more parties to evaluate a specified function of their inputs without revealing (anything else about) their inputs to each other (Goldreich, 2004; Yao, 1986). Although our overall solution is more efficient than using SFE, we do make use of SFE for two kinds of computations.

One is the problem of *private comparison of random shares*. Let $\boldsymbol{x} = (x_1, ..., x_d) \in \mathbb{Z}_N^d$. For our purposes, A and B have *random shares* of $\boldsymbol{x}$ if A has $\boldsymbol{x}^A = (x_1^A, ..., x_d^A)$ and B has $\boldsymbol{x}^B = (x_1^B, ..., x_d^B)$ such that x_i^A and x_i^B are uniformly distributed in $\mathbb{Z}_N$ such that $x_i = (x_i^A + x_i^B) \mod N$ for all i. If A holds $\boldsymbol{x}^A$ and B holds $\boldsymbol{x}^B$, where $\boldsymbol{x}^A$ and $\boldsymbol{x}^B$ are random shares of $\boldsymbol{x}$, then private comparison of random shares computes the index i^* such that $i^* = \arg\max_i (x_i^A + x_i^B)$ in such a way that A learns only i^* and B learns nothing.

The other is a problem of *private division of random shares*. The input of A and B are random shares of x, $x^A \in \mathbb{Z}_N$ and $x^B \in \mathbb{Z}_N$, respectively. Let K be an integer known to both parties. Then, private division of random shares computes random shares Q^A and Q^B of quotient $Q \in \mathbb{Z}_N$ such that $x = (QK + R) \mod N$, where $R \in \mathbb{Z}_N\, (0 \leq R < K), Q = (Q^A + Q^B) \mod N$. After the protocol, A and B learn Q^A and Q^B, respectively, and nothing else.

We use private division of random shares in several places in our protocols to achieve *private division of encrypted values*. Suppose agent A has a key pair $(\mathsf{pk}, \mathsf{sk})$ and agent B knows pk and $e_{\mathsf{pk}}(x)$. The following protocol allows B to learn the encrypted quotient $e_{\mathsf{pk}}(Q)$ from $e_{\mathsf{pk}}(x)$ and K:

1. B computes $c \leftarrow e_{\mathsf{pk}}(x) \cdot e_{\mathsf{pk}}(-x^B), x^B \in_r \mathbb{Z}_N$ and send c to A.
2. A computes the decryption $x^A \leftarrow d_{\mathsf{sk}}(c) (\equiv x - x^B \mod N)$.
3. Using SFE for private division on A and B's inputs x^A and x^B, respectively, A and B obtain outputs Q^A and Q^B, respectively.
4. A sends $e_{\mathsf{pk}}(Q^A)$ to B.
5. B computes $e_{\mathsf{pk}}(Q) \leftarrow e_{\mathsf{pk}}(Q^A) \cdot e_{\mathsf{pk}}(Q^B)$.

4. Private Q-Value Update

In this section, we describe privacy-preserving SARSA update of Q-values under random action selection is described for our two partitioning models. We extend this to (ϵ-)greedy action selection in Section 5. We assume that reward r, learning rate α, and discount rate γ are non-negative rational numbers and that $\sum_{t=1}^{\infty}(\gamma^t L r_{\max}) < N$, where $r_{\max}$ is the largest reward that agents can obtain and $L \in \mathbb{Z}_N$ is a parameter defined in Section 4.1. In this paper, we describe protocols for two agents; these can be extended to m-agent case ($m \geq 3$) straightforwardly, as will be shown in an extended version of the paper.

4.1. Partitioned-by-Time Model

We first restrict our attention to the case where agent A has perceptions during $T^A = \{1, ..., t-1\}$ and B has perceptions during $T^B = \{t\}$. In this setting, A first computes can learn intermediate Q-values during the time period T^A, because they can be locally computed only from A's perception. At time t, the new Q-values must be computed based on the intermediate Q-values known to A and B's observation at time t. In brief, we do this by carrying out the update on encrypted Q-values using the homomorphic property to carry this out privately. However, the update includes the multiplication of rational numbers, such as α or γ, so the computation is not closed in $\mathbb{Z}_N$. Hence, we first scale these rational numbers by multiplying with large enough integers so that all computations are closed in $\mathbb{Z}_N$. We use private division of encrypted values to remove the scaling.

