Delegation of access rights in a privacy preserving access control model

Md. Moniruzzaman
Dept. of Computer Science
University of Calgary, Calgary, AB, Canada
Email: mmoniruz@ucalgary.ca

Ken Barker
Dept. of Computer Science
University of Calgary, Calgary, AB, Canada
Email: kbarker@ucalgary.ca

Abstract – Delegation is a process of sharing access rights by users of an access control model. It facilitates the distribution of authorities in the model. It is also useful in collaborative environments. Despite the advantages, delegation may have an impact on the access control model’s security. Allowing users to share access rights without the control of an administrator can be used by malicious users to exploit the model. Delegation may also result in privacy violations if it allows accessing data without the data provider’s consent. Even though the consent is taken, the privacy can still be violated if the data is used differently than the data provider agreed. Our work investigates data privacy in delegation. As a contribution, a privacy model is introduced that allows a data provider setting privacy policies that state how their data should be used by different organizations or parties who are interested in their data. Based on this setting, a delegation model is designed to consider the privacy policies in taking delegation decisions and also, to set the data usage criteria for the access right receivers. In addition to privacy policies, several delegation policies and constraint have been used to control delegation operations. Delegation is studied within a party and between two parties.

Keywords – Delegation, privacy, access control, security, policy.

I. INTRODUCTION

In typical access control models, the set of access rights a user gets is predetermined. Predetermining a user’s access rights is equivalent to anticipating possible usages of the system by that user. However, users may need new access rights due to the dynamic nature of their work. There are two ways to assign access rights. First, a system administrator acts every time a user needs an access right. Secondly, a user gets the right from another user who already possesses it. The latter approach is called delegation.

Delegation brings flexibility to access control models. Zhang et al. [22] identify three cases when delegation is necessary. In the first, an individual is absent from their job and so, someone else should carry out the tasks. Secondly, delegation is allowed to decentralize the authority. Having one system administrator who assigns access rights to all the users in the system would decrease efficiency. In the final case, delegation is very useful in an environment where users collaborate with each other to complete a task.

Despite its usefulness, delegation may produce security risks for an organization. Consider the case where a system administrator does not assign some privileges to a user for security reasons. In a delegation enabled system, it may not be sufficient for security protection as the user may receive the privileges from other users. Delegation may also lead to data privacy violations. Delegation invites new users to access data which raises the question of whether the data provider’s consent is taken. Even if the provider is informed, the issue of how data will be used is also critical. Any of these issues may violate data privacy if they are not resolved. The security requirement in delegation mainly comes from an enterprise’s perspective while data privacy protection in delegation is required by the data provider. This work investigates data privacy protection in delegation.

To study privacy preserving delegation, we need an environment where data providers provide privacy policies for their data. The policies would state who can use the data and how the data should be used. The access control models in such environments control data accesses based on the privacy policies. These are known as privacy preserving access control models. Several models have been proposed [23, 26] in the literature. Most of the proposed models assume that data is accessed only by the collecting organization. However, in real life, many parties are interested in data including the collecting organization. One of our contributions is to define a privacy model that allows a data provider to set privacy policies for different organizations accessing their data.

We adapt an existing access control model called P-RBAC [14] which is an extension of role-based access control model. The proposed privacy model is used in conjunction with the access control model to create privacy-aware access rights i.e., the rights containing constraints that specify the valid use of data. Data users assigned to these rights use data according to the constraints. Based on these foundations, we propose a delegation model where access rights to a data item can be delegated only if it is allowed by the data item’s privacy policies. Since the delegated rights contain privacy constraints, the users who receive the rights are bound by the privacy constraints when they use the data. We also investigate prohibiting certain delegation operations to maintain the access control model’s security.

The paper is organized in the following ways. Section II describes the privacy model and the access control model. Delegation model is presented in Section III. Revocation is described in Section IV. An overview of the relevant literatures is given in Section V. Conclusion and future works are discussed in Section VI.

II. ACCESS CONTROL MODEL

In this section, we first propose a formal model to express privacy policies. How access rights or privileges are
created based on the privacy policies are described in the privilege model. Finally, we describe how access control decisions are taken.

A. Privacy model

Privacy policies are the data provider’s preferences for using their data. We can also say that privacy policies define the data collector’s practice with the data. This work assumes that a data collector sets the privacy policies for the data items that they collect. Data providers accept the policies if they agree. In addition, the collector may allow the data providers to customize or even opt out from some policies. Examples of such practice include the privacy statements of Amazon.com [1], Toys.com [19], etc.

In this work we formalize the privacy policies. There are several works that define the contents of the privacy policies [23]. Rather than proposing a new one, we choose several works that define the contents of the privacy policies. Examples of such practice include the privacy statements of Amazon.com [1], Toys.com [19], etc.

Data providers may be allowed to specify their preference about particular policies. For example, a privacy policy may state that the consent for using personal information for marketing purpose is optional and one can opt out from such use of their data. To support this in the privacy model, we introduce the notion of privacy policy metadata which are the data provider’s preferences and other related information necessary to enforce privacy policies. Note that one of the elements of a privacy policy is condition that checks the values of these metadata. The metadata are stored in a set of context variables denoted by $SV$. Let the set of data providers be $dProvider$, then the set of privacy metadata for visibility $i \in VIS$ is defined as follows:

$$StatDS_i \subseteq dProvider \times PPolicy_i \times \wp(SV)$$

Here, $\wp$ denotes power set.

