Delegation of Access Rights in A Privacy Preserving Access Control Model

by

Md. Moniruzzaman

A THESIS
SUBMITTED TO THE FACULTY OF GRADUATE STUDIES
IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE
DEGREE OF MASTER OF SCIENCE IN COMPUTER SCIENCE

DEPARTMENT OF COMPUTER SCIENCE

CALGARY, ALBERTA
APRIL, 2011

© Md. Moniruzzaman 2011
UNIVERSITY OF CALGARY
FACULTY OF GRADUATE STUDIES

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies for acceptance, a thesis entitled “Delegation of Access Rights in A Privacy Preserving Access Control Model” submitted by Md. Moniruzzaman in partial fulfilment of the requirements for the degree of Master of Science in Computer Science.

Supervisor, (Dr. Ken Barker and Computer Science)

(Dr. Philip W.L. Fong and Computer Science)

(Dr. Reda Alhajj and Computer Science)

(Dr. Mahmood Moussavi and Electrical & Computer Engineering)

APRIL 27, 2011
Date
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. This thesis investigates data privacy in delegation. As a contribution, a privacy model is introduced that allows a data provider setting privacy policies to state how their data should be used by different organizations or parties that are interested in their data. Based on this setting, a delegation model is designed to consider the privacy policies of data in taking delegation decisions and also, set the data usage criteria for the access right receivers. In addition to privacy policies, several delegation policies and constraints have been used to control delegation operations. Delegation is studied within a party and between two parties.
I want to give thanks to my supervisor Dr. Ken Barker who encouraged me to pursue research on my own ideas. It was my first research degree and I was able to enjoy it because of the support from Ken.

I also want to thank my colleagues and group mates, especially Leanne, Maryam, Mistu, Kambiz, Rosa, Dina and Sharmila. These people were never tired of discussing research with me. They often gave me valuable feedbacks on my work. I am also indebted to the weekly meetings of the PSec research group. It is an open and friendly meeting that encourages anyone to ask research related questions. This weekly session helped me a lot to get engaged in discussion with other group mates.

I am grateful to my wife, Shimu. She gave me enormous support in the entire period of my study. She even listened to all of my practice presentations for the defense. She gave me a lot of courage to pass through the final phases of the degree. My sincere gratitude goes to my mother. Her love and pray made me come this far. I love you, Mum. My father had a dream that I would have higher education. Today, I am one step closer to his dream. Don’t forget to smile from the heaven on the convocation day when I hold the diploma.

I dedicate this thesis to my nephew Pritul and my brother-in-law Shaheen Bhai who are staying in bed for about three months now after the terrible accident. I cannot feel the hardship you are going through, I can only image. I wish I could have half as much patience as you have.
# Table of Contents

**Abstract**

**Table of Contents**

**List of Tables**

**List of Figures**

1. **Introduction**
   1.1 Contribution
   1.2 Thesis organization

2. **Background information**
   2.1 Basic terminologies
   2.2 Concepts of data privacy
   2.3 Privacy negotiation
   2.4 Privacy enforcement using access control models
      2.4.1 Access control models using privacy-aware database
      2.4.2 Access control models based on RBAC
      2.4.3 Access control models based on transactions/workflows
      2.4.4 Access control models using XML based policy language
   2.5 Background literatures on delegation
      2.5.1 Units of delegation
      2.5.2 Classification of delegation based on the type of delegator and delegatee
      2.5.3 Delegation across security domains
      2.5.4 Delegation policy
      2.5.5 Multistep delegation and revocation
2.5.6 Security in Delegation
2.5.7 Privacy preserving delegation
2.5.8 Grant/Transfer delegation

3 Access control model: Platform for delegation
3.1 Privacy model
3.2 Privilege model
3.3 Formal specification of the access control model
3.4 Access control

4 Delegation
4.1 Introduction
4.2 Specification of the delegation model
4.3 Delegation policies and constraints
   4.3.1 Delegation policies
   4.3.2 Security policies
   4.3.3 Duplicate delegation policy
4.4 Delegation process
4.5 Revocation

5 Case study
   Case#1: Intra-visibility delegation
   Case#2: Inter-visibility delegation

6 Complexity analysis
   6.1 Complexity of the delegation policies and constraints
   6.2 Total complexity of the delegation decision algorithm
7 Conclusion
  7.1 Contribution 89
  7.2 Future works 89

Appendix A 93

Bibliography 102
List of Tables

2.1: Different views of data for privacy 12

3.1: Customer information 36
3.2: Privacy policies, $PPolicy_{ws}$ 36
3.3: Privacy metadata, $StatDS_{ws}$ 36

4.1: Utility predicates 54
4.2: Specification predicates 55
4.3: Decision predicates 55
4.4: Constraint predicates 56

5.1: Privacy policy for the visibility service provider, $PPolicy_{sps}$ 74
5.2: Privacy policy for EShop's subsidiaries and parent organizations, $PPolicy_{ps}$ 77

6.1: Notations used in the complexity study 80

A.1: Partial orders among purposes of Fig. A.1 94
A.2: Index for the purpose hierarchy of Table A.1 94
List of Figures

2.1: Role hierarchy 25
3.1: Purpose hierarchy 38
4.1: Delegation model (a Strawman architecture) 45
4.2: Delegation model 47
A.1: Simple purpose hierarchy 93
Chapter 1
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. It may not be feasible due to the dynamic nature of the workplace where users may need new access rights. 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 an access right from another user who already possesses it. The latter approach is called delegation.