- Public input; L, K, learning rate α, discount rate γ
- A's input: $Q(s,a)$ (trained by A during T^A)
- B's input: (s_t, a_t)
- A's output: Nothing
- B's output: Encryption of updated Q-value $c(s_t, a_t)$

1. A: Compute $e^A(Q(s,a))$ for all (s,a) and send to B.
2. B: Take action a_t and get r_t, s_{t+1}.
3. B: Choose a_{t+1} randomly.
4. Update Q-value:
   (a) B: Compute $e^A(K\Delta Q(s_t,a_t))$ by eq. 3.
   (b) B: Do private division of $e^A(K\Delta Q(s_t,a_t))$ with A, then B learns $e^A(\Delta Q'(s_t,a_t))$.
   (c) B: Update $c(s_t,a_t)$ by eq. 4.

Figure 2. Private update of Q-values in partitioned-by-time model (SARSA/random action selection)

We now describe our protocol for private update, shown in Fig. 2, in more detail. Let pk_A be A's public key. At step 1, A computes $c(s,a) = e_{\mathsf{pk}_A}(Q(s,a))$ for all (s,a) and sends them to B. B takes action a_t, gets r_t, s_{t+1} (step 2), and chooses a_{t+1} randomly (step 3). A and B must now update the encrypted Q-value $c(s,a)$. By encrypting both sides of SARSA update (eq. 1), we obtain:

$$\begin{aligned} c(s_t,a_t) &\leftarrow e_{\mathsf{pk}_A}(\Delta Q(s_t,a_t) + Q(s_t,a_t)), \\ &= e_{\mathsf{pk}_A}(\Delta Q(s_t,a_t)) \cdot e_{\mathsf{pk}_A}(Q(s_t,a_t)) \\ &= \Delta c(s_t,a_t) \cdot c(s_t,a_t), \end{aligned} \tag{2}$$

where $\Delta c(s_t,a_t) = e_{\mathsf{pk}_A}(\Delta Q(s_t,a_t))$. If $\Delta c(s_t,a_t)$ is computed by B from what B observes, B can update $c(s_t,a_t)$ by eq. 2 locally. Therefore, step 4 is devoted to the computation of $\Delta c(s_t,a_t)$.

As mentioned, large integers K and L are used to treat the multiplication of rationals α and γ, where $\alpha\gamma K \in \mathbb{Z}_N$ and $Lr_t \in \mathbb{Z}_N$ for all r_t. Multiplying K to both sides of eq. 1 and multiplying L to r_t, we obtain

$$K\Delta Q(s_t,a_t) \leftarrow K\alpha(Lr_t + \gamma Q(s_{t+1},a_{t+1}) - Q(s_t,a_t)),$$

in which the computation is closed in $\mathbb{Z}_N$. Encrypting both sides by A's public key, we obtain

$$\begin{aligned} &e_{\mathsf{pk}_A}(K\Delta Q(s_t,a_t)) \\ &\quad = e_{\mathsf{pk}_A}(Lr_t)^{\alpha K} \cdot c(s_{t+1},a_{t+1})^{\alpha\gamma K} \cdot c(s_t,a_t)^{-\alpha K} \end{aligned} \tag{3}$$

