The Hardness of Learning Access Control Policies

by

Xiaomeng Lei

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2023

Author's Declaration

This thesis consists of material all of which I authored or co-authored: see Statement of Contributions included in the thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Statement of Contributions

This thesis is adapted from my published paper "The Hardness of Learning Access Control Policies" collaborated with my supervisor Mahesh V. Tripunitara.

Citation: Lei, Xiaomeng, and Mahesh Tripunitara. "The Hardness of Learning Access Control Policies." Proceedings of the 28th ACM Symposium on Access Control Models and Technologies. 2023.

Abstract

The problem of learning access control policies is gaining significant attention in research. We contribute to the foundations of this problem by posing and addressing meaningful questions on computational hardness. Our study focuses on learning access control policies within three different models: the access matrix, Role-Based Access Control (RBAC), and Relationship-Based Access Control (ReBAC), as described in existing literature. Our approach builds upon the well-established concept of Probably Approximately Correct (PAC) theory, with careful adaptations for our specific context. In our setup, the learning algorithm receives data or examples associated with access enforcement, which involves deciding whether an access request for resource should be accepted or denied. For the access matrix, we pose a learning problem that turns out to be computationally easy, and another that we prove is computationally hard. We generalize the former result so we have a sufficient condition for establishing other problems to be computationally easy. Building upon these findings, we examine five learning problems in the context of RBAC, of which three are identified as computationally easy and two are proven to be computationally hard. Finally, we consider four learning problems in the context of ReBAC, all of which are found to be computationally easy. Every proof for a problem that is computationally easy is constructive, in that we propose a learning algorithm for the problem that is efficient, and probably, approximately correct. As such, our work makes contributions at the foundations of an important, emerging aspect of access control, and thereby, information security.

Acknowledgements

I would like to thank my supervisor, Mahesh V. Tripunitara, whose guidance, expertise, and unwavering support have been essential in shaping my research and academic journey. His mentorship and encouragement has not only enriched my understanding of the subject matter but also inspired me to push the boundaries of my knowledge and capabilities.

I would also like to thank my loving boyfriend, whose belief in me has been a constant source of motivation. His support, understanding, and encouragement have provided me with the strength to overcome challenges and pursue my academic goals.

Furthermore, I would like to thank my parents, whose endless love and unwavering support have been the foundation of my educational journey.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Access control policies often suffer from opacity, affecting not only users but also administrators overseeing the policies. The complexity of access control policies, particularly in large organizations, systems, companies, and complex network environments, makes them difficult to intuitively understand and explain. Additionally, the lack of comprehensive documentation and explanations further hinders administrators' comprehension of the policy rules. The application of machine learning in the field of access control policies has become a hot topic. By leveraging machine learning techniques and analyzing a significant volume of input-output data from access control policies, machine learning models can automatically learn and adapt policies, providing a cost-effective approach. In this paper, we employ the theoretical framework of the PAC-learning model from the field of machine learning and computational complexity theory to investigate the feasibility of applying machine learning in access control policies.

1.1 The Access Control Policy

Access control plays a crucial role in computer privacy and security. It ensures that only authorized users are able to exercise certain actions on resources. It is an important part of the security of a system and its data. Whether a particular request for access is allowed or denied is typically dictated by an access control *policy*.

A policy is an articulation of who may access what, and in what manner. For example, a policy may state that Alice is allowed to read a particular file, but Bob is not. It ensures that only authorized users are able to carry out specific operations, such as reading or

writing, on protected resources or files. They also prevent unauthorized users from making access requests for performing operations on protected resources or files, thereby denying their access. A typical example is the policy in a file system. Suppose there is a file system with a folder containing protected files. The defines two roles: an administrator role and a regular user role, and specifies which roles can access the folders and files. The administrator role has the ability to view and modify all files, while the regular user role can only read the files without the ability to modify. The access control policy achieves user access control by setting permissions for each file to allow only the administrator role full access, while the regular user role has only read permissions. This ensures that only users with the appropriate roles can perform corresponding operations on the files. Consequently, for example, a user Alice with the administartor role is allowed to read and write all files in the folder, while bob with a regular user role is restricted to read-only access for all the files.

An access control policy is typically based on an access control *model*. In this paper, our main focus is on three access control models: the Access matrix, RBAC (Role-Based Access Control), and ReBAC (Relationship-Based Access Control), which are widely used for implementing access control policies. An access control model specifies two things: (i) a syntax in which a policy is expressed, and, (ii) a syntax for an *access-request* and the manner in which *access-enforcement* is carried out. An access-request, as the term suggests, is a request by a subject to exercise a right on an object. The set of all possible access requests is induced by the policy. Access-enforcement is an algorithm that, given input an access-request, decides whether it should be allowed or denied.

As an example, consider the following variant of the access matrix model [9] (we introduce a different variant in Chapter 3). An example of a policy in this model is shown in Figure 1.1. As the figure suggests, a policy in this model is specified by a 4-tuple: a set of subjects S, a set of objects, O, a set of rights R and a function $m : S \times O \to 2^R$ where 2^R denotes the power set, i.e., set of subsets, of R. In the example in Figure 1.1, $S = \{\text{alice, bob, carol}\}$, $O = \{\text{alice, bob, carol, secret-file}\}$ (it is customary for policies in this model that $S \subseteq O$), $R = \{\text{admin, read, write}\}$ and m is rendered as a two-dimensional table in the figure.

The set of all access-requests induced by a policy in this model is $S \times O \times R$, i.e., an access-request is a tuple $\langle s, o, r \rangle$ which expresses that a subject s seeks to exercise the right r on an object o. An access-request is allowed if and only if in the policy, it is true that $s \in S, o \in O, r \in R$ and $r \in m[s, o]$. Access-enforcement is the corresponding straightforward set of lookups. For example, the access request $\langle \text{alice}, \text{bob}, \text{admin} \rangle$ is allowed, while $\langle \text{bob}, \text{secret-file}, \text{write} \rangle$ is denied.

	alice	bob	carol	secret-file
alice	admin	admin		read, write
bob				read
carol			admin	read

Figure 1.1: A policy in a variant of the access matrix model.

1.2 The learning problem

Interest in learning access control policies is motivated by prior work, for example, that of Le et al. [16] and Masoumzadeh [17]. The former suggests that a reason is the validation of an access control policy that is implemented, i.e., learning a policy from access enforcement so we can then reason about whether the policy is indeed what we intend. The latter suggests also learning from access enforcement, except in the somewhat different situation that the policy impacts particular entities, e.g., users in a social network, to whom the policy is not made explicit. The more recent work of Iyer and Masoumzadeh [12, 13] leverages these motivations to then construct an approach to learning a policy in such "black box" settings, i.e., from information that is generated during access enforcement.

We address the somewhat fundamental question as to whether there exists an algorithm with desirable properties that learns such an access control policy. The desirable properties we seek are that the algorithm is efficient, i.e., runs in time polynomial in certain parameters we associate with the problem, and that the output policy, with high probability, suffers low error only. (We make these precise in Section 2.1.)

A question with any learning problem is what data we provide an algorithm. Our focus, as in prior work [16, 17, 13, 12], is data generated by the process of access-enforcement. In practice, such information is typically logged for later use such as audits and forensics. An example of such data are $\langle$access-request, allow/deny$\rangle$ pairs, e.g., $\langle\langle \text{alice}, \text{bob}, \text{admin}\rangle, \text{allow}\rangle$ and $\langle\langle \text{bob}, \text{secret-file}, \text{write}\rangle, \text{deny}\rangle$ for our example policy from Figure 1.1. Other kinds of data are possible as well as input to a learning algorithm; for example, intermediate data that is generated towards determining an allow or deny verdict by access-enforcement (see the problems of Rows (2)–(4) in Table 5.1 of Chapter 5).

Another question is what the model for the policy that is output by the learning algorithm should be. What we suggest here, by intent, is that the model for the policy that is output by the learning algorithm does not necessarily have to be the same as that of the underlying policy. There is prior work that adopts this mindset; in the work of Iyer and Masoumzadeh [13] for example, the underlying policy is in a Relationship-Based Access

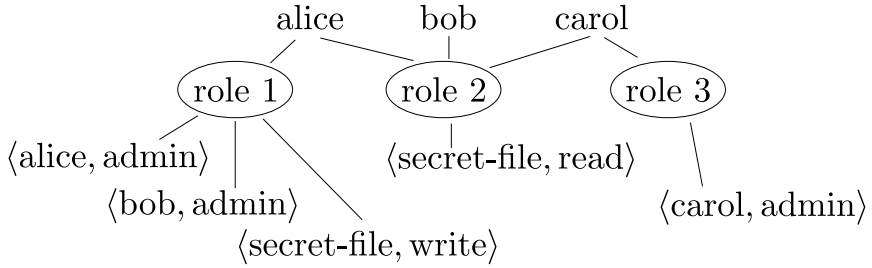


Figure 1.2: A Role-Based Access Control (RBAC) policy. Its authorizations are equivalent to the access matrix in Figure 1.1.

Control (ReBAC) model whereas the policy output by the learning algorithm is a state-machine. (We address exactly this problem in Chapter 5). There is also work outside of learning access control policies that addresses a different syntax for the output of the learning algorithm than the underlying concept, for example, requiring a learning algorithm to output an underlying boolean formula in disjunctive normal form [20]. In this regard, we can perceive our class of learning problems as a variant of the policy-mining problem that has been the subject of several pieces of prior work (see, for example, the survey of Mitra et al. [18]). In the policy-mining problem, one is given as input a policy in a model, and asked to output an equivalent policy in a different model. By "equivalent" here, we mean that the two policies allow exactly the same accesses. For example, in role-mining [18], the input policy model is an access matrix, and the output is a policy in Role-Based Access Control (RBAC). Indeed, as we address in Chapter 4, it is possible to pose exactly a generalization of some of the role mining problems that has been addressed in prior work as learning problems as we pose them.

To illustrate this point further here in the introductory chapter, we present an RBAC policy in Figure 1.2 that is equivalent, from the standpoint of authorizations, to the access matrix from Figure 1.1. In this variant of the RBAC model, a policy is a 5-tuple, $\langle U, P, R, UA, PA\rangle$, where U, P, R are pairwise disjoint sets of users, permissions and roles, respectively, $UA\colon U \to 2^R \setminus \{\emptyset\}$ is a function that associates a user with a non-empty set of roles, and $PA\colon P \to 2^R \setminus \{\emptyset\}$ is a function that associates a permission with a non-empty set of roles. A user $u \in U$ is authorized to a permission $p \in P$ if and only if $UA(u) \cap PA(p) \neq \emptyset$. In associating the access matrix from Figure 1.1 with the RBAC policy in Figure 1.2, we have adopted as the set of users U in RBAC exactly the set of subjects S in the access matrix. And, each permission in the RBAC policy is a pair $\langle \text{object}, \text{right}\rangle$ from the access matrix.

In the context of learning, depending on the data from access-enforcement that is

provided to a learning algorithm, the kinds of errors we are willing to tolerate in a policy that is output by the learning algorithm and any other constraints we impose, learning an RBAC policy from an underlying policy, be it an access matrix or another RBAC policy, may be computationally easy or hard. For example, the RBAC policy in Figure 1.2 minimizes the number of roles across all RBAC policies that are equivalent to the access matrix of Figure 1.1 and adopt the encoding that a permission is an $\langle \text{object}, \text{right} \rangle$ pair. We can demand that a learning algorithm satisfy such a constraint as well, as we discuss in Chapter 4. Then, the learning problem would be computationally hard, given that the problem of computing an RBAC policy of the minimum number of roles that is equivalent to a given access matrix is known to be **NP**-hard [4].