Delegation brings flexibility to access control models. Zhang et al. [ZOS03] 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.

Delegation has been studied in a variety of access control systems like role based access control models (RBAC) [SCF+96], workflow systems [BFA99] and identity management systems [GHH+05]. Several delegation models have been proposed covering different aspects of delegation. Several works [CK08-IJIS, WKB07] have been proposed that study partial and full delegation of access rights. How delegated authorizations are assigned to
users and how these are enforced while taking access control decision have been investigated [CK08-IJIS, JB06, ZOS03]. There are proposals [WKB07, ZAC03] that study multi-step delegation where a delegated access right is further delegated. Some models (e.g., [HMM10]) are unconstrained while others [AW05, ZAC03] apply rules that allow or deny a delegation operation. Some models study delegation within one security domain [CK08-IJIS, ZOS03] while others [CO03, HMM10] study delegation across security domains. Mechanisms to revoke delegated access rights are also studied by a handful of research groups [AW05, ZAC03].

Despite its usefulness, delegation may produce security risks for an organization. Consider the case where a system administrator does not assign sensitive privileges to a user $u$ for security reasons. In a delegation enabled system, it may not be sufficient for security protection as $u$ may receive the privileges from other users. Constraints can be applied to limit delegation operations that may pose security risks.

Delegation may also lead to data privacy violations. Delegation of access rights 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 should 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. Such models are known as privacy preserving access control models. Several models have been proposed [MFH10] in the literature.

The concepts of data privacy vary in these models. However, most of them agree that purpose is a necessary component for protecting data privacy. Purpose is the intention for using data. When data is collected, data providers specify the intended purposes for using the data. A privacy preserving access control model requires submitting a purpose as a prerequisite for data access. The request is granted if the access purpose and the provider’s intended purpose match.

Most of the privacy preserving access control models assume that data is accessed only by the collecting organization. In real life, many parties are interested in data apart than the collecting organization. One of the contributions of this work is defining a privacy model that allows data providers setting privacy policies for different organizations accessing their data.

Another common limitation of many privacy preserving access control models is that they are not fine grained. Data users are assigned to purposes and they get access to all data items that have the same intended purposes. Access control is mostly based on privacy policies. However, an organization may want to limit its users’ accesses to a subset of data items allowed by the privacy policy. To address this shortcoming, a role-based model called P-RBAC [NTB+07] is adapted in this work. P-RBAC extends the structure of a privilege in RBAC to incorporate data privacy policies. Similar to RBAC, privileges are
grouped into roles and roles are assigned to users in P-RBAC. Being an extension of RBAC, P-RBAC can express fine grained access control policies.

Based on these foundations, a delegation model is proposed where access rights to a data item can be delegated if it is allowed by the data item’s privacy policy. Delegated access rights contain privacy constraints. As a result, the receivers of the rights are bound by the privacy constraints when they use the data. The proposed model also investigates prohibiting certain delegation operations to maintain the access control model’s security and efficiency. Finally, this work studies revocation of delegated access rights.

1.1 Contribution

The contributions of this thesis are summarized as follows. (i) A privacy model is proposed that formalizes the data privacy policies. (ii) An existing access control model is adapted that attaches privacy restrictions into privileges. In addition, the relation between the privacy model and the access control model is discussed showing how privileges are created from privacy policies. (iii) The access control model is extended so that it groups users who access data into different visibility levels and define a different set of access rights for each level. (iv) We study delegation in a privacy preserving access control model where access rights to data items contain privacy constraints drawn from the privacy policies. Such rights tell the data users about how to use the data items. The proposed delegation model studies how these rights can be shared from one user to another user. When the users are from two different visibilities, our delegation model ensures that access rights are not delegated to a visibility which is not in the privacy agreement of the data. To our knowledge, no work exists in the literature that studies delegation in privacy preserving
access control model. To prevent security violation and maintain the system efficiencies, we also apply constraints in delegation. All the policies and constraints are written using a declarative logic language. The declarative nature of the language allows extension to incorporate new policies and constraints in future. (v) Several approaches for revoking delegated access rights are explored.