Example of the privacy model with a sample policy:
The example uses a real life privacy statement taken from the website Toys.com that sells toys online.

“From time to time, you may receive periodic mailings, telephone calls or e-mails from "R" Us Family members with information on products or services, discounts, special promotions, upcoming events or other offers from an "R" Us Family member or its marketing partners. You may opt out of receiving e-mail communications by clicking the link at the bottom of the e-mail received...” [19]

The policy allows the website’s employees (mentioned as "R" Us Family members) to access the customers’ email addresses (along with other media) for sending promotional offers. The policy also says that if a customer opts out, their data will not be used for this purpose. Note that this policy does not impose any obligation. As the data is being used by the website’s employees, let the visibility be termed as website (denoted by the acronym $ws$). Table 1.1 presents all types of information collected about the customers with a sample record. Table 1.2 is an instance of the relation $PPolicy_{ws}$ that stores the above privacy statement by breaking it into several formal policies. The conditions of the policies test the value of a variable called $OptOut$ which represents the opt in/out preferences of the customers. Table 1.3 is an instance of the relation $StatDS_{ws}$ and stores the values of the variable $OptOut$. When the variable has the value ‘Y’ for a particular data provider and a policy, it indicates that the provider has opted out from the policy.

We consider a hierarchical relation among the purposes. Purpose hierarchy is denoted by $(P, \preceq)$ where $\preceq$ is a partial relation defined over the set of purposes $P$. The relation is reflexive, transitive and anti-symmetric. Fig. 1 shows a sample purpose hierarchy. In our interpretation of the hierarchy, any node includes all of its connected predecessors.
For example, Sales includes Business operation which also represents (Sales ⊆ Business operation). The use of purpose hierarchy makes the policies more expressive. Consider the policy #4 in Table 1.2 which is (Email address, read, Promotion, OptOut ≠ Y, Ø). It allows accessing email address for purpose Promotion as well as for purposes Contest and eMarketing as both of them include Promotion in Fig. 1. Condition and obligation for these purposes are the same as for purpose Promotion.

A policy can be overridden by a more restrictive policy. Consider the following policy.

Privacy policy #5:

(Email address, read, eMarketing, OptOut ≠ Y ∧ Age > 18, Ø)

It states that to use email address for purpose eMarketing, two conditions must be true – data providers have not opted out from the policy and their age must be greater than 18 years. This policy overrides policy #4 by increasing the data usage restriction for purpose eMarketing. However, to use email address for other purposes (Promotion and Contest), only one condition needs to be satisfied as stated by policy #4.

B. Privilege model

Typically, a privilege consists of data and action. In a privacy preserving access control model, privileges also contain privacy restrictions specifying the valid use of data items. These privileges are created from the privacy policies. For instance, a privilege consists of data, action, purpose, condition and obligation in P-RBAC [14]. We adapt this structure to propose a more simple privilege structure. Like P-RBAC, we also group the privileges into roles which are in turn assigned to users.

When an enterprise creates privileges for the data users (e.g., its employees), it may create privileges more restrictive than the privacy policies. If a privacy policy allows using a data item for a set of purposes, the enterprise may choose a subset of the allowed purposes to create a privilege. We define purpose range to specify a subset of purposes in a privilege. A purpose range is given by 

\[ pr = < p_u, p_b > \] 

where \( p_b, p_u \in P \) and \( p_b \leq p_u \). If \( p_u = p_b \), the range has only one purpose. The set of privileges, \( DP \), are defined over the following range:

\[ DP \subseteq D \times ACT \times PR \]

Here, \( D, ACT, \) and \( PR \) are the sets of data, actions and purpose ranges, respectively. As an example, \( dp = (Email address, read, (Promotion, eMarketing)) \).

Users assigned to this privilege can use the data for these purposes \{Promotion, eMarketing\}. What condition and obligation users should fulfill will depend on the purpose they use to access data. The system will find the appropriate condition and obligation based on a user’s access purpose.

C. Formal specification of the access control model

The access control model uses the following entities:

- \( VIS \) is the set of visibilities
- \( D, ACT, P, C \) and \( OB \) are the sets of data, actions, purposes, conditions and obligations, respectively.
- \( U, R \) and \( DP \) are the sets of users, roles and privileges.