1.3 Computational hardness

The focus of our work is an identification, given a learning problem as we discuss above, as to whether there exists an efficient algorithm for it or not. Accordingly, our results are an association of "easy" or "hard" with each learning problem we consider (see, for example, Table 4.1 in Chapter 4). The setting we adopt so we can infer such results is that of Probably Approximately Correct (PAC)-learning [23], for which we provide background in Section 2.1. Some of the learning problems we consider do not conform strictly to PAC-learning, and in such cases, we extend the setting carefully and limitedly so we are still able to make such inferences (see, for example, the problems of Rows (2)–(4) in Table 5.1 in Chapter 5).

Under PAC-learning, we can dichomotize the computational hardness of a particular learning problem as either "there exists an efficient algorithm", i.e., "easy" in our parlance, or "an efficient algorithm is unlikely to exist", i.e., "hard". A proof for the former is by construction, i.e., by presentation of an algorithm, and a claim and associated proof for its correctness and efficiency. A proof for the latter is via reduction, and relies on the customary conjecture, $\mathbf{RP} \neq \mathbf{NP}$, where **RP** is the class of decision problems for which there exists a polynomial-time randomized algorithm, and **NP** is the class for which there exists a polynomial-time non-deterministic algorithm. The particular kind of reduction we leverage is similar to the polynomial-time Turing reduction [8], and the kind of problem from which we reduce is what we can call the *certificate construction* problem that corresponds to a decision problem that is **NP**-complete. An example certificate construction problem, denote it q_{cc}, which we use in two of our reductions (see Theorem 5 in Chapter 3, and the problem of Row (5) in Table 4.1 in Chapter 4) is: given an undirected graph $G = \langle V, E \rangle$ that is known to have a vertex cover of size k, where V and E are the vertex

set and the edge set of G respectively, output such a vertex cover C, i.e., a subset of V such that $|C| = k$ and every edge $e \in E$ is incident on a least one vertex in C. A corresponding decision problem, denote it q_{dec}, that is **NP**-complete is: given $\langle G, k \rangle$ where G is an undirected graph, does G have a vertex cover of size k? We can prove that if there exists a randomized polynomial-time algorithm for q_{cc}, then $q_{\text{dec}} \in \mathbf{RP}$, and because we know that q_{dec} is **NP**-complete, this would imply $\mathbf{RP} = \mathbf{NP}$, which conflicts with the conjecture that $\mathbf{RP} \neq \mathbf{NP}$. We refer the reader to Section 2.2 for a complete discussion.

1.4 Organization

The remainder of the paper is organized as follows. In the next chapter, we provide the necessary background and notions that we leverage in our work, in particular, the PAC-learning setting that we adopt. In Chapter 3 we address two learning problems for the access matrix. We address five learning problems for RBAC in Chapter 4. In Chapter 5, we address four learning problems for ReBAC. We discuss related work in Chapter 6, and conclude with Chapter 7 where we discuss also future work.

Chapter 2

Methodology

In this chapter, we elaborate on our theoretical understanding of the two primary techniques employed to study the hardness of access control policies. The first technique is the Probably Approximately Correct Learning model (PAC), which enables us to classify problems as either easy or hard. We refer to problems that can be effectively learned using PAC as easy, while those that cannot are considered hard. The second technique involves reductions, which we employ to prove that a learning problem is hard.

2.1 Learning Access Control Policies

When we seek to learn an underlying access control policy (a policy we aim to learn, often treated as a black box where only partial information is available), typically by sending access requests to the black box and receiving corresponding responses containing certain information, such as whether the access request is allowed or denied, about the policy, it is only natural to inquire whether it is feasible to produce a policy that closely approximates or is essentially identical to the underlying policy within a reasonable time, i.e. in polynomial time. The PAC-learning model, a highly influential theory in the field of machine learning, offers a solid foundation for addressing whether an access control policy can be learned efficiently or not. For a thorough understanding of the PAC-learning model, we suggest referring to the book mentioned [14].

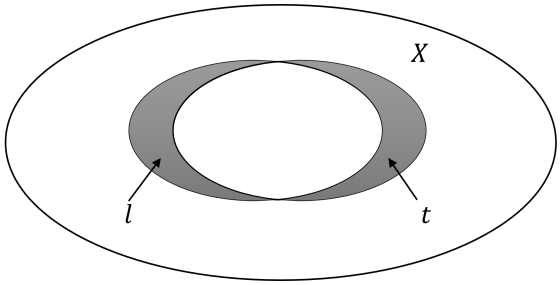


Figure 2.1: The measure of error, adapted from the work of Kearns and Vazirani [14]. X, l, and t represent the instance space, the underlying target, and the output target, respectively. The error can be quantified as the ratio between the shaded region, where the underlying and hypothesis targets do not overlap, and the total area of X, the entire region.

2.1.1 Definition of the PAC-learning model

The fundamental idea of the PAC learning model can be condensed into the following statement. Suppose there exists an unknown underlying target class L, and the objective of a learner is to learn any $l \in L$ where the underlying target l is just a set or a boolean function, and output t that is only approximations to l efficiently. The question we want to address is whether the underlying target class L is learnable, i.e., whether an efficient learning algorithm for learning every $l \in L$ approximate correctly exists. The PAC-learning model, in essence, defines the answer to this question: if a learning algorithm satisfying specific properties exists for a given underlying target set L, then we can say that the class L is PAC-learnable. In other words, we refer to the learning problem such that learning the class L as "easy."

The entire learning process is divided into two phases: training and testing. In the training phase, the learner is provided with labeled examples $x \in X$ that are randomly drawn from some fixed probability distribution $\mathcal{D}$ over X. Here, X refers to the instance space, which represents the set of all possible examples. Each example is assigned a label of either 0 or 1 by the underlying target l, indicating whether the example belongs to l or not. For example, consider an interval $[a, b]$ as the underlying target l, and the examples consist of tuples $\langle x, label \rangle$. If $a \leq x \leq b$, the *label* is set to 1, otherwise it is set to 0. The learner processes these examples and generates an output target t.

In the testing phase, the output t is evaluated using examples drawn from the same distribution $\mathcal{D}$ over the same instance space X used during training. Both the underlying target l and the output t can be seen as functions that map any given example x to either

0 or 1. Therefore, the label of an example x in the underlying target is denoted as $l(x)$, while the label assigned by the output is denoted as $t(x)$.

To quantify the disparity between the underlying target l and the output t, we use the term *error*, which measures the proportion of examples for which the labels provided by the underlying target and the output do not match. Mathematically, the error can be defined as:

$$error = \Pr_{x \sim \mathcal{D}} \{l(x) \neq t(x)\}$$

The error can also be visualized in figure 2.1, representing the probability that an example is randomly drawn from some fixed distribution $\mathcal{D}$ over X and falls within the shaded region.

When our goal is to learn an underlying target l, it is natural to aim for error that is as small as possible. This expectation aligns with the requirements of the PAC-learning model. Specifically, the PAC-learning model requires that the error should be bounded by any chosen value of ϵ, where $0 < \epsilon < 1/2$, with a high probability. When considering errors, we can rely on two intuitions. The first intuition is that the more examples the learner learns during the training process, the smaller the error is likely to be. The second intuition suggests that examples that the learner fails to learn during training are unlikely to contribute to errors during testing because they are also unlikely to appear during testing. This assumption holds since both the training and testing examples are drawn from the same distribution.

Formally, the PAC-learning defines that for any given values of ϵ and δ, where $0 < \epsilon < 1/2$ and $0 < \delta < 1/2$, class L is PAC-learnable if for every $l \in L$, the probability of the error being less than or equal to ϵ, denoted as $\Pr\{error \leq \epsilon\}$, is greater than or equal to $1 - \delta$, and if the learning algorithm exhibits polynomial time efficiency with respect to the size of the underlying target, i.e., *size(l)*, $1/\delta$, and $1/\epsilon$. In this context, we refer to problems that learning an efficiently PAC-learnable class as "easy" or while problems that learning an non-PAC-learnable class are considered "hard". Here, the term ϵ corresponds to "approximately," and $1 - \delta$ corresponds to "probably" in the phrase "probably approximately correct."

It is crucial to highlight that the choice of encoding for the underlying target class L can have a substantial effect on its PAC-learnability. We represent the encoding of the underlying class as the encoding class E. For instance, let us consider L as the underlying boolean formula class. If the encoding class is a 3-term conjunctive normal form (CNF), then the class L encoded in CNF is PAC-learnable. However, if the encoding class is a

3-term disjunctive normal form (DNF), then L encoded in DNF is non-PAC-learnable.

Lastly, we formally provide the definition of the PAC-learning model, which is adapted from [14].

Theorem 1. *Let L be a underlying target class over the instance space X and E be an encoding class over X. We say that L is PAC-learnable using E if there exists an algorithm A such that: for every target $e \in E$, for every distribution $\mathcal{D}$ on X, and for all $0 < \epsilon < \frac{1}{2}$ and $0 < \delta < \frac{1}{2}$, if algorithm A is given access to $EX(e, \mathcal{D})$ and inputs ϵ and δ, then with probability at least $1 - \delta$, algorithm A outputs t satisfying $error(t) \leq \epsilon$. This probability is taken over the random examples drawn by calls to $EX(e, \mathcal{D})$.*

2.1.2 The adapted PAC-learning model in our context

In this section, We introduce and customize some of the terminology and notions from Section 2.1.1 to our particular context of learning access control policies. Firstly, we present a formal definition to determine whether learning access control policies can be classified as "easy" or "hard". Then, in the subsequent paragraph, we establish a correspondence between the notations used in PAC learning, as discussed in Section 2.1.1, and the notations employed in the context of learning access control policies.

Theorem 2. *Let P_l be the underlying policy class over the access request space X and M_l be an encoding model over X. Let the learning problem P be: for each underlying policy $p_l \in P_l$ encoded in M_L, learn an output policy p_t over the access request space X and M_t be an encoding model over X. We say that the learning problem pis easy if there exists an algorithm A such that: for every $p_l \in P_l$, for every distribution $\mathcal{D}$ on X, and for all $0 < \epsilon < \frac{1}{2}$ and $0 < \delta < \frac{1}{2}$, if algorithm A is given access to $EX(p_l, \mathcal{D})$, then with probability at least $1 - \delta$, algorithm A outputs a policy p_t satisfying $error(p_t) \leq \epsilon$. This probability is taken over the random examples drawn by calls to $EX(p_l, \mathcal{D})$.*

The underlying policy class P_l in the learning access control policies context of learning the access control policies represents the underlying target class L, and the model M_l represents the encoding class E. The goal of learning P_l in model M_t over the access request space X is essentially learning L using E over the instance space X. The output policy p_t corresponds to t from the PAC-learning model. Furthermore, we extend our approach by not restricting the encoding of the output policy p_t, to be the same as the underlying policy p_l. In the subsequent discussions, for any learning problem, we explicitly indicate the encoding model M_l and M_t of the input and output policies, respectively. If there exists an algorithm A that satisfies the requirements stated in Theorem 1 for a given learning problem, we refer to that problem as "easy"; otherwise, it is considered "hard".