1.2 Thesis organization

This thesis starts by giving an overview of relevant literatures on data privacy, privacy preserving access control and delegation in Chapter 2. As the foundation pieces of the proposed delegation model, the privacy model and the access control model are described in Chapter 3. The proposed delegation model is presented in Chapter 4. Applying the proposed concepts, two case studies are given in Chapter 5. Complexity analyses of different algorithms used in this thesis are described in Chapter 6. Conclusion and future works are discussed in Chapter 7. Appendix A includes pseudo codes for different predicates used in the delegation model.
In access control models, delegation is the sharing of access rights among users. Delegation enables a user to work on another user’s behalf. Users may delegate a variety of access rights that can be permissions to access devices, data, etc. The scope of this thesis is limited to an access control model that restricts access to data.

An organization collects data about its customers. Data may include an individual’s private and sensitive information. Delegation of access rights allows new users to access data. This leads to a question that asks if data providers have given consents for the new users to use their data. Even if the providers’ consents are taken, the issue of how their personal information will be used is critical. Any of these issues may violate data privacy if they are not resolved. The main motivation of this work is to investigate privacy protection in delegation. A delegation model is proposed that prevents delegating data access to new users if it violates the provider’s privacy.

Once access rights are delegated, how the receiving users use data is another important privacy concern. There are several access control models that enforce data privacy [MFH10]. If a delegation model is built on top of this type of access control model, privacy will be enforced when the users use data. Therefore, a privacy preserving access control model [NTB+07] is adapted as a basis for the proposed delegation model. As the
background literatures for privacy preserving delegation, we will describe data privacy, privacy negotiation between data providers and collectors, privacy enforcing access control model and privacy preserving delegation over the next few sections.

2.1 Basic terminologies

Individuals who provide their personal information are called *data providers*. Examples include customers, patients, website users, etc. Organizations that collect and store information about individuals are called *data collectors*. *Data users* are those who use the collected data. For instance, employees and business partners of a data collector are examples of those who may use the data.

In an access control model, a user who gives access rights to another user is called a *delegator* and a user who receives access rights is called a *delegatee*. The terms permission and privilege are used interchangeably to indicate access right in access control models.

2.2 Concepts of data privacy

Agrawal *et al.* [AKS+02] list ten principles that are necessary to protect on individual’s privacy. A data collector or enterprise should maintain these principles before and after they collect data about their customers. These principles are: 1. specifying the intended use of data, 2. taking consent from owner, 3. collecting minimal required data, 4. using data only to the specified cases, 5. limiting disclosure of data, 6. specifying how long data is kept, 7. maintaining data accuracy, 8. ensuring security for data, 9. allowing owner to access data after collection and 10. allowing owner to check compliance to these principles.
The authors also propose a theoretic model called a Hippocratic database applying these principles.

Barker et al. [BAB+09] proposed a privacy taxonomy which expresses data privacy using four components – purpose, visibility, granularity and retention. Some of these components directly maps to the previously mentioned privacy principles. For example, purpose and retention maps to principle 1 and 6, respectively. Purpose is the reason or intention for using data. Privacy of a data item is breached if it is used for a purpose that is not explicitly specified in the privacy policy. Visibility refers to the categories of data users who can access data. Barker et al. identify four visibilities: Data provider, Data collector, Third party and World. Data with visibility Data provider can be accessed by the provider only. Data with visibility Collector can only be accessed by data users inside the collecting organization while visibility Third party let the collector share data with other users outside the organization. Finally, the visibility World allows everybody to access data. Granularity is the level of detail for a data item. Consider home address as a data item that can have the following granularities- exact address, only city name, only country name, etc. How long a collector should keep data is defined by its retention. It can be in the form of a date, time period or number of accesses. Data should be removed from the system once its retention period ends. Barker et al. [BAB+09] argue that these four components cover individuals’ privacy requirements for their personal information.

Obligation is another component that exists in privacy documents (e.g., [COP98]) and is used by many privacy preserving models [MJ08, NBL08]. It is a task that should be
performed by data users as a responsibility for accessing data. Notifying data providers after using their data is an example of obligation.

Privacy definitions used by the research community vary. However, in most cases these definitions are drawn from the privacy components described in this section. The definition of data privacy used in this project includes purpose and obligation.

2.3 Privacy negotiation

As described in the previous section, data privacy consists of a set of components that define how data should be used. For each data collection, both data providers and collectors can set the values of these components. Privacy negotiation is the process of comparing these two policies. The result of the negotiation is the privacy agreement and later, the data is used according to the agreement. The World Wide Web Consortium proposed a protocol called Platform for Privacy Preferences Project (P3P) [W3C]. P3P includes an XML based policy language that can be used by a website to encode their practices about data. Website users who are data providers can also set up their privacy preferences in the internet browsers. When a website is visited, the browser checks if the practices and preferences match. If not, the user is warned that the website is not compliant with the desired privacy settings. P3P has been well accepted in the online community. However, it does not solve the privacy problem as it cannot guarantee that a website will maintain its promises in practice. The next section discusses several access control models that enforce privacy policies. It is important to note that privacy negotiation takes place before data collectors
start using data. The access control models assume that the privacy negotiation is completed and the privacy policies for data are the result of the negotiation.