The dot operator indicates a specific component of a privilege; e.g., \( dp.d \) denotes the data contained in the
privilege $dp$. Following components are defined for a visibility $i \in VIS$:
- $U_i$, $R_i$ and $DP_i$ are the sets of users, roles and privileges for visibility $i$
- Role hierarchy is $(R_i, \preceq_{R_i})$ where $\preceq_{R_i}$ is a partial relation defined over the set of roles $R_i$. The relation is reflexive, transitive and anti-symmetric.
- Role-privilege assignment relation, $RDP_i \subseteq R_i \times DP_i$
- User-role assignment relation, $UAR_i \subseteq U_i \times R_i$
- $hasRights(u, PrivS)$ is a predicate that returns the privileges assigned to a user $u$ by instantiating the set $PrivS$. The predicate also returns true if the instantiation is successful.

$$hasRights: U \times \theta(DP) \rightarrow \mathbb{B}$$

The validation process of access requests is described as follows:
- Users submit data, action and purpose as the access request. Formally, access request $oar = \langle d, a, p \rangle$ where $d \in D, a \in ACT, p \in P$.
- A predicate used in the access request validation is described as follows.

$$ComplianceAC(dp, oar):$$
- The predicate accepts two arguments – privilege $(dp)$ and access request $(oar)$. Let the purpose range of privilege $dp$ is denoted by $pr_{dp}$ as well as $pr_{dp}, pr_u$ be the upper bound and $pr_{dp}, pr_u$ be the lower bound of the range. The predicate evaluates the following logic expression and returns a Boolean value.

$$oar. d = dp. d \land oar. a = dp. a \land pr_{dp}. pr_u \leq oar. p \land oar. p \leq pr_{dp}. pr_u$$

When the predicate returns true, we say that the privilege complies with or contains the access request. Suppose, a user $u$ submits an access request $oar$; the request is granted if the following logic expression is true:

$$hasRights(u, PrivS) \land \exists dp \in PrivS. ComplianceAC(dp, oar)$$

The expression tests if the requesting user is assigned a privilege that contains the access request.

The amount of data that is revealed to the user depends on the condition of the associated privacy policy. For instance, a condition $OptOut \neq Y$ will rule out the data of those providers who have opted out of the privacy policy. Therefore, another predicate is needed which will find the associated privacy policy for the access request and return the condition. In addition, obligation of the policy must also be performed. Details of the condition and obligation enforcements are beyond the scope of this work. The research works [15-16] give more insight to these areas.

III. DELEGATION

Data collector may share data with unknown parties if they do not follow the privacy policy. In the proposed model, delegation follows privacy policy which allows only legitimate parties accessing the data. It also sets the data usage guidelines for them. We refer the data sharing between two parties as *inter-visibility delegation*. The party or visibility which shares data is called *source visibility* while the visibility that receives data is called *destination visibility*. In addition, we study *intra-visibility delegation* where two users within a party share the access rights with each other. Users who delegate the rights are called *delegators* while users who receive the rights are called *delegatees*.

Delegation policies are the rules that state what delegation operations are valid. These rules are used to control delegation among the users. In this work, we define delegation policies for intra- and inter-visibility delegation. Besides the delegation policies, a security policy is used to maintain the separation of duty among the users. The security policy is often referred as the security constraint to differentiate it from the delegation policies. All the delegation operations in our model are controlled by these policies and constraint.

Fig. 2 presents an outline of the proposed delegation model. The delegation agent processes a delegation request by retrieving the delegation policies and constraint from module $DRP$. To test these rules, the agent uses the authorization records (module $AR$), privacy policies (module $PP$) and delegation histories (module $DRH$). Here, the authorization records include the authorization relations described in Section II (C) and Section III (A). These include both regular and delegated user-access right assignments. Delegation histories are the records of the delegation events that have taken place in the system so far.

If a delegation request satisfies all the policies and constraint, a new entry is added to the authorization records assigning the requested right to the delegatee. The new record is later used for taking access control decisions. The delegation event is also logged in the delegation histories. This was an overview of the proposed delegation model. The model will be described in detail over the next few sections.

A. Specification of the delegation model

This section describes the notations and relations used in the delegation model.
Delegation units: A role-based access control model is used in this work. In role-based access control models, access rights are typically delegated using two units—role and privilege. In role delegation, all the privileges of a role are delegated while a single privilege is given away in privilege delegation. Role and privilege delegations are often termed as full and partial delegations, respectively. The privacy policies are same for all the users with the same visibility; so, role delegation within a visibility is not affected by the privacy policies. Study of role delegation in this work would then be no different than the existing literatures [6, 22]. On the other hand, role delegation between two visibilities is a challenging problem. The delegated role should be mapped to one of the existing roles of the destination visibility. Since privacy policies for the source and destination visibilities can be different, it may not be possible to find all the privileges of the delegated role in one of the roles of the destination visibility. To keep the model simple, we study only partial delegation where a user can delegate individual privileges to other users.

Delegation request: A privilege in the proposed access control model gives users access to a data item for a range of purposes. Delegates may want to delegate the access for a subset of the purposes that they have. Thus, a delegation request should include the tuple \( odr = \langle a, p, pr \rangle \) representing an access right that allows performing action \( a \) on data \( d \) for purposes included in the range \( pr \). The tuple \( odr \) referred as delegation object which is only a part of the delegation request. The complete request will contain additional information about delegator, delegatee, and valid time. Valid time is a time period given to each delegated access right. The right is removed from a delegatee when the valid time ends. Let \( Z \) be the set of time points in the system clock. The complete delegation request is given by the following:

\[
dr = \langle u, v, odr, etm \rangle
\]

Here, \( odr \) is the delegation object, \( u \) and \( v \) are the delegator and delegatee, respectively, \( etm \) is the valid time where \( \{u, v\} \in U \) and \( etm \in Z \).