Since K, L, α, γ are public and B has r_t, $c(s,a)$, B can compute $e_{\mathsf{pk}_A}(K\Delta Q(s_t,a_t))$ by eq. 3 (step 4(a)). B needs to divide $e_{\mathsf{pk}_A}(K\Delta Q(s_t,a_t))$ by K, however, division is again not allowable. Instead, a quotient $\Delta Q'(s_t,a_t)$ satisfying $\Delta Q(s_t,a_t) = K\Delta Q'(s_t,a_T) + R (0 \le R < K)$ is computed by private encrypted division and B obtains $e_{\mathsf{pk}_A}(\Delta Q'(s_t,a_t))$ (step 4(b)). Then, B finally computes

$$c(s_t,a_t) \leftarrow e_{\mathsf{pk}_A}(\Delta Q'(s_t,a_t)) \cdot c(s_t,a_t). \tag{4}$$

It follows that eq. 4 is equivalent to eq. 2 except for the truncation error included by the private encrypted division step (step 4(c)). This truncation is negligibly small if L is sufficiently large.

Lemma 1. *If A and B behave semi-honestly, then after the private update of Q-values for SARSA and random action selection in partitioned-by-time model, B correctly updates encrypted Q-values but learn nothing else. A learns nothing.*

The proof of this lemma (omitted for space) follows the standardized proof methodology of secure multi-party computation (Goldreich, 2004), showing that one can create the required algorithms, called *simulators*, for A and B. Intuitively, Step 4(b) is secure because it is implemented by SFE. Everything else that B receives except for messages received at steps for step 4(b) are encrypted by A's public key, so do not reveal anything. A does not receive anything except messages that are part of the SFE in step 4(b), so does not learn anything. Thus, the protocol is secure overall.

For the general setting of T^A and T^B, after time t, if B interacts with the environment at time $t+1$ again, the protocol can be started from step 2. When interaction switches back to A, an SFE step is used to change the encryption of the Q-values from A's private key to B's private key via an SFE step, and then the roles of A and B are switched.

4.2. Partitioned-by-Observation Model

In this model, we use a $(2,2)$-threshold cryptosystem. Both parties share a common public key pk: encryption of m by pk is denoted by $e(m)$ in this section. A and B hold different secret keys sk^A and sk^B for decryption shares, respectively. A and B cannot decrypt without both cooperating.

In this partitioning model, we write $a_t = (a_t^A, a_t^B)$, $s_t = (s_t^A, s_t^B)$, and $r_t = r_t^A + r_t^B$. A receives only (s_t^A, a_t^A, r_t^A) and B receives only (s_t^B, a_t^B, r_t^B). Private update of Q-values in this model is shown in Fig. 3. In this model, eq. 3 is rewritten as

$$\begin{aligned} &e(K\Delta Q(s_t,a_t)) = X^A \cdot X^B \cdot X \\ &X^A = e(Lr_t^A)^{\alpha KL}, \quad X^B = e(Lr_t^B)^{\alpha KL}, \\ &X = c(s_{t+1},a_{t+1})^{\alpha\gamma K} \cdot c(s_t,a_t)^{-\alpha K}. \end{aligned} \tag{5}$$

X^A and X^B can be computed by A and B. To obtain $c(s_{t+1},a_{t+1})$ and $c(s_t,a_t)$, let h,i,j,k be indices of Q-tables where $h \in S^A, i \in S^B, j \in A^A, k \in A^B$. At

step 4(a), A sends X^A and tables $\{c_{ik}\}, \{c'_{ik}\}$ with re-randomization such that

$$c_{ik} = c(s_t^A, i, a_t^A, k) \cdot e(0)\ (i \in S^B, k \in A^B), \quad (6)$$
$$c'_{ik} = c(s_{t+1}^A, i, a_{t+1}^A, k) \cdot e(0)\ (i \in S^B, k \in A^B), \quad (7)$$

to B. B determines $c(s_t, a_t) = c_{s_t^B, a_t^B}$, $c(s_{t+1}, a_{t+1}) = c'_{s_{t+1}^B, a_{t+1}^B}$ and obtains $e(K\Delta Q(s_t, a_t))$ by eq. 5 (step 4(b)). Then computes $e(\Delta Q'(s_t, a_t))$ by private division (step 4(c)). For all $(hijk)$, B sets