2.4 Privacy enforcement using access control models

Moniruzzaman et al. [MFH10] divide the privacy preserving access control models into four classes: models that use a privacy-aware database, models that extend role-based access control, models that are based on transactions and workflow systems, and finally, models that use XML based policy language. In the following sections, each class is described and notable works of that class are presented. It is important to note that most of the privacy preserving models use or adapt role-based access control model (RBAC) [SCF+96]. RBAC groups the users of an organization by assigning roles related to their job responsibilities. Permissions are then assigned to roles that give users access to data items. However, RBAC was not designed with privacy in mind; so it cannot express or enforce privacy policy.

2.4.1 Access control models using privacy-aware database

The models of this type store privacy policies within a database. The policies are also tied to data items. Access to any data item should satisfy the associated privacy policy.

Agrawal et al. [ABG+05] propose an extension for relational database systems applying the concept of Hippocratic database. The proposed model stores the collected data in a relational database. During data collection, it uses P3P for the privacy negotiation. A policy translator module transfers the resultant privacy agreement that is in P3P syntax into a schema, say $TI$. Policies stored in $TI$ define what data can be used for which purpose and
by which recipients. In this model, data users and recipients are two different entities. A recipient can be the data collecting organization itself or any of its partner organizations. Schema $T_I$ also stores the information regarding a data provider's opt out information for a particular purpose and recipient. Based on the privacy policy in $T_I$, a special constraint $restriction$ is created which is then tied to a data item. Access to the data item is allowed when the attached constraint is satisfied. The authors provide a language derived from SQL to write the constraint.

Data users are given privileges using SQL authorization statement (e.g., grant). To be compliant with the extended model that uses $restriction$, each grant statement includes additional parameters like purpose and recipient. When a data query is submitted along with the purpose and recipient, an algorithm checks if there is any $restriction$ for each data reference in the query and it then replaces the reference by a dynamic view. The dynamic view is created on-the-fly based on the $restriction$ related to the data. Later, the output of the view works as the data source for the query. As the view reveals the amount of data allowed by the privacy policy, the authors claim that the data provider's privacy is protected.

The model allows the data provider to specify the purpose for which data can be used. In addition to purpose, the data provider can specify the recipient of the data i.e., who will use the data. For a particular purpose-recipient pair, a data provider can opt out and then that recipient with that purpose cannot use the data. This model does not investigate privacy components like retention, granularity and obligation.
Byun and Bertino [BB06] present a privacy preserving model that creates different views of a data item based on the data provider’s privacy preference. Here, the concept of “view” is identical to the privacy component granularity. The model assumes that during data collection, a data provider mentions the purpose for using data and the level of privacy for that use. The three levels of privacy are considered – Low (L), Medium (M) and High (H). For a given purpose, Low level indicates that the provider is not really concerned about using the data for that purpose, Medium level indicates a moderate concern of the provider and finally, a High level indicates that the provider is very concerned about such usage of the data. For each level of privacy, a different version of a data item is created. A sample data model is presented in Table 2.1 that shows different version of a customer record for different levels of privacy. The table also includes the customer’s preferred privacy levels for the purpose P_contact and P_promotion, which we will use later.

### Table 2.1: Different views of data for privacy

<table>
<thead>
<tr>
<th>Customer ID</th>
<th>Name</th>
<th>Address</th>
<th>Age</th>
<th>P_contact</th>
<th>P_promotion</th>
</tr>
</thead>
<tbody>
<tr>
<td>1010</td>
<td>John Smith</td>
<td>6 Essex Street, Toronto, ON</td>
<td>L</td>
<td>{L,L,H}</td>
<td>{M,L,M}</td>
</tr>
<tr>
<td></td>
<td>John S. J.S.</td>
<td>Toronto, ON</td>
<td>L</td>
<td>40 years</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>ON</td>
<td>M</td>
<td>35-45 years</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>H</td>
<td>&gt;30 years</td>
<td></td>
</tr>
</tbody>
</table>

The proponents suggest adapting any existing access control model for authorization management. Regardless of the authorization layer, a data user should submit an access purpose for querying the database. Query output depends on the preferred privacy level of the data provider for that access purpose. For instance, if a query runs on Table 2.1 with
purpose \textit{P\_promotion}, it will produce the output <1010, John S., <6 Essex Street, Toronto,ON>, 35-45 years>.