A predicate, \( compliance \), is used to validate delegation requests. It is similar to the predicate \( compliance_{AC} \) described in Section II (C).

\( compliance(dp, odr) \): The predicate tests if the delegation object \( odr \) is contained by the privilege \( dp \). The predicate returns true if the following comparisons are true.

1. \( dp.d = odr.d \)
2. \( dp.a = odr.a \)
3. \( pr_{dp} \leq pr_{odr} \wedge p_{odr} \leq p_{dp} \leq p_{u} \)

Here, notations \( pr_{dp} \) and \( pr_{odr} \) represents the purpose range of \( dp \) and \( odr \), respectively. We say that the privilege contains the delegation object when the predicate returns true. In the predicate \( compliance \), both arguments contain purpose range while in the predicate \( compliance_{AC} \), one argument contains purpose range. This is the only difference between these two predicates.

Relations storing delegation records: In the role-based access control model, users have direct relation with roles through the user-role assignment while having an indirect relation with privileges which is derived through the user-role and role-privilege assignments. In the proposed delegation model, a user receives privileges without the mediation of roles. Thus, delegation sets up a direct relation between users and privileges. The relation is described below followed by other relations that are also used in the delegation model.

- \( TUADP_i \subseteq U_i \times DP_i \times Z \): A relation recording user-privilege assignments resulted from delegation events in a visibility \( i \) including the valid time period for the assignment.
- Each delegation operation is logged by the system. In addition to the information about the delegator, delegatee and access right, a log includes the time when a delegation takes place and its valid time period. Logging uses a relation \( DH \) that is defined over the range:

\[
DH \subseteq U \times U \times DP \times Z \times Z
\]

An entry \( \langle u_1, u_2, o, etm, stm > \in DH \) denotes that user \( u_1 \) has delegated an access right \( o \) to user \( u_2 \) at time \( stm \) which will expire at time \( etm \).

B. Delegation policies and constraint

We apply several policies to control delegation operations among the users. Of them, delegation policies are used to specify what delegation operations are valid. A security policy is applied to maintain the separation of duty among the users. These policies are expressed in a declarative logic language which is described as follows:

Policy language: A slight variant of First Order Logic is used as the policy language. The language consists of a set of variables and constants. Let \( v \) represents a variable and \( cn \) represents a constant. A term \( tm \) is either a variable or a constant. If \( pc \) is a \( n \)-ary predicate and \( tm_1, tm_2, ..., tm_n \) are \( n \) terms, then \( pc(t_1, t_2, ..., t_m) \) is an atomic formula. All the well-formed formulas have the following Backus Naur Form (BNF):

\[
\phi ::= T \mid pc(t_1, t_2, ..., t_m)(\neg \phi) \mid \phi_1 \land \phi_2 \mid (\exists v. \phi) \mid (\forall v. \phi)
\]

Instead of \( \phi_2 \rightarrow \phi_1 \), we write \( \phi_1 \leftarrow \phi_2 \) for convenience. All the formulas or rules that are used to enforce delegation policies and constraint are written in the form of a definite program clause [12].

\[
\phi_1 \leftarrow \phi_2 \text{ where } \phi_1 \text{ is an atomic formula}
\]

Left and right side of the formula is called head and body, respectively. The semantics of the rule is that if the body is true, the head is true. When the body of the rule is the Boolean constant \( T \), it is called called unit clause [12].

\[
\phi \leftarrow T
\]

The semantics of the unit clause is that the head is true.

The predicates used in the delegation policies and constraint can be categorized into four classes: utility, specification, decision and constraint predicates. Though some of these classes have the same names as the ones in [4], the semantics are not the same. The semantics of the
predicate classes are defined as follows. Utility predicates provide functionalities like set operations, counting and comparisons. Specification predicates are used to retrieve information from the system configuration. For example, the set of privileges assigned to a user is returned by one of the predicates of this class. Decision predicates define the enforcement of the rules. Being used as the head of a rule, these predicates denote the consequence when they become true. Constraint predicate is used to enter the security policies into the system. All the predicates used in this paper are listed in Table 2 where \( B \) is set of Boolean values, \( N \) is the set of natural numbers and \( \varnothing \) denotes power set.

### B.1 Delegation policies

A delegation policy is a rule or a set of rules that validates a delegation request. In other words, it is used to decide if a delegation operation is permitted. Denning [25] proposed a policy stating that a user cannot give an access right they do not have to other users; it is called the principle of attenuation of privilege (PAP). Note that this principle imposes restriction only on delegators. The delegation policies used in this work are motivated by this principle.

Policy for intra-visibility delegation:

In intra-visibility delegation, access rights to data items are delegated within the same visibility. The policy for this type of delegation is defined as follows:

Users are eligible to participate in a delegation operation as a delegator if they have a privilege containing the requested delegation object.