$$\Delta c_{hijk} \leftarrow \begin{cases} e(\Delta Q'(s_t, a_t)) & (i = s_t^B, k = a_t^B) \\ e(0) & \text{(o.w.)} \end{cases} \quad (8)$$

and sends $\{\Delta c_{hijk}\}$ to A (step 4(d)). Finally, for all (ik), Q-values are updated as

$$c(s_t^A, i, a_t^A, k) \leftarrow c(s_t^A, i, a_t^A, k) \cdot \Delta c_{s_t^A i a_t^A k}. \quad (9)$$

by A. With this update, $e(\Delta Q'(s_t, a_t))$ is added only when $(h, i, j, k) = (s_t^A, s_t^B, a_t^A, a_t^B)$. Otherwise, $e(0)$ is added. Note that A cannot tell which element is $e(\Delta Q'(s_t, a_t))$ in $\{\Delta c_{hijk}\}$ because of the re-randomization. Thus, eq. 9 is the desired update.

Lemma 2. *If A and B behave semi-honestly, then after the private update of Q-values for SARSA and random action selection in partitioned-by-observation model, A updates encrypted Q-values correctly but learns nothing. B learns nothing.*

By iterating private updates, encrypted Q-values trained by SARSA learning are obtained.

5. Private Greedy Action Selection

Private distributed algorithms for greedy action selection to compute $a^* = \arg\max_a Q(s, a)$ from encrypted Q-values in both partitioning models are described. These are used for: (1) (ϵ-)greedy action selection, (2) max operation in updates of Q-learning, and (3) extracting learned policies from final Q-values. In the partitioned-by-time model, this is readily solved by using private comparison, so is omitted.

5.1. Private Greedy Action Selection in Partitioned-by-observation Model

When A and B observe s_t^A and s_t^B, respectively, private greedy action selection requires that (1) A obtains a^{A*} and nothing else, (2) B obtains a^{B*} and nothing else, where $(a^{A*}, a^{B*}) = \arg\max_{(a^A, a^B)}(Q(s_t^A, a^A, s_t^B, a^B))$.

The protocol is described in Fig. 4. Threshold decryption is used here, too. First, A sends encrypted Q-values $c(s_t^A, i, j, k)$ with re-randomization for all

- Public input; L, K, learning rate α, discount rate γ
- A's input: (s_t^A, a_t^A), B's input: (s_t^B, a_t^B)
- A's output: Encryption of updated Q-value $c(s_t, a_t)$
- B's output: Nothing

1. A: Initialize $Q(s, a)$ arbitrarily and compute $c(s, a)(= e(Q(s, a)))$ for all (s, a).
2. Interaction with the environment:
   - A: Take action a_t^A and get r_t^A, s_{t+1}^A.
   - B: Take action a_t^B and get r_t^B, s_{t+1}^B.
3. Action selection:
   - A: Choose a_{t+1}^A randomly.
   - B: Choose a_{t+1}^B randomly.
4. Update Q-value:
   (a) A: Send $X^A, \{c_{ik}\}, \{c'_{ik}\}$ to B by eq. 6, 7.
   (b) B: Compute $e(K\Delta Q(s_t, a_t))$ by eq. 5
   (c) B: Do private division of $e(K\Delta Q(s_t, a_t))$ with A, then B learns $e(\Delta Q'(s_t, a_t))$.
   (d) B: Generate $\{\Delta c_{hijk}\}$ by eq. 8 and send it to A.
   (e) A: Update $c(s, a)$ with $\{\Delta c_{hijk}\}$ by eq. 9.