Byun \textit{et al.} propose a purpose-based access control model for relational database systems [BL08]. They consider purpose as the only privacy preference given by a data provider. Purpose is stored in the same table with data in the database and acts as the intended purpose when accessing the data. PBAC uses a role based approach for authorization management where a data user is given a role and instead of assigning permissions to a role, purposes are assigned to a role. Later, users use these purposes to request data. A data access request is granted if the access purpose matches the intended purpose of the data. In PBAC, role-purpose assignment is conditional in that users of a role should satisfy the condition to have the purpose. For example, \((r, Admin, UserID='123'AND time=[9am:5pm])\) is a role-purpose assignment that states any user with the role \(r\) can use purpose \textit{Admin} only if the user’s ID is 123 and time of data access is between 9am and 5pm. The model considers both positive and negative notions of purpose. If a data item has a negative intended purpose, then the item can be accessed for any purpose except the intended one. One limitation of the model is that the access control is not fine grained; once a data user is authorized for a purpose, they can access all data having the same intended purpose.

\textbf{2.4.2 Access control models based on RBAC}

Many privacy preserving models use RBAC for authorization management among data users. These models fall into two categories - the models that extend RBAC and the models
that use RBAC like an add-on. This section describes the models that have extended or modified RBAC to express privacy policy.

Ni et al. propose a family of role-based privacy preserving access control models called P-RBAC [NTB+07, NLB+07]. In a typical role-based access control model, an access privilege consists of data and action. In P-RBAC, privilege has the form \((\text{data } d, \text{ action } a, \text{ purpose } p, \text{ condition } c, \text{ obligation } ob)\) which expresses a privacy-aware access control policy stating that in order to perform the action \(a\) on the data \(d\), the intended purpose should be \(p\) while satisfying the condition \(c\) and obligation \(ob\). In this model, privileges are created based on the privacy policy of data. Similar to the RBAC model, permission is assigned to a role which in turn is assigned to a data user. Condition used in permissions is a Boolean expression that verifies the values of variables. Here, variables store contextual information that is necessary to validate a privacy-aware permission. Commonly stored information include parental consent and data provider’s consent.

Ni et al. [NBL08] provide a framework for encoding obligation in privileges for the P-RBAC model. Obligation consists of a general constraint, user, action, object and temporal constraint. Here, general constraint is a Boolean expression, user is the person performing the obligation, temporal constraint stores the activation time of obligation with respect to data access and finally, object may include users and temporal constraints. A sample obligation policy is discussed in the following.

Consider *Legal procedure* as a purpose. An enterprise uses a customer’s information for this purpose when it receives a court order to disclose the information. Assume that the
enterprise promises its customers that when such use will happen, they will be notified. The policy is modeled using the following privilege.

\[
<\text{ pdata, read, Legal procedure, }<\text{ LPNotified=NO, self, notify(), dprovider, tc}>>
\]

The privilege gives access to \( \text{pdata} \) for purpose \( \text{Legal procedure} \). Here, \( \text{pdata} \) can be replaced with any personal information like name, address, social insurance number, etc. The underlined part in the privilege is the obligation which states that a user who access data should notify (\( \text{notify()} \)) the data providers (\( \text{dprovider} \)). The enterprise has a policy for notifying a customer once for each legal procedure. Based on the value of the context variable \( \text{LPNotified} \), the users decides if they need to do the obligation. The term \( \text{self} \) denotes that users accessing privilege and performing obligation should be the same. Temporal constraint \( \text{tc} \) typically consists of start and end time that say when an obligation should be performed. The constraint might also specify when a user should reattempt to do obligation if there is no reply from the target object (\( \text{dprovider} \)).

In P-RBAC, the authorization rules or privileges are created from the privacy policy. However, the absence of a formal model for privacy policy must be addressed. P-RBAC is adapted in this thesis by proposing a formal privacy model. We also introduce a simplified version of P-RBAC privilege.

2.4.3 Access control models based on transactions/workflows

In traditional access control models like RBAC, there is no control on the order of data access. However, in workflows and transactions the set of data accesses to complete any
objective are predetermined. This section describes the approaches that enforce data privacy via transaction or workflow models.

Yang et al. [YBZ07] extends the purpose-based access control model (PBAC) originally proposed by Byun et al. [BL08]. (See Section 2.4.1 for PBAC description.) PBAC was designed for relational database systems. Yang et al. propose a privacy aware data model that is independent of any database system. In the data model, data is categorized into several classes. A sample data class is Contact record which covers all the contact information of a data provider include home address, telephone number and so on. Data providers give their privacy preference for each data class instead of for each data item. Privacy preference contains purpose and retention.

The authorization model is similar to the original PBAC where purpose is conditionally assigned to a role. Yang et al. constrain data access through transactions. They define the set of necessary data accesses to fulfill each purpose and the transactions that would contain those accesses. For example, (Marketing, Contact record, T#21) expresses that one of the necessary data accesses to fulfill the purpose Marketing is to access data of type Contact record through the transaction T#21. In addition to purpose, the model includes retention. However, it does not provide any framework to enforce retention that would delete data when the retention period expires.