2.1.3 Learning access control policies using PAC-learning model

In this section, we provide a more detailed introduction to the learning notions that are relevant to our contributions. Additionally, we expand these notations to enhance their applicability within the access control policy learning domain.

Every learning problem we consider is set up as follows. There is an *underlying* access control policy $p_l \in P_l$ in some model M_l that the learning algorithm seeks to learn. We ask that the learning algorithm *output* a policy p_t in a model M_t where it may or may not be the case that $M_l = M_t$. We call each unit of data that is provided the learning algorithm an *example*; each such example is of the form $\langle \text{access-request}, \pi \rangle$, where π is some string that is generated as part of the process of access enforcement for the access-request. (Recall, from Chapter 1 that every such model M_l is associated with an algorithm for access enforcement.)

For example, M_l may be the access matrix model we discuss in Chapter 1 in the context of Figure 1.1, and M_t may be the RBAC model we discuss in the context of Figure 1.2. And if the examples provided the learning algorithm are $\langle \text{access-request}, b \rangle$, where $b \in \{0, 1\}$ denotes "allow" or "deny", then this would exactly be a learning version of the role-mining problem from the literature [18]. If we impose no further constraints on the output RBAC policy p_t, then the corresponding role-mining problem is easy: e.g., we could simply create one role per subject in the access matrix, and assign to the role those permissions, each of which is an $\langle \text{object}, \text{right} \rangle$ pair, to which the subject is authorized in the access matrix.

There are two goodness properties we seek in a learning algorithm. One is that the output policy needs to be *probably approximately correct*. To characterize what this means, we first specify a meaningful notion of an *error* in the output policy p_t when compared to the underlying policy p_l. An error is the probability that for an access-request that is drawn from the same distribution $\mathcal{D}$ as was adopted at the time the learning algorithm ran (but where the $\langle \text{access-request}, \pi \rangle$ pair was not necessarily seen by the learning algorithm before it outputs p_t), p_l differs from p_t. That is, if a represents an access-request, and we think of each of the underlying and output policies p_l, p_t as polynomial-time computable functions that take a as input, then, $\textit{error} = \Pr_{a \leftarrow \mathcal{D}} \{p_l(a) \neq p_t(a)\}$.

For example, if M_l is the access matrix model from Chapter 1, and M_t the RBAC model, then we may adopt as error the probability that an access-request is drawn from the distribution $\mathcal{D}$ for which the allow/deny verdict is different for the underlying access matrix policy p_l than the output RBAC policy p_t. Under this notion of error then, the error in the RBAC policy in Figure 1.2 relative to the underlying access matrix policy in Figure 1.1 is 0. Suppose, as a different example but for the same notion of error, the

output RBAC policy does not have the permission $\langle$secret-file, write$\rangle$ assigned to role 1, but otherwise, our output RBAC policy is the same as in Figure 1.2. Then, the error is the probability that the access request $\langle$alice, secret-file, write$\rangle$ is drawn, because for that access-request, the underlying policy would issue an "allow" verdict, but the output policy would issue a "deny".

In allowing an arbitrary string π as the second component of an example, and in leaving the exact characterization of what difference between p_l and p_t can cause non-zero error, we generalize classical PAC-learning [23]. In that classical setup, $\pi \in \{0, 1\}$, and p_l, p_t are functions whose codomain is $\{0, 1\}$.

Given a characterization for an error as we discuss above, we then require that the probability that the output policy suffers error more than an input parameter ϵ is bounded by another input parameter δ. For a learning algorithm to be deemed to be probably approximately correct, we require that for every ϵ, δ and distribution $\mathcal{D}$, where $0 < \epsilon < 1/2$ and $0 < \delta < 1/2$, it is the case that $\Pr\{\text{error} \leq \epsilon\} \geq 1 - \delta$. That is, ϵ corresponds to "approximately", and $1 - \delta$ corresponds to "probably" in the phrase "probably approximately correct".

The other goodness property we seek in a learning algorithm is that it must be efficient, i.e., run in time polynomial in (i) the size of (the encoding of) the underlying policy p_l, (ii) the inverse of the error-threshold, i.e., $1/\epsilon$, and, (iii) the inverse of the probability we have an error beyond the error-threshold, i.e., $1/\delta$. This implies that the size of the output policy p_t must be at worst polynomial in (i)–(iii), and so must the number of examples the algorithm is allowed to see before it outputs p_t. The reason for the former is that the size of any algorithm's output is bounded by the time it is allowed to run. The reason for the latter is that we think that it takes at least some non-zero time for the learning algorithm to read an example.

If there exists an algorithm that satisfies both the above goodness properties, then we deem the learning problem to be "easy" in our parlance.

2.2 Reduction to Prove Hardness

We now discuss the manner in which we say that a learning problem is "hard". It is via a reduction. Our reduction is akin to the polynomial-time Turing reduction [8]. Such a reduction is conjectured to be weaker than the one that is akin to the polynomial-time many-one reduction that some prior work on learning adopts, e.g., the work of Pitt and Valiant [20]. Nonetheless, our reduction derives a contradiction to the claim that both the

following are simultaneously true: **RP** $\neq$ **NP**, and an efficient, PAC-learning algorithm exists for the problem. In other words, our reduction proves that under the assumption **RP** $\neq$ **NP**, an efficient PAC-learning algorithm does not exist for the problem.

2.2.1 Problems, RP and NP

A decision problem is a function whose codomain is $\{\mathsf{true}, \mathsf{false}\}$. For example: given $\langle G, k\rangle$ where $G = \langle V, E\rangle$ is an undirected graph whose set of vertices is $V \neq \emptyset$ and edges is E, and k is an integer such that $k \in \{1, \ldots, |V|\}$, does G have a vertex cover of size k? A vertex cover $C \subseteq V$ is a subset of the vertices such that every edge in E is incident on at least one of the vertices in C. This decision problem, denote it VertexCoverDec, is known to be **NP**-complete [6]. A decision problem is in **NP** if for each true instance of the problem, there exists a string called a certificate whose size is at worst polynomial in the size of the instance which attests to the true-ness of the instance, and a two-input verification algorithm that, given input the instance and certificate, outputs true in polynomial-time. For VertexCoverDec, a natural certificate is a subset $C \subseteq V$ of the vertices that is a vertex cover of G. A corresponding verification algorithm would simply check that $|C| = k$, and that every edge is indeed incident on some vertex in C.

A randomized algorithm is one that is allowed to toss fair coins, where each coin-toss takes some non-zero, constant time. A decision problem is in **RP** if there exists a randomized polynomial-time algorithm that given as input a true instance of the problem, the algorithm outputs true with probability $\geq 1/2$, and given a false instance, outputs false with probability 1. We know that **RP** $\subseteq$ **NP**; the customary conjecture is that **RP** $\neq$ **NP**.

The problem that we call *certificate construction* for a decision problem in **NP** is the following. Given an instance of the decision problem that is known to be true, compute a certificate for it. For example, corresponding to VertexCoverDec and the certificate above that we adopt for it, the certificate construction problem VertexCoverCC is: given $\langle G, k\rangle$ where G has a vertex cover of size k, compute and output such a vertex cover. Via the following theorem, we establish that if there exists a randomized algorithm for VertexCoverCC, then **RP** = **NP**.

Theorem 3. *Suppose there exists a randomized algorithm for* VertexCoverCC *that given input* $\langle G, k\rangle$ *such that* G *has a vertex cover of size* k*, outputs such a vertex cover with probability* $\geq 1/2$ *in polynomial-time. Then,* ***RP*** = ***NP****.*

To prove the above theorem, we adapt a proof from Cormen et al. [2], for the different question as to whether a polynomial-time algorithm exists that correctly outputs true

or false given input an instance of a decision problem, given only an algorithm that is guaranteed to correctly output true give input a true instance of the problem in polynomial-time.

Denote the randomized algorithm A. As A is a randomized polynomial-time algorithm, we know that there exist three positive constants n_0, c, c' such that given an input $\langle G, k \rangle$ whose size is n, where $n > n_0$, A returns a vertex cover for G of size k with probability $\geq 1/2$ in time $\leq c \cdot n^{c'}$ provided G indeed has such a vertex cover.

We construct a randomized polynomial-time algorithm B for VertexCoverDec as follows. The existence of such an algorithm immediately implies **RP** = **NP** because VertexCoverDec is **NP**-complete. If an input instance $\langle G, k \rangle$ of VertexCoverDec is of size $n \leq n_0$, B solves by brute-force and returns. Otherwise, B invokes A with input $\langle G, k \rangle$. If A returns within time $c \cdot n^{c'}$, B checks whether the returned set is indeed a vertex cover of size k for G. If yes, B outputs true and halts. Otherwise, B outputs false and halts. For an input instance of VertexCoverDec, B is guaranteed to output true with probability $\geq 1/2$, and for an input instance that is false, B is guaranteed to output false with probability 1, as desired.

2.2.2 Reductions in our context

Given an access control model M_l for the underlying policy, we reduce as follows. We choose a decision problem, denote it L, that is known to be **NP**-complete, for example VertexCoverDec from above. We produce a policy p_l in M_l that encodes (i) a true instance i of L, and (ii) a certificate c_i for that true instance i. Also, we map the instance i only (and not the certificate) to examples $\langle \text{access-request}, \pi \rangle$ such that knowledge of all the examples implies, at most, knowledge of the problem instance i only, and not the certificate c_i. We then challenge a learning algorithm to produce an output policy p_t in the desired model M_t that is equivalent to the underlying policy p_l, i.e., encodes not only the instance i but also the certificate c_i.

If a learning algorithm is able to produce such an output policy p_t, then that learning algorithm will have computed the certificate that is encoded as part of p_l. However, if there exists an algorithm that can compute such a certificate in an efficient, probably approximately correct manner, then **RP** = **NP** because there would exist a randomized polynomial-time algorithm for the decision problem L that we know is **NP**-complete.

We can produce such an algorithm by adapting arguments from Kearns and Vazirani [14]. A PAC-learning algorithm, denote it A, needs to be able to support any distribution $\mathcal{D}$ under which the examples are chosen. Adopt as the distribution $\mathcal{D}$ the uniform distribution.

Also suppose the set S is the set of all pairs of examples each of the form $\langle$access-request, $\pi\rangle$, and adopt $\epsilon = 1/(2|S|)$. Now, if A errs on even one example in S, i.e., if there exists $\langle a, \pi\rangle \in S$ such that $p_l(a) = \pi \neq p_t(a)$, then the error incurred by the algorithm is $\geq 1/|S| = 2\epsilon > \epsilon$. Thus, p_t must encode the certificate c_i of the true instance i as well, and A is now a randomized polynomial-time algorithm for the certificate construction problem that corresponds to L. And by Theorem 3, this would imply **RP** = **NP**.

2.2.3 Comparison to prior work

As mentioned in Section 2.2, the reduction we use differs from the approach used in prior work of Pitt and Valiant[20]. They employed the polynomial-time many-one reduction, also known as Karp reduction, to prove that learning 3-term DNF boolean formula is hard. In their approach, the examples used are the assignments for the boolean formula.