Figure 3. Private update of Q-values in partitioned-by-observation model (SARSA/random action selection)

(i, j, k). For all (i, k), B generates and sends a table $\{c_{ik}\}$ and $\{\sigma_{ik}\}$ whose values are set to

$$c_{ik} = c(s_t^A, i, s_t^B, \pi(k)) \cdot e(-Q^B_{i\pi(k)}), \quad (10)$$
$$\sigma^B_{ik} = d^B(c_{ik}), \quad (11)$$

where $\pi : S^B \mapsto S^B$ is a random permutation and $Q^B_{i\pi(k)} \in_r \mathbb{Z}_N$. At the third step, A recovers $Q^A_{ik} (= Q(s_t^A, i, s_t^B, k) - Q^B_{ik})$. With these random shares of $Q(s_t^A, i, s_t^B, \pi(k))$, the values $(i^*, k^*) = \arg\max_{(i,k)}(Q^A_{ik} + Q^B_{ik})$ are obtained by A using private comparison. Finally, B learns $a^{B*} = \pi^{-1}(k^*)$, where π^{-1} is the inverse of π.

Lemma 3. *If A and B behaves semi-honestly, then, after the execution of private greedy action selection, A learns a^{A*} and nothing else. B learns a^{B*} and nothing else.*

Note that a^{B*} is not learned by A because index k is obscured by the random permutation generated by B.

5.2. Security of PPRL

Privacy-preserving SARSA learning is constructed by alternate iterations of private update and random action selection. The policy π can be extracted by computing $\arg\max_a Q(s, a)$ for all (s, a) using private greedy action selection. The security follows from the earlier lemmas:

- A's input: $c(s,a)$ for all (s,a), s_t^A, B's input: s_t^B
- A's output: a^{A*}, B's output: a^{B*}

1. A: For all $i \in S^B, j \in A^A, k \in A^B$, send $c(s_t^A, i, j, k)$ to B.
2. B: For all $j \in A^A, k \in A^B$, compute c_{ik}(eq. 10), σ_{ik}^B(eq. 11) and send $\{c_{ik}\}, \{\sigma_{ik}\}$to A.
3. A: For all $i \in A^A, k \in A^B$, compute $\sigma_{ik}^B = d^B(c_{ik})$. Then, compute Q_{ik}^A by applying the threshold decryption recovery algorithm with public key pk and shares $\sigma_{ik}^A, \sigma_{ik}^B$.
4. A and B: Compute $(i^*, k^*) = \arg\max_{(i,k)}(Q_{ik}^A + Q_{ik}^B)$ by private comparison. (A learns $(i^*.k^*)$.)
5. A: Send k^* to B. Then output $a^{A*} = i^*$.
6. B: Output $a^{B*} = \pi^{-1}(k^*)$.

Figure 4. Private greedy action selection in partitioned-by-observation model

Theorem 1. *SARSA learning with private update of Q-values and random action selection is secure in the sense of Statement 1.*

Privacy-preserving SARSA learning and Q-learning with (ϵ-)greedy action selection can be constructed by combining private update and private greedy random action selection. However, these PPRLs do not follow Statement 1 because it does not allow agents to know greedy actions obtained in the middle of the learning. Therefore, the problem definition is relaxed as follows:

Statement 2. *The ith agent takes H^i as inputs. After the execution of PPRL, all agents learn a series of greedy actions during learning steps and a policy π which is equivalent to π^c. Furthermore, no agent learns anything else.*

Theorem 2. *SARSA and Q-learning with private update of Q-values and private greedy/ϵ-greedy action selection is secure in the sense of Statement 2.*

6. Experimental Results

We performed experiments to examine the efficiency of PPRL. Programs were written in Java 1.5.0. As the cryptosystem, (Dåmgard & Jurik, 2001) with 1024-bit keys was used. For SFE, Fairplay (Malkhi et al., 2004) was used. Experiments were carried out under Linux with 1.2 GHz CPU and 2GB RAM.

6.1. Random Walk Task

This random walk task is partitioned by time. The state space is $S = \{s_1, ..., s_n\}(n = 40)$ and the action space is $A = \{a_1, a_2\}$. The initial and goal states are s_1 and s_n, respectively. When a_1 is taken at $s_p(p \neq n)$, the agent moves to s_{p+1}. When a_2 is taken at $s_p(p \neq 1)$, the agent moves to s_{p-1}, but the agent does not move when $p = 1$. A reward $r = 1$ is given only when the agent takes a_1 at s_{n-1}; else, $r = 0$. The episode is terminated at s_n or after $1,000$ steps.