As discussed before, one limitation of the original PBAC is that it does not support complex access control policies. Yang et al. address this problem by specifying the necessary accesses for each purpose. In their model, a data user is authorized for a purpose. Users can access a data item if the data is listed in the set of necessary data accesses for the
assigned purpose. Finally, the assigned purpose should equal the intended purpose of the data. Thus, access control is fine grained.

In privacy preserving access control, a user must submit a purpose to access data. Some models ([JAV08], [HHS+03]) trust users and accept whatever purpose they provide while some other models ([BL08], [YBZ07]) authorize users (or roles) to certain purposes that they can use. Jafari et al. [JSS09] propose using workflow systems to govern data access where the access purpose is determined by the context of a workflow that issues the access request.

In a hospital, when patients are admitted for treatment, a workflow starts for giving treatment to the patients. Tasks in the workflow require access to patients’ data. Therefore, access request issued from the workflow will submit treatment as the access purpose that comes from the context of the workflow. Since a workflow can have sub-workflows, a purpose can have sub-purposes or more specific purposes. For instance, purpose treatment can have sub-purposes like examination and prescription.

Once the access purpose is determined, to check if the privacy policy allows access for that purpose, the authors suggest using one of the existing privacy preserving access control models. The main focus of this model is how to manage purpose at the enterprise level.

2.4.4 Access control models using XML based policy language

Access control models in this class include an XML based policy language. One interesting property of the policy language is its extensibility to suit the requirements of new policies.
eXtensible Access Control Markup Language (XACML) defines a general-purpose access control system [OX]. It offers an XML based policy language to specify the access requirements for a protected resource. The language uses a set of attributes that encode different properties of subject (data user), resource (data), action and environment. The semantics of the attributes are defined in an XML schema file.

The privacy profile of XACML, an extension of the policy language [OXP], includes two additional attributes: resource purpose that is the intended purpose for using a data item and action purpose that is the reason for using data by a data user. An access request is allowed when these two purposes match. The privacy profile also suggests a deny override algorithm in the case when more than one policy apply to an access request and one of them denies the request. In addition to the policy language, XACML includes a policy processing model that interprets policies in the relevant application's context.

XACML has several limitations. The policy language is complex and verbose. To encode a simple policy, it requires many lines of code. It also considers purpose as the only privacy requirement. Since the policy language can be extended, privacy components like visibility and retention can be implemented by defining them as attributes of the user and data, respectively.

IBM proposed a policy language, Enterprise Privacy Authorization Language (EPAL) [AHK+03], for encoding privacy policies. An EPAL policy is more like an access control rule or permission that can be enforced by an enforcement engine. Similar to XACML, the language is based on XML and uses a set of attributes called vocabularies. The semantics of the vocabularies are defined for the application domain in an XML schema file. Key
components of an EPAL policy are target, condition and obligation. The target consists of user categories, data categories, purpose and action. If applicable to an access request, a policy inference gives one of the three decisions—allow, deny and not applicable. One limitation of EPAL is that it does not support nested policies. It also applies a limited concept of role and role hierarchy.

2.5 Background literatures on delegation

Delegation has several characteristics [QRM+10]. Many delegation models have described in the literature where each model each delegation model contributes to a subset of these characteristics. This section gives an overview of the literature. The overview is categorized by different delegation characteristics.

2.5.1 Units of delegation

In role-based access control models, delegation units are roles and privileges. When users delegate their role, all the privileges assigned to that role are delegated. This is called full delegation. In partial delegation, users delegate a subset of their privileges. In some role-based delegation models [HMM10, ZOS03], users can create a new unit that contains both roles and privileges. After delegation, roles and privileges from the unit are assigned to delegatees. In workflow systems, typically the delegation unit is a task right. However, there are some role-based workflow systems where users can delegate roles as well as task rights.

In the identity management systems, users delegate access rights in the form of a credential [CO03]. A credential is a certificate containing the attributes of delegators and other
information about the delegation like delegator’s and delegatee’s public key, timestamp, etc. Attributes are equivalent to roles in RBAC. Delegators can create a credential containing all, or a subset of their attributes, and then delegate it to others. Two commonly used credentials are SAML assertions [OSM05] and X.509 certificates [CO03].

Privileges are used as the delegation unit in the proposed role-base delegation model. The model does not support delegating roles. More on this is described in Chapter 4.

2.5.2 Classification of delegation based on the type of delegator and delegatee

A delegator can be a user or the system itself. Users can delegate the access rights assigned to them. In addition, the system can delegate the access rights based on some delegation rules [ZOS03, CM10]. The receiver of a delegation can be a role or a user. When a role receives a delegation, all the members of that role can use the delegated access rights. Taking a combination of each type of delegator and delegatee, Joshi and Bertino [JB06] list four types of delegation: user-to-user, role-to-role, user-to-role and role-to-user delegation. This thesis studies user-to-user delegation where a user delegates access rights to another user.