We use a different reduction method for two main reasons. Firstly, the learning problem from Pitt and Valiant's is restricted that the syntax or encoding of the underlying target and the output should be the same, whereas we do not have such a restriction. Secondly, the relationship between the number of examples and the size of the underlying target varies across our learning problems and their's. In the three cases we consider: learning an access matrix, learning an RBAC, and learning an ReBAC**??** the set of all access requests is, under reasonable assumptions about the encoding of the underlying policy, polynomial in the size of the underlying policy. For instance, in the case of an underlying access matrix $p_l = \langle S_l, O_l, R_l, m_l\rangle$, the size of p_l is polynomial with respect to the number of access requests, which is $|S_l| \cdot |O_l| \cdot |R_l|$. This is under the assumption that m_l is encoded as a one-dimensional array of size $|S_l| \cdot |O_l| \cdot |R_l|$, with elements stored sequentially according to a specific order. However, when learning a 3-term DNF boolean formula using the assignment examples, let us assume the size of a 3-term DNF boolean formula with n literals is polynomial with n (the boolean formula is encoded using binary representation). In this case, the total number of assignments is 2^n, which can be exponential in the size of the boolean formula in the worst case.

Chapter 3

The Access Matrix

We now pose two learning problems for the access matrix. In both, the models for the underlying and output policies are the same, and the examples are $\langle$access-request, allow/deny$\rangle$ pairs. One of the problems turns out to be easy, and the other hard.

3.1 Learning an Access Matrix with positive rights only

In this section, we consider the problem of learning the access matrix with positive rights only. The access matrix is encoded as a tuple $\langle S, O, R, m\rangle$, where S, O, and R refer to subjects, objects, rights, and a matrix, which is a function $m\colon S \times O \to 2^R$. An example of the access matrix is shown in 3.1.

In this learning problem, the models M_l and M_t for the underlying and output policies, respectively, is the access matrix model. Adopt as examples for the learning algorithm pairs $\langle$access-request, response$\rangle$, where an access-request is a triple
$\langle$subject, object, right$\rangle$ and the response is "allow" or "deny", such that the response is "allow" if and only if each of the subject, object and right exist, and the subject has the right over the object in the matrix m. Adopt $M_t = M_l$ as the model for the output of the learning algorithm. We will show that this learning problem is easy.

Theorem 4. *The above learning problem is easy.*

Proof. By construction. Consider the simple learning algorithm, denote it A that builds an output access matrix $p_t = \langle S_t, O_t, R_t, m_t\rangle$ as follows. Initially, S_t, O_t, R_t and m_t are all

	s_1	s_2	s_3	s_4	s_5	o
s_1			✓			✓
s_2				✓		✓
s_3	✓			✓	✓	✓
s_4		✓	✓		✓	
s_5			✓	✓		

Figure 3.1: An access matrix instance containing only positive rights r is represented using the symbol ✓ within the table, whereas figure 3.2 illustrates a different version of the access matrix that has both positive and negative rights.

empty. For every access-request $\langle s, o, r\rangle$ that is allowed, A performs $S_t \leftarrow S_t \cup \{s\}, O_t \leftarrow O_t \cup \{o\}, R_t \leftarrow R_t \cup \{r\}$, and $m_t[s, o] \leftarrow m_t[s, o] \cup \{r\}$ after first creating a row for s and column for o in m_t if either does not already exist. A ignores any access-request that is denied. The Pseudo-code of A is shown in Algorithm 1.

To prove that this algorithm is effective, we observe that p_t can err only in denying some request $\langle s, o, r\rangle$ that is allowed by the underlying policy p_l; any request that is denied by p_l is denied by p_t as well. Now suppose we are given some ϵ, δ and any fixed distribution $\mathcal{D}$ over the access-request set X (Note that in this distribution setting, each access request can be independently drawn from the distribution, and each request can be drawn from the access request set X multiple times. The probability of each example being drawn is not necessarily equal.). And suppose $n = |S_l| \cdot |O_l| \cdot |R_l|$, representing the cardinality of the access-request set X. We observe that n is polynomial in the size of the underlying policy p_l.

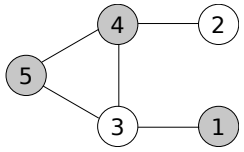
Consider any request, denote it $\alpha = \langle s, o, r\rangle$ that is allowed by p_l but denied by p_t and occurs with probability $\geq \epsilon/n$. If no such request α exists, then the error in $p_t \leq \epsilon$, because the number of possible distinct access requests $\leq n$. We ask: how many examples m suffice so no such request α exists? We observe that after m examples, the probability that any such α has not been seen by the learning algorithm is $\leq n\left(1 - \frac{\epsilon}{n}\right)^m$, where the "$n$" is from the union bound across all possible examples, and the "$\left(1 - \frac{\epsilon}{n}\right)^m$" is the probability that any one particular α has not been seen across m examples. And because $(1 - x) \leq e^{-x}$, for it to be true that $n\left(1 - \frac{\epsilon}{n}\right)^m \leq \delta$, it suffices that $m \geq (n/\epsilon)\left(\ln(n) + \ln(1/\delta)\right)$, which is polynomial in the size of p_l, $1/\epsilon$ and $1/\delta$.

□

Algorithm 1: Learning as an access matrix with positive rights only

Input : E which is a set of examples $\langle \langle s, o, r \rangle, \text{allow} \backslash \text{deny} \rangle$
Output: An access matrix with positive rights

```
1 Initialize ⟨S, O, R, m⟩ to ∅;
2 foreach example ⟨⟨s, o, r⟩, allow\deny⟩ ∈ E do
3 |   if example is "allow" then
4 |   |   if m[s, o] ≠ r then
5 |   |   |   S ← S ∪ s;
6 |   |   |   O ← O ∪ o;
7 |   |   |   R ← R ∪ r;
8 |   |   |   m[s, o] ← r
9 return ⟨S, O, R, m⟩
```



	s_1	s_2	s_3	s_4	s_5	o
s_1	✗		✓			✓
s_2				✓		✓
s_3	✓			✓	✓	✓
s_4		✓	✓	✗	✓	
s_5			✓	✓	✗	

Figure 3.2: An instance of VertexCoverCC and its encoding as an access matrix with positive and negative rights. "✓" represents positive rights r, and "✗" represents negative rights $\widetilde{r}$. The three shaded vertices comprise a vertex cover of size $k = 3$.

3.2 Access Matrix with both positive rights and negative rights

Consider a variant of the access matrix model in which for every "positive" right r, there is a "negative" right $\widetilde{r}$, but an access-request remains the same as we specify for an access matrix with positive rights only. In this variant, a policy is specified by a tuple $\left\langle S, O, R \cup \widetilde{R}, m \right\rangle$, where S, O are the sets of subjects and objects respectively, and the set of rights is $R \cup \widetilde{R}$, where $R \cap \widetilde{R} = \emptyset$ and $r \in R \iff \widetilde{r} \in \widetilde{R}$. The access matrix is $m : S \times O \to 2^{R \cup \widetilde{R}}$. An access-request is a triple $\langle s, o, r \rangle$ as before, where $r \in R$. The request is allowed if and only if $s \in S, o \in O, r \in R$, $r \in m[s, o]$ and $\widetilde{r} \notin m[s, o]$. That is,

we adopt a "deny overrides" discipline in access enforcement.

Adopt as the model for the output policy the same model as the underlying policy, and characterize an error as any difference in the entries of the underlying and output access matrices. That is, if m_l is the access matrix in the underlying policy and m_t in the output policy, given an access-request $\langle s, o, r\rangle$ drawn from some distribution $\mathcal{D}$, an error is the probability that $m_l[s, o] \cap \{r, \widetilde{r}\} \neq m_t[s, o] \cap \{r, \widetilde{r}\}$. (If one of s, o does not exist in the output policy, simply assume $m_t[s, o] = \emptyset$.)

Theorem 5. *The above learning problem is hard.*

Our proof is via reduction from VertexCoverCC, the certificate construction problem for VertexCoverDec (see Section 2.2). Suppose we are given $\langle G, k\rangle$ where $G = \langle V, E\rangle$ is an undirected graph which has a vertex cover of size k, and that $C \subseteq V$ is such a vertex cover. The high idea behind our reduction is as follows: we encode G, k, and C as an underlying policy p_l, and G, k as the set of all access-requests $\langle s, o, r\rangle$. This setting prevents the learning algorithm from obtaining information about the certificate, i.e. the vertex cover. If a learning algorithm that can output a policy which is probably approximately correct to the underlying policy exists, this suggests the existence of a randomized algorithm for the VertexCoverCC problem. Based on Theorem 3, this leads to the conclusion that **RP** = **NP**. (Illustration of the reduction is provided through an example in Figure 3.2.)

Here is the specific setting of the reduction. We encode G, k, C in an underlying policy p_l as follows. Assume the set of vertices in G, $V = \{1, \ldots, |V|\}$. For each vertex $i \in V$, create a subject s_i. Each subject s_i is also an object in p_l. In addition, create another object, denote it o. Our set of positive rights comprises one right r only; thus, our set of negative rights is $\{\widetilde{r}\}$. For each edge $\langle i, j\rangle \in E$, add the right r to $m_l[s_i, s_j]$. For each vertex i in the vertex cover C, add the negative right $\widetilde{r}$ into $m_l[s_i, s_i]$. Finally, for $i = 1, \ldots, k$, add the right r into $m_l[s_i, o]$. (We assume $k \in \{1, \ldots, |V|\}$.)

The set of all access requests is $S_l \times O_l \times R_l$, where $S_l = \left\{s_1, \ldots, s_{|V|}\right\}, O_l = S \cup \{o\}$ and $R_l = \{r\}$. Thus, from the set of all examples, a learning algorithm discovers $\langle G, k\rangle$. Any cell along the diagonal would correspond to "deny" examples only; for example, $\langle\langle s_1, s_1, r\rangle, \text{deny}\rangle$ and $\langle\langle s_3, s_3, r\rangle, \text{deny}\rangle$, and a learning algorithm is unable to distinguish that one has a negative right and the other does not, means that it does not get any information about the certificate that it is requested to output.

3.3 Generalization

A utility of the above two results is that they serve as the basis for other learning problems in access control, as is apparent in the following two chapters. The proof for Theorem 5 serves as a template for proving other hardness results in this context. And Theorem 4 can be generalized to the following, which is for any model for the underlying policy.

Corollary 1. *Suppose we have a model M with the property that for every policy p in M, there exists a policy p_a in the model for an access matrix we characterize in Chapter 1 such that the size of p_a is at worst polynomial in the size of p. Assume also that the examples are $\langle$access-request, allow/deny$\rangle$, and an error is characterized as the probability that an authorization is allowed in p_a but denied in p, or denied in p_a but allowed in p.*

If there are no more constraints on the output policy, then the learning problem is easy.

The statement in the above corollary does not constrain us to a model for the output policy, and therefore, towards a proof, we choose the access matrix that we characterize in Chapter 1. Then, the proof for the corollary is identical to that of Theorem 4, except that we would first adopt the access matrix policy, p_a, in lieu of the original policy p, to carry out the proof.

Chapter 4

Role-Based Access Control (RBAC)

In this chapter, we focus on the learning problem that the underlying policy is encoded as a RBAC model, while the encoding of the hypothesis policy can be vary. RBAC is a kind of access control policy model that is specified by a tuple $\langle U, P, R, UA, PA \rangle$. U, P, R are sets of users, permissions, and roles respectively. $UA : U \rightarrow 2^R \setminus \{\emptyset\}$ represents the relationship between U and P. It is a function that maps a user to a non-empty set of roles. $PA : P \rightarrow 2^R \setminus \{\emptyset\}$ is a function that associates a permission with a non-empty set of roles.