A learns $15,000$ steps and then B learns $15,000$ steps. CRL, IDRL, PPRL, and SFE were compared. SARSA learning with random or ϵ-greedy action selection was used for all settings. Table 2 shows the comparison results of computational cost, learning accuracy (number of steps to reach the goal state, averaged over 30 trials, and number of trials that successfully reach the goal state), and privacy preservation.

Learning accuracy of PPRL and SFE are the same as CRL because the policy learned by PPRL and SFE are guaranteed to be equal to the one learned by CRL. In contrast, the optimal policy is not obtained successfully by IDRL because learning steps for IDRL agents correspond to the half of others. Because most of the computation time is spent for private division and comparison, computation time with random selection is much smaller than with ϵ-greedy selection. These experiments demonstrate that PPRL obtains good learning accuracy, while IDRL does not, though computation time is larger than DRL and IDRL.

6.2. Load Balancing Task

In these experiments, we consider a load balancing problem (Cogill et al., 2006) in the partitioned-by-observation model with two factories A and B. Each factory can observe its own backlog $s^A, s^B \in \{0, ..., 5\}$. At each time step, each factory decides whether or not to pass a job to other factories; the action variable is $a^A, a^B \in \{0, 1\}$. Jobs arrive and are processed independently at each time step with probability 0.4 and 0.48, respectively. Agent A receives reward $r^A = 50 - (s^A)^2$. If A passes the job to B, then A's reward is reduced by 2 as a cost for redirection. If an overflow happens, the job is lost and $r^A = 0$ is given. Similarly, r^B is computed as well. Perceptions (s^A, a^A, r^A) and (s^B, a^B, r^B) are to be kept private. (In this task, actions cannot be kept private because the parties learn them from whether the job was passed or not.)

Distributed reward DRL (RDRL) (Schneider et al., 1999) is tested in addition to the four RLs tested earlier. RDRL is a variant of DRL, which is the same with IDRL except that global rewards are shared among distributed agents (Schneider et al., 1999). SARSA/ϵ-greedy action selection was used in all settings. Fig 5 shows the changes of sum of global rewards per episode. For avoiding overflows, cooperation be-

Table 2. Comparison of efficiency in random walk tasks

	comp. (sec)	accuracy avg.	accuracy #goal	privacy loss
CRL/rnd.	0.901	40.0	30/30	disclosed all
IDRL/rnd.	0.457	247	8/30	Stmt. 1
PPRL/rnd.	4.71×10^3	40.0	30/30	Stmt. 1
SFE/rnd.	$> 7.0 \times 10^6$	40.0	30/30	Stmt. 1
CRL/ϵ-grd.	0.946	40.0	30/30	disclosed all
IDRL/ϵ-grd.	0.481	—	0/30	Stmt. 2
PPRL/ϵ-grd.	3.36×10^4	40.0	30/30	Stmt. 2
SFE/ϵ-grd.	$> 7.0 \times 10^6$	40.0	30/30	Stmt. 2

Table 3. Comparison of efficiency in load balancing tasks.

	comp. (sec)	accuracy	privacy loss
CRL	5.11	90.0	disclosed all
RDRL	5.24	87.4	partially disclosed
IDRL	5.81	84.2	Stmt. 1
PPRL	8.85×10^5	90.0	Stmt. 2
SFE	$> 2.0 \times 10^7$	90.0	Stmt. 2