2.5.3 Delegation across security domains

A security domain is a system where all entities of the system trust a common authority for authentication and authorization [YS96]. Generally, we can think of a business organization as a security domain. Delegation can happen inside a domain or across domains. This section presents the research works that study cross domain delegation. The terms domain and organization have been used interchangeably.
In a model proposed by Hasebe et al. [HMM10], data providers take services on many occasions in different domains and provide their personal information. Collaboration takes place between two domains where the users of one domain access data from another domain. To enable the collaboration, a domain adds the users from other domains as guest users. Each domain has two types of users - regular users and guest users. The delegation model supports that a regular user can delegate to another regular user or a guest user.

Hasebe et al. use a capability as the delegation unit which consists of roles and privileges. Being assigned to a capability that contains roles, a guest user can activate the roles and use all privileges that belong to those roles. If the capability contains privileges, a new role is created with those privileges. The new role is then assigned to the guest user. This is how the cross domain delegation is realized.

Gomi et al. [GHH+05] use SAML assertions as the authentication and authorization token in their model. Data providers allow users from different service providing domains to access their data. These users are assigned the assertions. To access data, they submit the assertions to the domain that stores the data. The domain storing the requested data is called the target domain.

The target domain validates the assertion and based on the validation, it either accepts or denies the request. To manage delegation, Gomi et al. use a trusted third party called the delegation authority (DA). The DA keeps the delegation policies which determine if a delegation between two domains is allowed. Suppose a user $A$ from one domain has the privilege to access a data item and it wants to delegate its privilege to a user $B$ from another domain. $A$ requests the DA to create an authorizing assertion for $B$. The DA checks the
delegation policies to see if the delegation is allowed. If so, the DA creates an assertion containing signatures that prove that authorization is passed from $A$ to $B$. In addition, the assertion contains information about what data a delegatee can use. With this assertion, $B$ gets access to data. Though a credential (or assertion) can be delegated from one domain to another domain, it should originally be issued by the target domain storing the data of interest.

Like the previous model, Chadwick et al. [CON06] use a similar setting where a federation of domains shares their resources with each other. Unlike the previous mode, it allows a credential originally issued by one domain can be used to access data in other domains too. Chadwick et al. give a real life example where people use credit cards issued by one company to buy products from another company. As explained in Section 2.5.1, credentials contain attributes which are equivalent to roles in RBAC. When a domain gets an access request from outside, it maps the attributes of the requesting users to its local attributes and gives accesses that are allowed for the local attributes. For example, a domain issues the attribute $Project Manager$ to a user which is mapped to the attribute $Project Coordinator$ when the user requests access to a target domain. In a federation of domains, a domain may trust a subset of other domains.

The authors propose an additional module called credential validation service (CVS) for the existing XACML framework [OX]. The module is used to interpret delegated credential. This work does not describe how delegation takes place from one domain to another. Rather, it describes how delegated rights are enforced.
When a user $u$ requests access to a resource in the target domain, the CVS of that domain retrieves the credentials issued to the user. If a credential is not issued by a trusted domain, the CVS checks the delegation chain of the credential. The delegation chain contains information about all delegations of the credential starting from the first delegator to the last delegatee. The CVS starts searching from the user $u$ in the chain and goes up. If it finds a domain that it trusts, the credential is valid. Once the CVS gather all the credentials for the requesting user, it maps the attributes of the credentials to the local attributes which are then forwarded to the policy decision point (PDP) module for taking access decision.

2.5.4 Delegation policy

Delegation policies are the rules that validate delegation requests. Some delegation models use explicit delegation rules which list delegator, delegatee and delegation unit. In some models, delegation policies are implicit. To give an example of such model, Graham et al. [GD72] define two versions of an access right: execution and delegation versions. If users are assigned the execution version, they can only use the right. Users can give a right to other users if they have the delegation version. Zhang et al. use a similar approach in their role-based delegation model PBDM [ZOS03]. They divide each role into two roles: delegatable and non-delegatable. Delegatable roles contain privileges that can be shared while privileges in a non-delegatable role cannot be shared. Both roles are assigned to users. They can use the privileges of both roles but can only delegate privileges in the delegatable role.

In addition to separating the privileges into delegatable and non-delegatable sets, the authors use an explicit delegation policies that state who can delegate what access right to
whom. An example of such policy is
\((rManager, rFrontDeskExec, (rAssistManager, priv5))\) states that the members of
role \(rManager\) can give the privilege \(priv5\) from the role \(rAssistManager\) to the
members of role \(rFrontDeskExec\).