In the rest of this chapter, we propose 5 learning problems and provide the proof of their hardness. For the learning problems that are easy, we also provide efficient algorithms for them. The underlying policy of those 5 learning problems is encoded in RBAC while the encoding of the hypothesis policy is vary. The learning problems are summarized in Table 4.1.

4.1 Learning as an Access Matrix

The first row of Table 4.1 indicates that the desired output model is an access matrix, the examples we present to a learning algorithm consist of tuples containing user-permission pairs along with allow/deny labels, i.e., $\langle \langle \text{user}, \text{permission} \rangle, \text{allow/deny} \rangle$, and the error is characterized as the probability of authorization mismatches. Under these conditions, the learning problem is easy.

This result follows immediately from Corollary 1 in the previous Chapter. The only nuance is that the variant of an access matrix to which an RBAC policy is related poly-

	Examples	Output policy	Hardness
(1)	$\langle$user-permission, allow/deny$\rangle$	equivalent access matrix	easy
(2)	$\langle$user-permission, allow/deny$\rangle$	equivalent RBAC	easy
(3)	$\langle$user-permission, allow/deny$\rangle$	isomorphic RBAC	hard
(4)	$\langle$user-role, allow/deny$\rangle$, $\langle$user-permission, allow/deny$\rangle$	equivalent RBAC	easy
(5)	$\langle$user-role, allow/deny$\rangle$, $\langle$user-permission, allow/deny$\rangle$	isomorphic RBAC	hard

Table 4.1: The hardness of five learning problems for RBAC (From the work of.

nomially is of the form $\langle U, P, m\rangle$, where U, P are the sets of users and permissions from the RBAC policy, and the matrix is a function of the form $m : U \times P \to \{\text{allow}, \text{deny}\}$. However, this variant of the access matrix is, in turn, related polynomially to the variant we characterize in Chapter 1: to map a policy in that variant to this one, we would simply introduce a single right r, adopt as the objects exactly the set of permissions P, adopt as the set of subjects the users U, and add the right r to the cell for $\langle u, p\rangle$ if and only if $m(u, p) = \text{allow}$.

4.2 Learning as an Equivalent RBAC policy

Row (2) of Table 4.1 states that if our desired output model is the RBAC model, the examples we provide a learning algorithm are $\langle\langle \text{user}, \text{permission}\rangle, \text{allow/deny}\rangle$. The label of an example is "allow" if $UA(u) \cap PA(p) \neq \emptyset$, and "deny" otherwise. An error is characterized as the probability of a mismatch in authorizations, and there are no more constraints on the output policy, then the learning problem is easy. By "mismatch in authorizations" for the error, what we mean is the following. Let the underlying RBAC policy and the hypothesis RBAC policy be p_l and p_t respectively, and function $p_l(u, p)$ and $p_t(u, p)$ both return "allow" or "deny". Then, the error is: $\Pr_{\langle u,p\rangle \sim \mathcal{D}} \{p_k(u, p) \neq p_t(u, p)\}$.

The proof is via construction. Denote a learning algorithm as A, and it works as follows. A simply ignores any example whose second component is "deny". For an example $\langle\langle u, p\rangle, \text{allow}\rangle$ that A sees, A checks whether there already exists a role in the output policy

to which the permission p is assigned. If not, A creates a new role r_p and assigns each of u and p to it. If such a role already exists, A assigns u to the role. Thus, in the output policy, there is an invertible mapping between the permissions and roles, and a user to assign to a role r if and only if the user is authorized (in an "allow" example seen by A) to the unique permission p which is assigned to the role. The Pseudo-code of A is shown in Algorithm 2.

It is easy to know that in lines 3-12 of Algorithm 2, the time for algorithm A to process each example is in polynomial. The only question that remains is what the minimum number of examples it needs to see is, given the error and probability parameters ϵ and δ. The proof that it needs to see at worst polynomially many in the size of the underlying policy, $1/\epsilon$ and $1/\delta$ is identical to that of Theorem 4 in the previous Chapter.

Algorithm 2: Learning as an Equivalent RBAC policy

Input : Example $\langle \langle u, p \rangle, \text{allow} \backslash \text{deny} \rangle$
Output: An equivalent RBAC policy

```
1  Initialize ⟨U, P, R, UA, PA⟩ to ∅;
2  foreach example ⟨u, p, allow⟩ do
3      if u ∉ U then
4          U ← U ∪ u;
5      if p ∉ P then
6          P ← P ∪ p;
7      if PA(p) = ∅ then
8          Initialize a new role r_p;
9          PA(p) ← r_p;
10         UA(u) ← r_p;
11         R ← R ∪ r_p;
12     else if PA(p) ≠ ∅ then
13         UA(u) ← PA(p);
14 return ⟨U, P, R, UA, PA⟩
```

4.3 Isomorphic RBAC policy

Row (3) of Table 4.1 addresses learning as an RBAC policy, with the same examples as Row (2). A difference with Row (2) is that we impose the additional constraint that the output policy needs to be *isomorphic* to the underlying policy. Isomorphism, in this context, is

characterized in a natural way, in a manner similar to that for graphs [15]. Given two RBAC policies $\alpha_1 = \langle U_1, P_1, R_1, UA_1, PA_1 \rangle$ and $\alpha_2 = \langle U_2, P_2, R_2, UA_2, PA_2 \rangle$, we say that they are isomorphic to one another if and only if: (i) $U_1 = U_2, P_1 = P_2, |R_1| = |R_2|$, and (ii) there exists an invertible function $g : R_1 \rightarrow R_2$ such that a user u is authorized to a permission p through a role r in the policy α_1 if and only if u is authorized to p through $g(r)$ in α_2; that is, $r \in UA_1(u) \cap PA_1(u)$ if and only if $g(r) \in UA_2(u) \cap PA_2(u)$. Our notion of error for the learning algorithm is the probability, given that examples are drawn under some distribution $\mathcal{D}$, that the output RBAC policy is isomorphic to the underlying policy.

We introduce this notion of isomorphism, and not simply say "the same RBAC policy" to emphasize that the names of roles are inconsequential. Note that there may be good reasons to seek to learn an isomorphic policy. Given a set of user-permission authorizations, there can exist exponentially many (in the size of that set) RBAC policies that encode exactly those authorizations. The question as to which of those RBAC policies are "good" is the topic of role-mining for which there has been considerable prior work [18]. Imposing the additional constraint of isomorphism on a learning algorithm expresses an intent that the goodness of the underlying policy is preserved in the output policy.

In Row (3) of Table 4.1, we assert that this learning problem is hard. Our reduction is similar to the proof for Theorem 5 in the previous Chapter, except that the problem from which we reduce is the certificate construction problem that corresponds to role-mining for the minimum number of roles [4]. That is, the certificate construction problem we adopt is the one that corresponds to the following decision problem: given as input an access matrix of the form $\langle U, P, m \rangle$ as we characterize under our discussions above for Row (1) of Table 4.1, and an integer n, does there exist an RBAC policy as we characterize in Chapter 1 which is equivalent in authorizations to the input access matrix, and whose number of roles is n? This problem is known to be **NP**-complete [4], and is related polynomially to the optimization problem: given as input such an access matrix, compute an equivalent RBAC policy whose number of roles is minimized.

In our reduction, we adopt as an underlying RBAC policy one that minimizes the number of roles for the access matrix that corresponds to that policy. The set of all examples communicates the access matrix only to a learning algorithm.

$\langle\langle \text{alice}, \text{role 1}\rangle, \text{allow}\rangle$
$\langle\langle \text{carol}, \text{role 3}\rangle, \text{allow}\rangle$
$\langle\langle \text{alice}, \text{role 3}\rangle, \text{deny}\rangle$
$\langle\langle \text{alice}, \langle \text{bob}, \text{admin}\rangle\rangle, \text{allow}\rangle$
$\langle\langle \text{alice}, \langle \text{alice}, \text{admin}\rangle\rangle, \text{allow}\rangle$
$\langle\langle \text{alice}, \langle \text{carol}, \text{admin}\rangle\rangle, \text{deny}\rangle$
$\langle\langle \text{carol}, \langle \text{carol}, \text{admin}\rangle\rangle, \text{allow}\rangle$

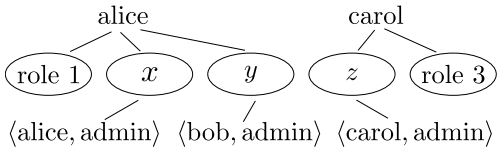


Figure 4.1: Some examples and the output RBAC policy constructed for those by the algorithm we propose for Row (4) of Table 4.1, where the underlying RBAC policy is the one from Figure 1.2.

4.4 Learning as an Equivalent RBAC policy with role-activation

This is the learning problem stated in Row (4) of Table 4.1, which expands the examples, and the notion of error of the PAC model, to incorporate user-role activation.

In systems that adopt RBAC for access control, it is common to require a user to first *activate* a role before they are able to exercise a permission that is assigned to that role. The intent is to provide the user a mechanism by which they can adhere to the principle of least-privilege [21] — in a *session*, a user should assume only those privileges that the user requires, and not necessarily all privileges they can. For example, for the RBAC policy in Figure 1.2, the user carol may activate role 2 in a session, but not role 3, if they do not intend to exercise the $\langle \text{carol}, \text{admin}\rangle$ permission in that session.

Role activation by a user is part of access-enforcement; a user u requests a role r to be activated, and it is allowed if and only if the u is assigned to r in the policy, i.e., $r \in UA(u)$. When u seeks to exercise a permission p, that is allowed if and only if the u has previously activated a role r to which u and p are assigned, i.e., $r \in UA(u) \cap PA(u)$ for some r for which $\langle\langle u, r\rangle, \text{allow}\rangle$ has occurred previously.

For the problem that corresponds to Row (4) in Table 4.1, we are (can be) more demanding with regards to the error when compared to our notion of error for Row (2), given that the learning algorithm is provided the additional information on role-activation. Specifically, we ask not only that user-permission authorizations are the same in the output policy as the underlying policy, but also that user-role assignments are preserved.