In short, the policy says that a user can delegate a privilege if it is assigned to them. The rule validating a delegation request against this policy is given in the following.

\[
\text{canGiveSV}(u, v, dp, odr) \leftarrow \text{hasRights}(u, \text{PrivS}) \land \exists dp \in \text{PrivS}. \text{compliance}(dp, odr)
\]

The rule validates a delegation operation that involves \( u \) as the delegator, \( v \) as the delegatee and \( odr \) as the delegation object. The rule works as follows. The predicate \( \text{hasRights} \) returns all the privileges that are assigned to the delegator \( u \) as \( \text{PrivS} \). The list of the privileges is then probed using the predicate \( \text{compliance} \) for the existence of a privilege \( dp \) that contains the delegation object \( odr \). The predicate stops when such privilege is found and then returns true. Otherwise, false is returned. If the body of the rule is true, the head becomes true indicating that the delegator \( u \) can delegate a privilege containing \( odr \) to the delegatee \( v \).

This policy complies with the PAP principle as it requires that the delegators must have the privileges they want to delegate. In intra-visibility delegation, the delegator and delegatee are from the same visibility. So, it is not necessary to check the privacy policies for taking delegation decision because any privilege valid for the delegator must also be valid for the delegatee.

### Table 2: Predicates

<table>
<thead>
<tr>
<th>Utility predicate</th>
<th>Functionality</th>
</tr>
</thead>
<tbody>
<tr>
<td>compliance: ( DP \times ODR \rightarrow B )</td>
<td>( \text{compliance}(dp, odr) ) returns true if the expression holds: ( dp.d = odr.d \land dp.a = odr.a \land pr_{dp}.p_b \leq pr_{odr}.p_b \land pr_{odr}.p_u \leq pr_{dp}.p_u )</td>
</tr>
<tr>
<td>equality: ( DP \times ODR \rightarrow B )</td>
<td>( \text{equality}(dp, odr) ) returns true if the expression holds: ( dp.d = odr.d \land dp.a = odr.a \land pr_{dp}.p_b = pr_{odr}.p_b \land pr_{odr}.p_u = pr_{dp}.p_u )</td>
</tr>
<tr>
<td>combine: ( \varnothing(DP) \times ODR \times \varnothing(DP) \rightarrow B )</td>
<td>( \text{combine}(\text{SetA}, odr, \text{SetA}') ) instantiates ( \text{SetA}' ) by doing the following operation: ( \text{SetA}' = \text{SetA} \cup {odr} )</td>
</tr>
<tr>
<td>common: ( \varnothing(DP) \times \varnothing(DP) \times N \rightarrow B )</td>
<td>( \text{common}(ST1, ST2, k) ) instantiates ( k ) with the number of privileges of the set ( ST1 ) that matches with one of the privileges in the set ( ST2 )</td>
</tr>
</tbody>
</table>

**Decision predicate**

<table>
<thead>
<tr>
<th>Specification predicate</th>
<th>Functionality</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{vis}: U \times VIS \rightarrow B )</td>
<td>( \text{vis}(v, f) ) instantiates ( f ) with the visibility of the user ( v ).</td>
</tr>
<tr>
<td>( \text{hasRight}: U \times \varnothing(DP) \rightarrow B )</td>
<td>( \text{hasRight}(u, \text{PrivS}) ) instantiates ( \text{PrivS} ) with the privileges assigned to the user ( u ).</td>
</tr>
<tr>
<td>( \text{dlookup}: VIS \times ODR \rightarrow B )</td>
<td>( \text{dlookup}(j, odr) ) returns true if the following expression holds: ( \exists pp \in \text{Policy}<em>j \land odr.d = pp.d \land odr.a = pp.a \land pp.p \leq pr</em>{odr}.p_b )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Decision predicate</th>
<th>Functionality</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{canGiveSV}: \text{canGiveDVC}: \text{canGiveDVE}: U \times U \times DP \times ODR \rightarrow B )</td>
<td>If any of these predicates with the arguments ( (u, v, dp, odr) ) becomes true, it denotes that a user ( u ) can delegate a privilege containing ( odr =&lt; d, a, pr &gt; ) to a user ( v ). The eligibility of the user ( u ) comes from the authorization of a privilege ( dp ). Decision predicates are used as the head of a rule. The value of ( dp ) is obtained by the inference of the rule.</td>
</tr>
<tr>
<td>( \text{error}: U \times U \times ODR \rightarrow B )</td>
<td>If ( \text{error}(u, v, odr) ) is true, it denotes that delegation between users ( u ) and ( v ) is not permitted.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Constraint predicate</th>
<th>Functionality</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{allUserCannotHave}: \text{VIS} \times \varnothing(DP) \times N \rightarrow B )</td>
<td>If ( \text{allUserCannotHave}(j, MDP, k) ) is true, it denotes that a user cannot have ( k ) or more privileges from the set ( MDP ) in visibility ( j ).</td>
</tr>
</tbody>
</table>
Policy for inter-visibility delegation:
In an inter-visibility delegation, access rights to data items are delegated from one visibility to another. The scope of our work is limited to the case where access rights can be delegated to a non-enterprise visibility only from the enterprise visibility. Recall the data collection by the website Toys.com in Section II (A) where the enterprise visibility. In an inter-visibility delegation, access rights to data items are also visible to the source visibility with the destination visibility if the data items visible to the destination visibility are also visible to the source visibility with the same privacy policies. It is important to note that the collaboration policy is the PAP principle compliant. Delegation policy for collaboration is the following:

Users are eligible to participate in a delegation operation as a delegator if they have a privilege containing the requested delegation object as long as the object is valid according to the privacy policy of the destination visibility. The first part of the delegation policy is the PAP principle stating that the delegator must have a privilege containing the requested object. The second part of the policy states that there must be a privacy policy supporting the use of the delegation object by the destination visibility. The rule validating a delegation request with this policy is given in the following.

\[
\text{canGiveDVC}(u, v, dp, odr) \leftarrow \text{hasRights}(u, \text{PrivS}) \wedge \exists dp \in \text{PrivS}. \text{compliance}(dp, odr) \wedge \text{vis}(v, j) \wedge dlookup(j, odr)
\]

Using the predicates hasRights and compliance, the rule checks if the delegator has a privilege that contains the delegation object odr. The predicate vis returns the visibility of the delegatee as j. Next, the predicate dlookup returns true if there is a privacy policy for visibility j that allows the delegation request.

It is important to note that the collaboration policy is always enforceable between a source and a destination visibility if the data items visible to the destination visibility are also visible to the source visibility with the same privacy policies i.e., if the following expression holds.

\[
\forall l \in \text{PPolicy}_a \exists l' \in \text{PPolicy}_d . (l. d = l'. d \wedge l. a = l'. a \wedge l. p = l'. p)
\]

Here, \text{PPolicy}_a and \text{PPolicy}_d are the privacy policies for the source and destination visibilities.

Delegation policy for exchange:
In exchange, the source and destination visibilities have separate business process. Data items are not necessarily visible to both of them with the same privacy policies. A delegation policy (like the policy for collaboration) asking that a delegator must have access to the requested data item with the requested action and purposes, may not be useful in exchange. We can also use the privacy policy as the only delegation policy which means that access to a data item with an action and a set of purposes will be given to a delegatee if there is a privacy policy for the destination visibility supporting such data usage. In such case, no users from the source visibility will be involved in delegation.

To ensure the involvement of the users from the source visibility, we define a delegation policy that is more relaxed than the policy for collaboration. Instead of requiring that a delegator have access to the requested data with the requested action and purposes, the new policy requires that the delegator have access to the requested data only; this may be with different action and purposes. Like the collaboration policy, it also checks if the delegation request is supported by a privacy policy of the destination visibility. The policy is given by the following text:

Users are eligible to participate in a delegation operation as a delegator if they have access to the requested data item as long as the delegation object is valid according to one of the privacy policies of the destination visibility. The rule validating a delegation request with the exchange policy is given in the following.

\[
\text{canGiveDVE}(u, v, dp, odr) \leftarrow \text{hasRights}(u, \text{PrivS}) \wedge \exists dp \in \text{PrivS}. (dp. d = odr. d) \wedge \text{vis}(v, j) \wedge dlookup(j, odr)
\]

The rule first checks if the delegator has a privilege \(dp\) that contains the requested data \((odr. d)\). Next, using the predicates \text{vis} and \text{dlookup}, the rule validates the delegation request against the privacy policies of the destination visibility.

The exchange policy is always enforceable between a source and a destination visibility if the data items visible to the destination visibility are also visible to the source visibility i.e., if the following expression holds.

\[
\forall l \in \text{PPolicy}_d \exists l' \in \text{PPolicy}_s . (l. d = l'. d)
\]

B.2 Security constraint
Delegation allows users in an access control system sharing their access rights without the mediation of a system administrator. Schaad [17] showed how delegation can be used to violate the system’s security. To protect a system’s security, its administrator should identify what property they want to preserve throughout the delegation operation and then they should apply policy to enforce that property. In this work, we design a security constraint to enforce the separation of duty (SoD) among the users of the system. The objective is to distribute the access rights among multiple users to prevent fraud. Since delegation is a process of assigning access rights to users, only the static version of the SoD policy [13] is applied in this work and
other versions of the policy which are enforced at runtime are not considered here.

Static Separation of Duty (SSoD) policy: When a sensitive task is comprised of n steps, SSoD policy requires the cooperation of at least k (for some k ≤ n) different users to complete the task. [13]

Consider a sample SSoD policy that says at least three users together can have the privileges \{dp₁, dp₂, dp₃, dp₄\}. The policy can be written in the format \(<dp₁, dp₂, dp₃, dp₄, 3\>). Li et al. [13] show that an SSoD policy can be converted to one or more sub policies where each sub policy involves a single user. Let this new policy be denoted by SSoD’. How an SSoD policy can be expressed by one or more SSoD’ is demonstrated with an example here. Consider the previously mentioned SSoD policy \(<dp₁, dp₂, dp₃, dp₄, 3\>) that says at least 3 users can have the privileges altogether. It can be converted to an SSoD’ policy \(<dp₁, dp₂, dp₃, 2\>) stating that no user can have two or more than two privileges from the specified set. Enforcing this constraint will lead to a state where three or more users will have these privileges \{dp₁, dp₂, dp₃, dp₄\} which also satisfies the SSoD policy.