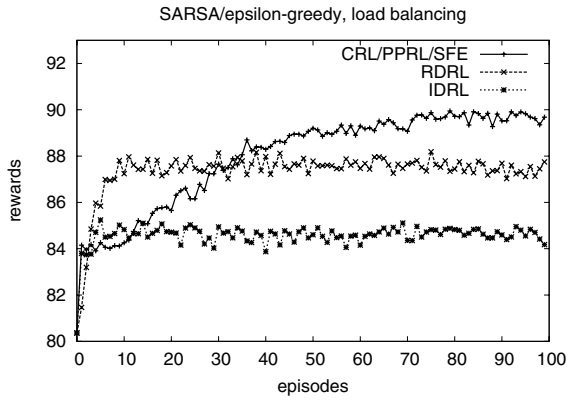

Figure 5. Performance evaluation (sum of global rewards in an episode, normalized by the number of steps in an episode) in load balancing tasks (average of 100 trials).

tween agents is essential in this task. The performance of IDRL agents is inferior to others because selfish behavior is learned. In contrast, CRL, PPRL and SFE agents successfully obtain cooperative behavior. The performance of RDRL is intermediate because perceptions of RDRL agents are limited. Efficiency is shown in Table 3. Since ϵ-greedy action selection was used, the privacy of IDRL, PPRL and SFE follow Statement 2. The privacy preservation of RDRL is between CRL and PPRL. As discussed in Section 1, PPRL achieves both the guarantee of privacy preservation and the optimality which is equivalent to that of CRL; SFE does the same, but at a much higher computational time.

Acknowledgments

This work was started at Tokyo Institute of Technology and carried out partly while the first author was a visitor of the DIMACS Center. The third author is partially supported by the National Science Foundation under grant number CNS-0822269.

References

Abe, N., Verma, N., Apte, C., & Schroko, R. (2004). Cross channel optimized marketing by reinforcement learning. *ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining* (pp. 767–772).

Cogill, R., Rotkowitz, M., Van Roy, B., & Lall, S. (2006). An Approximate Dynamic Programming Approach to Decentralized Control of Stochastic Systems. *LNCIS*, *329*, 243–256.

Dåmgard, I., & Jurik, M. (2001). A Generalisation, a Simplification and Some Applications of Paillier's Probabilistic Public-Key System. *Public Key Cryptography 2001*. Springer.

Gambs, S., Kégl, B., & Aïmeur, E. (2007). Privacy-preserving boosting. *Data Mining and Knowledge Discovery*, *14*, 131–170.

Goldreich, O. (2004). *Foundations of Cryptography: Volume 2, Basic Applications*. Cambridge University Press.

Jagannathan, G., & Wright, R. N. (2005). Privacy-preserving distributed k-means clustering over arbitrarily partitioned data. *ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining* (pp. 593–599).

Kearns, M., Tan, J., & Wortman, J. (2007). Privacy-Preserving Belief Propagation and Sampling. *NIPS 20*.

Lindell, Y., & Pinkas, B. (2002). Privacy Preserving Data Mining. *Journal of Cryptology*, *15*, 177–206.

Malkhi, D., Nisan, N., Pinkas, B., & Sella, Y. (2004). Fairplay: a secure two-party computation system. *USENIX Security Symposium* (pp. 287–302).

Moallemi, C. C., & Roy, B. V. (2004). Distributed optimization in adaptive networks. *NIPS 16*.

Sakuma, J., & Kobayashi, S. (2008). Large-scale k-means Clustering with User-Centric Privacy Preservation. *Pacific-Asia Conf. on Knowledge Discovery and Data Mining (PAKDD) 2008, to appear*.

Schneider, J., Wong, W., Moore, A., & Riedmiller, M. (1999). Distributed value functions. *Int'l Conf. on Machine Learning* (pp. 371–378).

Sutton, R. S., & Barto, A. G. (1998). *Reinforcement Learning: An Introduction*. MIT Press.

Watkins, C. (1989). *Learning from Delayed Rewards*. Cambridge University.

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

Yu, H., Jiang, X., & Vaidya, J. (2006). Privacy-preserving SVM using nonlinear kernels on horizontally partitioned data. *ACM SAC* (pp. 603–610).

Zhang, S., & Makedon, F. (2005). Privacy preserving learning in negotiation. *ACM SAC* (pp. 821–825).