More precisely, suppose, given a user u and role r, Event_1 is: $p_l(u, r) \neq p_o(u, r)$, where p_l and p_o are the underlying and the output RABC policies respectively. This is the event

that the user u is assigned to the role r in the underlying policy if and only if u is assigned to r in the output policy. (Note that when we refer to a role that exists in both the underlying and output policies, we mean a role of the same name.) And suppose, given a user u and permission p, Event_2 is: $p_l(u,p) = p_t(u,p)$. This is the event that u is authorized to the permission p in the underlying policy if and only if u is authorized to p in the output policy. Our notion of error for the problem that corresponds to Row (4) of Table 4.1 is the probability that (at least) one of Events (1), (2) does not occur. This probability is under some distribution $\mathcal{D}_r$ under which a user-role pair $\langle u, r\rangle$ is chosen for Event_1, and some distribution $\mathcal{D}_p$ under which a user-permission pair $\langle u, p\rangle$ is chosen for Event (2). We can represent it as:

$$\Pr_{\langle u,r\rangle\sim\mathcal{D}_r,\langle u,p\rangle\sim\mathcal{D}_l}\left\{p_l(u,r) \neq p_o(u,r) || p_l(u,p) \neq p_o(u,p)\right\}$$

In Row (4) of Table 4.1, we state that this learning problem is easy. With some tweaks, the learning algorithm we use for Row (2) can be used as a proof by construction. Specifically we ignore any examples of the form $\langle \cdot, \text{deny}\rangle$. For an example $\langle \langle u, r\rangle, \text{allow}\rangle$ that we see, we create a role r in the output policy if it does not already exist, and assign u to r. For an example $\langle \langle u, p\rangle, \text{allow}\rangle$ that we see, do exactly what the algorithm we propose for Row (2) does. The pseudo-code is shown in Algorithm 3. As with the algorithm for Row (2), the algorithm above is polynomial-time in the number of examples. Also, we can prove that the number of examples we need to see to ensure that we meet the bounds set by ϵ and δ is at worst polynomial in the size of the underlying policy in the same manner as we do for Theorem 4 in Chapter 3.

However, this algorithm can be seen as somewhat unsatisfactory, as it results in roles to which no permission is assigned, and there are roles in the output policy that were never activated. As an example, we show in Figure 4.1 the output policy the algorithm would construct given the examples shown there. It is possible to construct a "better" algorithm that assigns permissions to the roles it sees in $\langle \langle u, r\rangle, \text{allow}\rangle$ examples; we leave a characterization of such an algorithm and the tighter notion of error that it satisfies for future work.

Isomorphic policy with role-activation Row (5) of Table 4.1 addresses the problem of learning an RBAC policy that is isomorphic to the underlying policy (see our characterization under our discussions for Row (3) above), given both user-role and user-permission examples. To show that this problem is hard, it suffices for us to retain the notion of error that we propose for Row (3). That is, our error is the probability that the output policy is isomorphic to the underlying policy, given that examples are drawn under some

Algorithm 3: Learning as an Equivalent RBAC policy

Input : Example $\langle \langle u, p \rangle, \text{allow} \backslash \text{deny} \rangle$ and example $\langle \langle u, r \rangle, \text{allow} \backslash \text{deny} \rangle$
Output: An equivalent RBAC policy

```
Initialize ⟨U, P, R, UA, PA⟩ to ∅;
foreach example ⟨u, r, allow⟩ do
    R ← R ∪ r;
    UA(u) ← r;
foreach example ⟨u, p, allow⟩ do
    if u ∉ U then
        U ← U ∪ u;
    if p ∉ P then
        P ← P ∪ p;
    if PA(p) = ∅ then
        Initialize a new role r_p;
        PA(p) ← r_p;
        UA(u) ← r_p;
        R ← R ∪ r_p;
    else if PA(p) ≠ ∅ then
        UA(u) ← PA(p);
return ⟨U, P, R, UA, PA⟩
```

distribution $\mathcal{D}$.

Our reduction is from the certificate construction problem for vertex cover, VertexCoverCC (see Section 2.2). Note, however, that unlike in Row (3), we need to account for the fact that a learning algorithm may learn user-role assignments. To adapt for this, we adopt a different certificate than we do in the previous section. Suppose we are given an undirected graph $G = \langle V, E \rangle$ which has a vertex cover of size k. Then, a certificate that attests to the fact that G has such a vertex cover is a function $m \colon E \to V$ which identifies, for each edge $e \in E$, the vertex that covers it. That is, if $\langle u, v \rangle = e \in E$, then $m(e) \in \{u, v\}$, and $m(e)$ identifies exactly which of u or v we include in our vertex cover of size k for the edge e. The function m is a valid certificate, and can be encoded, e.g., as a table, whose size is linear in G.

Our reduction is as follows. Assume that we are given $\langle G, k, m \rangle$, where $G = \langle V, E \rangle$ is an undirected graph, $k \in [1, |V|]$ is an integer, and $m \colon E \to V$ is a mapping that encodes a vertex cover of G of size k as we characterize above. We encode $\langle G, k, m \rangle$ in an underlying RBAC policy as follows. We have one user u only. We have k roles, $r_1, \ldots, r_k$, where each

Edge	$m(\cdot)$
$\langle 1, 3 \rangle$	1
$\langle 2, 4 \rangle$	4
$\langle 3, 4 \rangle$	4
$\langle 3, 5 \rangle$	5
$\langle 4, 5 \rangle$	5

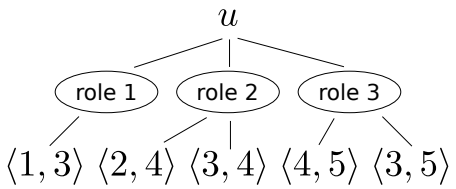


Figure 4.2: The graph and vertex cover of size $k = 3$ from Figure 3.2 encoded as a table m of edges to vertices, and the underlying RBAC policy to which we map, for the reduction for Row (5) of Table 4.1.

r_j corresponds to a vertex v_j in the vertex cover encoded by m, i.e., $\{v_1, \ldots, v_k\}$ is the range of m. For each edge $e_1, \ldots, e_q \in E$, we have a permission, $p_1, \ldots, p_q$. The user u is assigned to all the roles $r_1, \ldots, r_k$. A permission p_i is assigned to a role r_j if and only if $m(e_i) = v_j$, i.e., the edge e_i that corresponds to the permission p_i is covered by the vertex v_j that corresponds to the role r_j.

The set of all user-role examples communicates k to the learning algorithm. The set of all user-permission examples communicates the set of all edges, i.e., the graph G, to the learning algorithm. The challenge for the learning algorithm is to identify the role-permission edges; knowledge of those edges would identify m, and thereby, a vertex cover of size k for G. In Figure 4.2, we show the function m encoded as a table and the underlying RBAC policy to which we map for the graph with $k = 3$ of Figure 3.2.

Chapter 5

Relationship-Based Access Control (ReBAC)

We now address four learning problems in the context ReBAC. The particular model of ReBAC on which we focus is the one on which prior work on learning focuses [13].

The intent of ReBAC is to capture authorizations that result from relationships between entities. In recognition that there can exist different kinds of relationships, a component of a ReBAC policy is a set, L, of relationship *labels*, which we can perceive as a finite set of symbols or an alphabet. Each symbol can usually be thought of as a word or phrase in English, for example, the relationship label "CHILD" corresponds to "child" in English. A *relationship pattern* is a finite string in the alphabet L, i.e., a finite sequence of labels from L, e.g., if $L = \{\alpha, \beta\}$, then $\alpha.\alpha.\beta$ is a relationship pattern, where we use the "." as a separator between labels. A component of a ReBAC policy is a set, P, of such relationship patterns; the significance of a pattern p belonging to this set P is that the only accesses we allow are those between entities that are related by such a pattern p.

Finally, there is a *system graph* G that specifies the entities in the system, and the manner in which they are related. $G = \langle V, E, l \rangle$, where V is the set of entities, E is a set of directed edges, each of which represents a (direct) relationship, and $l \colon E \to L$ is a labelling function, which labels an edge.

A ReBAC policy is characterized as a triple, $\langle G, L, P \rangle$. An access-request is a pair $\langle u, v \rangle$ where $u, v \in V$. It is allowed if and only if there exists a path $u \rightsquigarrow v$ in G of, say, k, edges such that the sequence of labels of edges on that path $\alpha_1.\alpha_2.\ldots.\alpha_k \in P$.

We show an example of ReBAC policy in Figure 5.1. In the example, as the caption of the figure says, the access request $\langle$Bob, Alice-record$\rangle$ is allowed because there is a path

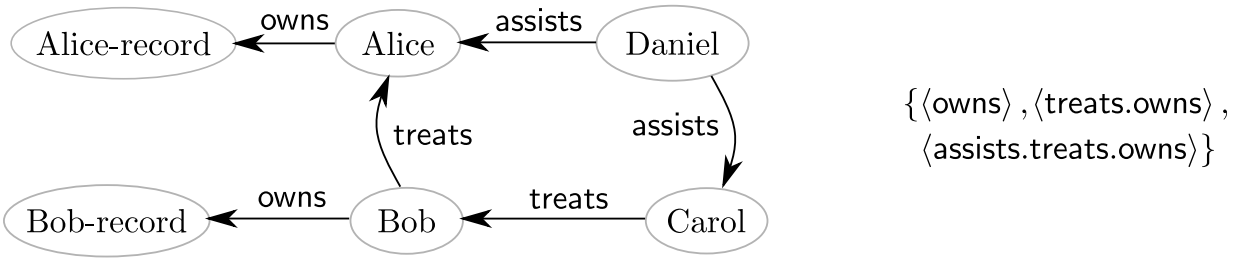


Figure 5.1: An example, adapted from prior work [13], of a ReBAC policy that comprises a system graph and a set of label-sequences. The access-request $\langle \text{Bob}, \text{Alice-record} \rangle$ is allowed; $\langle \text{Daniel}, \text{Alice-record} \rangle$ is denied, as is $\langle \text{Daniel}, \text{Carol} \rangle$.

in the system graph Bob $\xrightarrow{\text{treats}}$ ◯ $\xrightarrow{\text{owns}}$ Alice-record, and the label-sequence treats.owns is in the set P. We render the intermediate vertex on the path as "◯" because we do not care which vertex it is from the standpoint of access-enforcement. The access-request $\langle$Daniel, Alice-record$\rangle$ is denied because given any path Daniel $\rightsquigarrow$ Alice-record in the graph, the sequence of labels on the path is not a member of P. For the same reason, the access-request $\langle$Daniel, Carol$\rangle$ is denied as well.

One more notion to which three of our learning problems in Table 5.1 pertain is what we can think of an "intersection" between G and P in a ReBAC policy, which we denote $G \sqcap P$. It is motivated by the fact that a label-sequence must exist in both G and P for an access-request to be allowed, and also because it underlies prior work that is related to ours [13]. A perspective towards $G \sqcap P$ is as a set of strings, i.e., label-sequences, that is a subset of P. $G \sqcap P$ perceived as a set of strings is a subset $S \subseteq P$ where $s \in S$ if and only if $s \in P$ and a path exists in G with the label-sequence s. For our example in Figure 5.1, such a set of strings is exactly the set P, because every string in P appears in a path of G there.

Another perspective towards $G \sqcap P$ is as a system graph, which is the subgraph of G with the same set of vertices, but in which an edge e exists if and only if any path contains the edge e exists in G and the label-sequence along the path is a member of P. For example, the system graph that corresponds to $G \sqcap P$ in our example of Figure 5.1 would not have the edge Daniel $\xrightarrow{\text{assists}}$ Alice, but otherwise would be the same as G. The reason is that no path that includes that edge has a label-sequence that is in the set of label-sequences P.

We summarize our results for the four learning problems we consider in Table 5.1, and now discuss each in turn.

	Learning objective	Examples	Output policy	Hardness
(1)	Entire policy $\langle G, L, P\rangle$	$\langle\langle u, v\rangle, \text{allow/deny}\rangle$	access matrix	easy
(2)	$G \sqcap P$ as a set of strings	$\langle\langle u, v\rangle, \pi\rangle$	DFA	easy
(3)	$G \sqcap P$ as a set of strings	$\langle\langle u, v\rangle, \pi\rangle$	DFA of min. size	easy
(4)	$G \sqcap P$ as a system graph	$\langle\langle u, v\rangle, \pi\rangle$	System graph	easy

Table 5.1: The hardness of four learning problems for ReBAC. We clarify the notion of error for each, and what we mean by "$G \sqcap P$", in the prose. π denotes a string, i.e., label-sequence.