In a delegation event, only the delegatee receives privileges. So we need to check the privileges of the delegatee to see if the delegation violates the separation of duty. As the SSoD’ policies also put restriction on a single user, they are suitable to apply in delegation. So, we use the idea of using SSoD’ policies to enforce the separation of duty. Li et al. give a detail description of how an SSoD policy can be converted to one or more SSoD’ policies. We skip this description here. The rest of this section describes how the SSoD’ policies can be expressed using the policy language.

General format for an SSoD’ policy is \(<MDP', \{dp₁, dp₂, ..., dp₇\}, k\>) that says a user cannot have k or more privileges from the set MDP. The following type of rule is used to enter the SSoD’ policies for a visibility j into the system.

\[ \text{aUserCannotHave}(j, MDP, k) \leftarrow T \]

Since the rule is a unit clause, the system will accept the information entered using the predicate aUserCannotHave. The rule that enforces the SSoD’ policies is defined below.

\[ \text{error}(u, v, odr) \leftarrow \text{vis}(u, j) \]
\[ \land \text{aUserCannotHave}(j, MDP, k) \land \text{hasRight}(v, \text{PrivS}) \]
\[ \land \text{combine}(\text{PrivS}, \text{odr}, \text{PrivS'}) \]
\[ \land \text{common}(\text{PrivS'}, MDP, k') \land k' \geq k \]

Using the predicates vis and aUserCannotHave, all the SSoD’ policies of the delegatee’s visibility are returned. For each of the returned policies, the following things are done. The privileges assigned to the delegatee are returned as the set PrivS using the predicate hasRight. The predicate combine, combines the delegation object with the set PrivS and returns the output as the set PrivS’. The predicate common returns the number of privileges of the set PrivS’ that have a match with one of the privileges of the set MDP. Two privileges match when they have the same data and action and the intersection of their purpose ranges is not null. If the number of matched privileges is k or more, the predicate error becomes true and the delegation will be denied.

C. Delegation process

The delegation agent (in Fig. 2) processes delegation request by using the delegation decision algorithm. Inputs to the algorithm are u and v as the delegator and the delegatee, a set A of delegation objects, the valid time etm for the requested rights, and a empty set A’ which is used to store the processed requests. The algorithm validates each request by testing either intra- or inter-visibility delegation policies (line 4-9). If the request is valid by one of these policies, it is then validated against the security constraint (line 10-11).

**Delegation decision algorithm**

<table>
<thead>
<tr>
<th>Input: ( (u, v, A, etm, A') ). Output: Boolean value indicating successful formation of the set A’</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 ( i := \text{vis}(u) ), ( j := \text{vis}(v) )</td>
</tr>
<tr>
<td>2 ( A' := \emptyset ), ( dp := \emptyset )</td>
</tr>
<tr>
<td>3 For each ( odr \in A ):</td>
</tr>
</tbody>
</table>
| 4 \( \text{If} \ (i = j) \text{ then} \)
| 5 \( \text{If} \ \lnot \text{(canGiveSV}(u, v, dp, odr) \text{ then} \)
| 6 \( \text{return false} \)
| 7 \( \text{Else if} \ (i \neq j \text{ and } i = "Enterprise") \text{ then} \)
| 8 \( \text{If} \ \lnot \text{(canGiveDVC}(u, v, dp, odr) \text{ then} \)
| 9 \( \text{return false} \)
| 10 \( \text{If} \ \text{error}(u, v, odr) \text{ then} \)
| 11 \( \text{return false} \)
| 12 \( \text{If} \ \text{equality}(dp, odr) \text{ then} \)
| 13 \( A' := A' \cup dp \)
| 14 \( \text{Else} \)
| 15 \( dp' := odr \)
| 16 \( DP_i := DP_i \cup dp' \)
| 17 \( A' := A' \cup dp' \)
| 18 \( \text{End For each} \)
| 19 \( \text{return true} \) |

If the delegation request fails to pass any of the previous steps, it is denied by returning false. Otherwise, the process goes to the next step where the algorithm builds a set \( A' \) of access rights from the validated delegation requests. The set is built by comparing each validated request \( odr \) with the privilege \( dp \) (line 12-17). Note that privilege \( dp \) is returned by the inference of the delegation policy. If the privilege is not equal to the request, a new privilege is created based on the request for the delegatee’s visibility and added to the set \( A' \). If the privilege is equal to the request, it can be directly delegated and so, it is added to the set \( A' \).

The delegation decision algorithm does not describe the assignment of the privilege set \( A' \) to the delegatee. The
assignment should be done in the following ways. Each privilege of the set \( A' \) is assigned to the delegatee by adding an entry to the temporary privilege assignment relation \( TUADP \) with the valid time of delegation. The assignment is also followed by the addition of an entry to the delegation history \( DH \) that consists of delegator, delegatee, delegated privilege, valid time and start time for delegation.

### Complexity of the decision algorithm:

Due to the space limitation, we present only the results of our complexity analysis. In the current implementation, the separation of duty policy enforcement is the most expensive of all the policies in both intra- and inter-visibility delegation. In the worst-case scenario, the cost of the decision algorithm is equivalent to the cost of the security policy. So, we only describe the complexity of enforcing the security policy.