$\langle\langle \text{Daniel}, \text{Bob-record}\rangle, \text{allow}\rangle$
$\langle\langle \text{Bob}, \text{Alice-record}\rangle, \text{allow}\rangle$
$\langle\langle \text{Daniel}, \text{Alice-record}\rangle, \text{deny}\rangle$

	Alice	Bob	Carol	Daniel	Alice-record	Bob-record
Alice						
Bob					r	
Carol						
Daniel						r
Alice-record						
Bob-record						

Figure 5.2: The output access matrix from our algorithm for Row (1) of Tabel 5.1 for the three examples shown in the figure.

5.1 Learning as an access matrix with allow/deny

In Row (1) of Table 5.1, we address learning an underlying ReBAC policy $p_l = \langle G_l, L_l, P_l\rangle$, where an example is a $\langle \text{access-request}, \text{allow/deny}\rangle$ pair, i.e., $\langle\langle u, v\rangle, \text{allow/deny}\rangle$, where u, v are vertices in the underlying system graph G_l. Our characterization of error is a mismatch in authorizations between the output policy and underlying policy. We ask whether we can learn the underlying policy as an access matrix, and otherwise, do not impose any constraints.

This learning problem is easy; Corollary 1 in Chapter 3 applies. That is, given any ReBAC policy $R = \langle G, L, P\rangle$, there exists an encoding of it as an access matrix M such that the size of M is at worst polynomial in the size of R. If we adopt the variant of the access matrix from Chapter 1, M simply adopts as its set of subjects and objects the set of vertices V in G. There is only one right, denote it r, and we add this right to a cell $\langle u, v\rangle$ in the matrix if and only if u has access to v as per the access-enforcement rules of ReBAC. Thus, the size of M is at worst quadratic in the size of R.

$\langle\langle$Daniel, Bob-record$\rangle$, $\langle$assists.treats.owns$\rangle\rangle$
$\langle\langle$Bob, Alice-record$\rangle$, $\langle$treats.owns$\rangle\rangle$

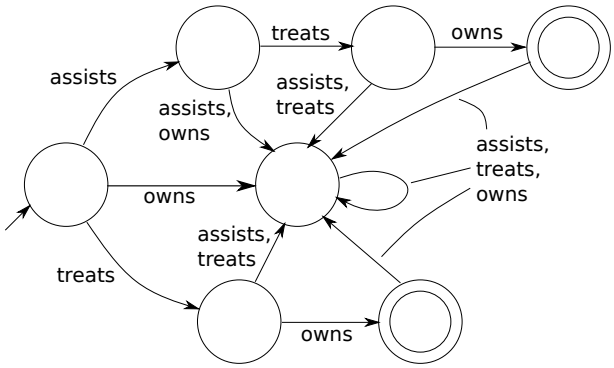


Figure 5.3: The output DFA from our algorithm for Row (2) of Table 5.1 for the two examples shown in the figure. An accepting state is shown with a double circle; the initial state is to the far left.

In figure 5.2, we show the access matrix learned by the algorithm we propose above using three examples, where the underlying policy is the one from Figure 5.1.

5.2 Learning $G \sqcap P$ as a DFA

In Row (2) of Table 5.1, we address learning the set of strings that corresponds to $G_l \sqcap P_l$; that is, the set of strings in the alphabet L_l each of which exists both as a sequence of edge-labels in G_l, and is a member of the set of label-sequences P_l.

Our intent with the learning problems of Rows (2) and (3) is to pose, in our framework, the learning problem of prior work [13]. With that in mind, an example is a pair $\langle\langle u, v\rangle, \pi\rangle$, where $\langle u, v\rangle$ is an access-request, i.e., a pair of vertices, and π is a label-sequence, i.e., a string in the alphabet L_l with the following property. If the access-request $\langle u, v\rangle$ is allowed, then π is a label-sequence that is a member of P_l and a sequence of edges in a path $u \rightsquigarrow v$ in G_l. If the access-request $\langle u, v\rangle$ is denied, then π is the special symbol $\phi \notin L_l$. For the ReBAC policy in Figure 5.1, the following are some examples that a learning algorithm may be provided: $\langle\langle$Bob, Alice-record$\rangle$, treats.owns$\rangle$, $\langle\langle$Daniel, Alice-record$\rangle$, $\phi\rangle$, $\langle\langle$Daniel, Carol$\rangle$, $\phi\rangle$ and $\langle\langle$Bob, Bob-record$\rangle$, owns$\rangle$.

Thus, the examples provide richer information than the examples we adopt for the problem in Row (1) of Table 5.1. We can certainly infer from the second component π as to whether an access-request $\langle u, v\rangle$, is allowed or denied. In addition, we get the label-sequence π which is presumably computed as part of access-enforcement, because we need to check that such a label-sequence exists in both a sequence of edges in G_l and

in the set P_l, and a straightforward way of doing that is by generating π as part of the access-enforcement process.

Our output policy for Row (2) is a DFA. Our notion of error is natural: it is the probability that either the output DFA D_t rejects a string that is in the set of strings that corresponds to $G_l \sqcap P_l$, or it accepts a string that is not in the latter set.

The proof that this learning problem is easy is similar to that of Theorem 4 in Chapter 3. Specifically, we propose the following algorithm. We start with D_t as a DFA that accepts no strings — e.g., D_t has one state only, which is the initial state and non-accepting; every transition on any symbol is back to that state. For every $\langle \langle u, v \rangle, \pi \rangle$ that we see where $\pi \neq \phi$, we "grow" D_t to accept the string π. A way to grow D_t in this manner is to simply add at most as many states as symbols in π. We say "at most as many" and not "exactly as many" because there may already exist transitions in D_t to states that are not the initial state for some prefix string of π. Thus, the size of D_t is guaranteed to be at-worst polynomial in the size of the set of strings that corresponds to $G_l \sqcap P_l$, where our characterization of the size of the latter is the sum of lengths of each of its members. The pseudo-code of the above algorithm is shown in Algorithm 4

Algorithm 4: Learning $G \sqcap P$ as an DFA

Input : Example $\langle \langle u, v \rangle, \pi \rangle$
Output: A DFA
Initialize a DFA D_t to $\emptyset$;
Add a dead state s_ϕ to D_t;
foreach *example* $\langle u, v, \pi = \{\pi[1], \pi[2], ..., \pi[n]\} \rangle$ **do**
    foreach $\pi[i] \in \pi$ **do**
        if *the transitions from state* $s_{\pi[i-1]}$ *to the state* $s_{\pi[i]}$ *does not exist* **then**
            break
Create $n - i$ states $\{s_{\pi[i-1]}, ..., s_{\pi[n]}\}$ as symbols in $\{\pi[i], ..., \pi[n]\}$;
Connect the goal state $s_{\pi[n]}$ to s_ϕ whose transition Condition contains all relationship labels from $G_l \sqcap P_l$;
return D_t

In Figure 5.3, we show the DFA of this learning algorithm that has seen the examples shown there, where the underlying policy is the one from Figure 5.1.

The only possible errors in D_t are that it may reject strings that are in the set of strings that correspond to $G_l \sqcap P_l$; any string not in the latter set is guaranteed to be rejected by D_t. The only remaining question regards the minimum number of examples our learning

algorithm needs to be provided to guarantee that we meet the ϵ and δ bounds. In exactly the manner we establish in the proof for Theorem 4, we can establish that this number is at worst polynomial in the size of the underlying policy; more specifically, the size of the set of strings that corresponds to $G_l \sqcap P_l$.

5.3 Learning $G \sqcap P$ as a min. DFA

In Row (3) of Table 5.1, we address the same problem as Row (2), except that we additionally demand that the output DFA must be one of minimum size, i.e., has the fewest number of states across all DFAs that accept exactly the set of strings that corresponds to $G_l \sqcap P_l$, and rejects all other strings. The notion of error is then expanded to: the probability that our output DFA (i) either accepts a string not in S_l or rejects a string in S_l, or, (ii) is not of minimum size.

Our algorithm that establishes that this learning problem is easy as well adds to the algorithm we propose for Row (2) above. Given the set of examples, we first construct a DFA as we say for Row (2). We then employ a well-known, efficient algorithm for minimizing the number of states with that DFA as input [10].

We have already established for Row (2) that we can efficiently meet the ϵ and δ bounds for an error of type (i). For an error of type (ii), we first observe that the only way our output DFA, denote it D_t, is not of minimum size is if D_t does not accept a string that is in the set of strings that corresponds to $G_l \sqcap P_l$. Therefore:

$$\begin{aligned} & \Pr_{\langle\langle u,v\rangle,\pi\neq\phi\rangle\leftarrow\mathcal{D}}\{D_t \text{ rejects } \pi\} \leq \epsilon \\ \Longrightarrow & \\ & \Pr_{\langle\langle u,v\rangle,\pi\neq\phi\rangle\leftarrow\mathcal{D}}\{D_t \text{ is not minimum-sized}\} \leq \epsilon \end{aligned}$$

That is, guaranteeing that we meet the ϵ and δ bounds for an error of type (i) is a sufficient condition to guarantee that we meet the ϵ and δ bounds for an error of type (ii). Figure 5.4 shows a DFA of minimum size (number of states) that corresponds to the DFA of Figure 5.3.

5.3.1 Reconciling the work of Gold [7]

Our "easy" result for the problem of Row (3) of Table 5.1 may appear to be at odds with the result of Gold [7], from which we can infer that learning a DFA of minimum-size which

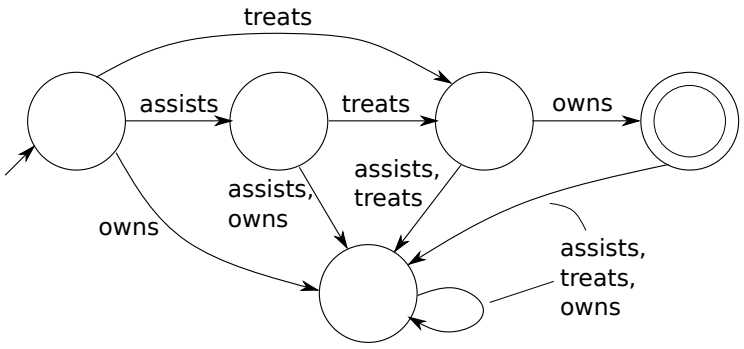


Figure 5.4: A DFA of the minimum number of states corresponding to the DFA of Figure 5.3.

is consistent with a given set of positive and negative examples is (NP-)hard — a positive example is a string the DFA must accept; a negative example is one it must reject. The reason is that our problem and that one are different. In our problem, the input is one set of strings only, and our demand is for a DFA of minimum size that accepts all strings in the set and rejects all other strings. In the corresponding problem as it pertains to the work of Gold [7], the input is two sets of strings, one of which must be accepted and the other rejected. Also, most importantly, the union of the two sets is not necessarily the set of all strings. Thus, the problem in that work introduces a "degree of freedom" for the learning algorithm in that a string in the underlying alphabet may appear in neither of the two sets that are the input, and the algorithm must choose whether its DFA accepts or rejects such a string; the choice can determine whether the output DFA indeed minimizes the number of states. In our problem, no such degree of freedom exists for the learning algorithm.

5.3.2 Comparison to prior work [13]

The learning problem we consider in Row (3) of Table 5.1 is the same as that of prior work [13]. That work adopts the rather different learning model of Angluin [1]. In that learning model, the learning algorithm gets to choose the distribution under which examples are provided to it; in our case, an algorithm needs to support any distribution. However, in that work, an example that is provided to the learning algorithm may be a false-positive; that is, the learning algorithm may be provided as an example a string that is not a member of the set of strings it seeks to learn. However, that learning model allows for so-called equivalence queries, whereby the learning algorithm is able to query an oracle with the

$\langle\langle$Bob, Bob-record$\rangle$, owns$\rangle$
$\langle\langle$Daniel, Bob-record$\rangle$, $\langle$assists.treats.owns$\rangle\rangle$
$\langle\langle$Bob, Alice-record$\rangle$, $\langle$treats.owns$\rangle\rangle$

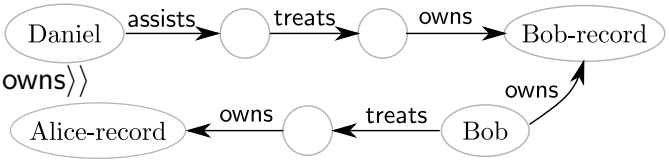


Figure 5.5: The system graph for the examples shown above that our algorithm for Row (4) of Table 5.1 outputs. What we call anonymous vertices are shown as circles without names.

DFA it has constructed so far, and ask whether there exists a counterexample, i.e., a string that the DFA accepts that it should reject.

Thus, the two learning models are qualitatively different. Whether one learning model is more appropriate than the other in a particular setting would depend on which of the two sets of assumptions can be satisfied in that setting. It is unclear whether the learning model from prior work [13] can be used to infer computational hardness results in the manner we do in this work.

5.4 Learning as a system graph

As we say at the start of this section, $G_l \sqcap P_l$ can be perceived as a system graph, denote it $\widetilde{G_l}$, which is a subgraph of G_l whose only paths are those with label-sequences that are members of P_l. As a final learning problem, in Row (4) of Table 5.1, we address learning $\widetilde{G_l}$ as a graph. We adopt as our examples the same as those for Rows (2) and (3), i.e., tuples of the form $\langle\langle u, v\rangle, \pi\rangle$. Suppose the output system graph is denoted $\widetilde{G_t}$. We adopt as our notion of error the probability that there exists a path in $\widetilde{G_l}$ with a label-sequence for which no path exists in $\widetilde{G_t}$, or a path in $\widetilde{G_t}$ with a label sequence for which no exists in $\widetilde{G_l}$, i.e. $\Pr_{\langle\langle u,v\rangle,\pi\rangle\sim\mathcal{D}}\left\{\widetilde{G_l}(u,v) \neq \widetilde{G_t}(u,v)\right\}$.

This learning problem is easy, as we say for Row (4) in Table 5.1. An algorithm, which sees an example $\langle\langle u, v\rangle, \pi\rangle$ where $\pi \neq \phi$ simply adds a path with the labels in π, with "anonymous" intermediate vertices. We show in Figure 5.5 the output graph for the examples we show there. We can prove, exactly in the same way that we do for Theorem 4 in Chapter 3, that the number of examples we need to see to meet the ϵ and δ bounds is at worst polynomial in the size of the underlying policy $\langle G_l, L_l, P_l\rangle$.

Chapter 6

Related Work

Our work addresses learning access control policies under a slight generalization of the PAC-learning framework. As such, work that is closely related to ours are those on PAC-learning, access control and those at the intersection of access control and machine learning.

PAC-learning was, to our knowledge, first characterized by the work of Valiant [23]. The book of Kearns and Vazirani [14] provides an excellent overview. Of particular interest to us is the manner in which one establishes that a problem lends itself to an efficient learning algorithm, i.e., is "easy" in our parlance, or is unlikely to, i.e., is "hard". Towards the former, the book provides a few examples and proofs, which we have leveraged, e.g., to prove Theorem 4 in Chapter 3. Towards the latter, we rely on a version of the polynomial-time Turing (or Cook) reduction, which is conjectured to be weaker than the more customary polynomial-time many-to-one reduction [8]. Nonetheless, it is evidence of computational hardness. An example of the latter kind of reduction in the context of PAC-learning is in the work of Pitt and Valiant [20]. For our proofs of hardness, we have leveraged the well-known problem of computing a vertex cover of a given size [6] and an RBAC policy of a certain number of roles given an access matrix as input [4], each of which is known to be **NP**-complete.

The access control policy models to which our work refers are the access matrix [9], RBAC [22] and ReBAC [5]; for ReBAC, the model we address is one that prior work on learning has adopted [13].

As for the intersection of machine learning and access control, there are already considerably many pieces of work, as the survey of Nobi et al. [19] discusses. None, to our knowledge, addresses computational hardness as we do. The work that is closest to ours

is that of Iyer and Masoumzadeh [13], which addresses learning a ReBAC policy under a different learning model than ours, as we discuss in Chapter 5 above.

Chapter 7

Conclusions and Future Work

We have addressed several learning problems in the context of access control across three models: the access matrix, RBAC and ReBAC. Our setting is one in which there is an underlying access control policy in a particular model, and the learning algorithm seeks to output an equivalent policy in the same or a different model. We consider different kinds of examples a learning algorithm is provided; however, in all cases, our examples are data generated during the process of access-enforcement, when an access-request is made that is evaluated against the underlying policy. The framework we adopt which allows us to infer computational hardness results is slightly generalized PAC-learning. Of the two problems we consder for the access matrix, one is computationally easy and the other is hard. Of the five problems we consider for RBAC, two are hard and the others are easy. All the four problems we consider for ReBAC are easy. An advantage with an "easy" result is that the proof is constructive: in each case we propose a learning algorithm that is not only polynomial-time in the size of the underlying policy and the inverse of the error parameters ϵ and δ, but also probably-approximately correct, i.e., meets the bounds prescribed by the parameters ϵ and δ.

There is tremendous scope for future work. One is to consider more access control models; a natural candidate is Attribute-Based Access Control (ABAC) [11], for which there is prior work on policy-mining [3, 24], which can be seen as a counterpart of learning. Another is to propose algorithms for the learning problems that turn out to be easy that are superior to our algorithms in the kinds of errors they can tolerate. For example, we comment in this work that the algorithm we propose for the learning problem of Row (4) in Table 4.1 in Chapter 4 may be deemed to be unsatisfactory. Devising an algorithm that satisfies a tighter notion of error, e.g., that in the output policy a role must be assigned to at

least one user and at least one permission, or showing that the problem is computationally hard, would be interesting future work.

References

[1] Dana Angluin. Learning regular sets from queries and counterexamples. *Information and Computation*, 75(2):87–106, 1987.

[2] Thomas Cormen, Charles Leiserson, Ronald Rivest, and Clifford Stein. *Introduction to Algorithms, Fourth Edition*. MIT Press, 2022.

[3] Carlos Cotrini, Thilo Weghorn, and David Basin. Mining abac rules from sparse logs. In *2018 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 31–46. IEEE, 2018.

[4] Alina Ene, William Horne, Nikola Milosavljevic, Prasad Rao, Robert Schreiber, and Robert E. Tarjan. Fast exact and heuristic methods for role minimization problems. In *Proceedings of the 13th ACM Symposium on Access Control Models and Technologies*, SACMAT '08, pages 1–10, New York, NY, USA, 2008. Association for Computing Machinery.

[5] Philip W.L. Fong. Relationship-based access control: Protection model and policy language. In *Proceedings of the First ACM Conference on Data and Application Security and Privacy*, CODASPY '11, pages 191–202, New York, NY, USA, 2011. Association for Computing Machinery.

[6] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[7] E Mark Gold. Complexity of automaton identification from given data. *Information and Control*, 37(3):302–320, 1978.

[8] Oded Goldreich. *Computational Complexity: A Conceptual Perspective*. Cambridge University Press, 2008.

[9] Michael A. Harrison, Walter L. Ruzzo, and Jeffrey D. Ullman. Protection in operating systems. *Commun. ACM*, 19(8):461–471, aug 1976.

[10] John E Hopcroft, Rajeev Motwani, and Jeffrey D Ullman. Introduction to automata theory, languages, and computation. *Acm Sigact News*, 32(1):60–65, 2001.

[11] Vincent C. Hu, David Ferraiolo, Rick Kuhn, Adam Schnitzer, Kenneth Sandlin, Robert Miller, and Karen Scarfone. Guide to attribute based access control (abac) definition and considerations. NIST Special Publication 800-162, https://doi.org/10.6028/NIST.SP.800-162, 2014.

[12] Padmavathi Iyer and Amirreza Masoumzadeh. Active learning of relationship-based access control policies. In *Proceedings of the 25th ACM Symposium on Access Control Models and Technologies*, SACMAT '20, pages 155–166, New York, NY, USA, 2020. Association for Computing Machinery.

[13] Padmavathi Iyer and Amirreza Masoumzadeh. Learning relationship-based access control policies from black-box systems. *ACM Transactions on Privacy and Security*, 25(3):1–36, 2022.

[14] Michael J Kearns and Umesh Vazirani. *An introduction to computational learning theory*. MIT press, 1994.

[15] Johannes Kobler, Uwe Schöning, and Jacobo Torán. *The graph isomorphism problem: its structural complexity*. Springer Science & Business Media, 2012.

[16] Ha Thanh Le, Cu Duy Nguyen, Lionel Briand, and Benjamin Hourte. Automated inference of access control policies for web applications. In *Proceedings of the 20th ACM Symposium on Access Control Models and Technologies*, SACMAT '15, pages 27–37, New York, NY, USA, 2015. Association for Computing Machinery.

[17] Amirreza Masoumzadeh. Inferring unknown privacy control policies in a social networking system. In *Proceedings of the 14th ACM Workshop on Privacy in the Electronic Society*, WPES '15, pages 21–25, New York, NY, USA, 2015. Association for Computing Machinery.

[18] Barsha Mitra, Shamik Sural, Jaideep Vaidya, and Vijayalakshmi Atluri. A survey of role mining. *ACM Comput. Surv.*, 48(4), feb 2016.

[19] Mohammad Nur Nobi, Maanak Gupta, Lopamudra Praharaj, Mahmoud Abdelsalam, Ram Krishnan, and Ravi Sandhu. Machine learning in access control: A taxonomy and survey. https://arxiv.org/abs/2207.01739, 2022.

[20] Leonard Pitt and Leslie G Valiant. Computational limitations on learning from examples. *Journal of the ACM (JACM)*, 35(4):965–984, 1988.

[21] Jerome H. Saltzer. Protection and the control of information sharing in multics. *Commun. ACM*, 17(7):388–402, jul 1974.

[22] R.S. Sandhu, E.J. Coyne, H.L. Feinstein, and C.E. Youman. Role-based access control models. *Computer*, 29(2):38–47, 1996.

[23] Leslie G Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, 1984.

[24] Zhongyuan Xu and Scott D Stoller. Mining attribute-based access control policies. *IEEE Transactions on Dependable and Secure Computing*, 12(5):533–545, 2014.