The rule enforcing the security constraint verifies if a delegation operation violates any of the SSoD’ policies of the system. In the rule, the predicate \( allUserCannotHave \) returns one SSoD’ policy at a time and all the remaining elements are tested for that policy. In the presence of a role hierarchy, the cost of finding the privileges assigned to a user by the predicate \( hasRight \) is \( nR^2 \). The cost of predicate \( combine \) is negligible and ignored here since it does a union operation. The cost of finding the common privileges between two privilege sets by the predicate \( common \) is \( (nDP^2 \cdot nP) \). Finally, the total cost of validating one SSoD’ policy consists of the costs of the predicates, \( hasRight, \) and \( common, \) which is \( O(nR^2 + nDP^2 \cdot nP) \). It is equivalent to \( O(nDP^2 \cdot nP) \). If the number of SSoD’ policies entered into the system is \( nSoD \), the complexity becomes \( O(nSoD \cdot nDP^2 \cdot nP) \).

### IV. REVOCATION

Revocation is the process of removing delegated access rights from a delegatee. In the proposed delegation model, one possible revocation policy would be allowing a delegator to revoke the privileges they delegated in the past. We can call this type of revocation as **forced revocation** as the delegators can revoke the privilege at any time they want. Since we consider temporal delegation where a privilege is delegated for a specific period of time, another revocation policy would be allowing the system to revoke the delegated privileges from the users when the valid time ends. This type of revocation is termed as **auto revocation**. These revocation policies can be formally expressed by the policy language. Due to space limitation, we do not present the details of these policies here.

Successful inference of a revocation policy will initiate the cleanup task that would remove the delegated privilege from the delegatee. The revocation event should also be logged using a relation similar to the delegation history that would contain who initiated the revocation - either a user or the system itself, what privilege was revoked, who the delegatee was, and when the revocation took place.

### V. RELEVANT WORKS

Many delegation models have been proposed in the literature covering different aspects of delegation including role and privilege delegation [6, 20]. The assignment of delegated authorizations to users and their enforcement through access control decisions have also been investigated [6, 11, 22]. There are proposals [20-21] that study multi-step delegation where a delegated access right is further delegated. Some delegation models (e.g., [10]) are unconstrained while others [2, 21] apply delegation policies that allow or deny a delegation operation. Mechanisms to revoke delegated access rights have been studied by several research works [2, 21].

However, there are a few works in the literature that investigate data privacy in delegation. One of these models [9] is based on the identity management systems where data providers get services from different service providers by giving access to their data. To provide the service, a service provider may rely on other service providers transitively and so, the data is shared with them too. The model does not reveal the data provider’s identity (ID) to any of the service providers. Instead, it uses a trusted third party to maintain a pseudo ID of the data provider for each service provider. When the data provider wants to provide access to their data to a service provider, the third party issues a credential [7] containing the data provider’s pseudo ID for that service provider. Credentials are authorization certificates used in the identity management systems. If a service provider wants to delegate the credential to another service provider, a new credential is created containing the data provider’s pseudo ID for the new service provider. The data providers remain anonymous in the entire process, so the authors claim that the privacy is protected. However, hiding only the identity information may not be sufficient because the exposure of other information like address and date of birth can lead to the identification of a data provider [18]. In addition, the data provider’s identity is required to provide some services, e.g., delivering products. So, some details may need to be exposed to receive the service. Instead of removing the identity information from the data, our model requires that the data provider be aware of the privacy policies that state who will access their data and how their data will be used. These policies later control the requests to use and share the data.

Bussard et al. [5] propose an XML-based policy language to encode the privacy preferences of a data provider. The language is designed with multiple data transfers in mind i.e., a provider can specify if one visibility can allow other visibilities to access the data and if so, how these new recipients should use the data. The authors also propose to use the privacy policies as the access control policies for data. In our model, we separate the access control policies from the privacy policies. This separation gives an organization better control over the data usage because it can create more restrictive privileges than the privacy policies. We also study hierarchical purposes and roles in our work while Bussard et al. consider flat purposes and roles.
With the rise of cloud based applications and social networks [27, 29], people now disclose more personal information than ever. Inappropriate use of the information can harm their lives. There are many examples of such occurrence in recent years. In one news, a Canadian woman who was on sick leave for depression was denied of her benefits when her insurance agent saw her pictures in Facebook where she was having fun with friends [8].

An effective way for privacy protection is to set the privacy policies that are the agreements between the data provider and collector about who can use the personal information and how the information should be used. Accesses to the information are then controlled by these policies. In this work, we propose a privacy model to formalize privacy policies for multiple parties who access the data.

In a privacy preserving access control model, access rights contain privacy restrictions. Data users must satisfy the restrictions in order to get access. Another contribution of our work is to propose a delegation model that facilitates access rights sharing in this type of access control model. The proposed delegation model takes privacy policies into consideration for taking delegation decisions. The model also ensures that when an access right is delegated, it is constrained by the appropriate privacy policy for the receivers. We study delegation within a visibility and between two visibilities.

One of our immediate future works is to investigate multistep delegation where the receiver of an access right can further delegate it. Besides, delegation is highly practiced in health care sectors where accesses to patients’ data are exchanged among the professionals of different health organizations. In future, we plan to extend our delegation model for this application.

REFERENCES