As the explicit policy state what privilege of a role should be delegated, it can replace the
need for maintaining the delegatable and non-delegatable privilege sets. However, the
limitation of an explicit policy is that it requires all possible future delegation events be
encoded in the policies. However, in real life many delegation operations may be needed
which are hard to anticipate.

Crampton et al. [CK08-IJIS] propose delegation policies based on role hierarchy. In their
model, users can delegate a role if it is in the administrative scope of the roles assigned to
them. A role \(a\) is in the administrative scope of another role \(b\) if all the paths in the role
hierarchy, starting from role \(a\), go through role \(b\). If we consider the role hierarchy in
Figure 2.1, a user having role \(r5\) can delegate role \(r3\) but cannot delegate role \(r2\). Since,
only one of the two paths starting at role \(r2\) go through role \(r5\), role \(r2\) is not in the
administrative scope of role \(r5\).
The model also defines the eligibility for a delegatee to receive any access right. Delegatees can receive a role if it is greater than or equal to any of the roles already assigned to them. For example, a user having role \( r1 \) can receive \( r3 \) but a user having role \( r4 \) cannot receive \( r3 \) as these roles are not comparable.

Atluri et al. [AW05] propose a conditional delegation model for workflow systems. The authors use a role-based system where users are assigned to roles and roles are assigned to tasks. The delegation policy used by Atluri et al. is a variation of the explicit policy used by Zhang et al. [ZOS03]. The policy format is \((\text{delegator}, \text{access right}, \text{delegatee})\) where \text{delegator} and \text{delegatee} can be a user or a role and \text{access right} can be a role or a task. Atluri et al. also limit a delegation operation based on the time of delegation, the delegatee’s workload and the value of different task attributes. An example of a complete delegation policy used in this model is given below.

"Jamie can delegate his assigned task \( T12 \) to Johny if the workload of Johny is less than 5 tasks and delegation takes place over the weekend."
Here, this explicit delegation policy is supplemented with two conditions - workload of Johny should be less than 5 tasks and delegation should happen during the weekend.

2.5.5 Multistep delegation and revocation

If a delegatee can further delegate the access rights to other users, it is called multistep delegation. A majority of the delegation models confine their scope to single step delegation. However, a good number of delegation models exist in the literature [HMM10, WKB07, ZAC03, ZOS03] that study multi-step delegation.

Delegation models must have a policy for controlling multistep delegation. The policy determines if a delegatee can further delegate and if so, how many times it can delegate. The existing delegation models [WKB07, ZOS03] attempt to express this policy with a number $n$. Delegators give a value for $n$ to each delegated access right. Therefore, delegatees can further delegate the right for $n$ times. When delegatees delegate the right, they can also limit the number of users the new delegatees can forward the right to.

Multiple delegation of an access right creates a chain where each delegation operation is denoted by a directed edge. Revoking a privilege in multistep delegation model can result in cascaded revocation where revoking a privilege from a user also revokes from those who received the privilege from that user. Revocation is more complex in a model that does not restrict the case where a user can receive the same access right from multiple delegators. Such delegation results in a graph containing loops. Wainer et al. [WKB07] use a methodology where revoking a delegation from a user revokes any subsequent delegation.
edges from that user to other users in the graph. Revocation also deletes a user from the graph if there is no path left to reach it after deleting the edges.

Considering multistep delegation incurs many new challenges for delegation management, this thesis considers only single step delegation where delegates do not forward the access rights to other users.

Revocation can be initiated by delegators [ZAC03, WKB07]. In addition, some models study temporal delegation where the delegated rights last for a certain period of time [HMM10] and once the delegation period ends, the system itself initiates the revocation. In this thesis, both approaches are considered for revocation.

2.5.6 Security in Delegation

Delegation allows users to exchange access rights without the intervention of a system administrator. Malicious users can utilize it to exploit the system. Schaad [SCH03] showed how delegation can be used to violate a system’s security. Therefore, protecting security is very important in delegation. Several delegation models investigate security in delegation. They apply security policies like the separation of duty policy [SZ97] or other custom security policies.

Zhang et al. [ZAC03] consider a security policy that prevents a user from receiving two mutually exclusive roles at the same time. In their model, the system identifies these pair of conflicting roles before delegation. For each pair of conflicting roles, if users are assigned one of them, they cannot receive the other one by delegation. The authors deploy another similar policy that prevents two colluding users from exchanging any access rights. Like conflicting roles, the system should identify any pair of such users and write the policy that
will prohibit any delegation among them. However, in real life, collusions could involve more than two parties. Therefore, a secured delegation model should be able to prohibit delegation within a group of users or roles.

Wainer et al. [WKB07] use a security policy that prohibits certain relations between access rights and roles. For example, a person having a role \textit{Teller} cannot receive the right to purchase goods. The authors also apply policies ensuring that certain properties in the system are not violated by delegation. An example of such policy is that subordinate employees cannot have more privileges than their boss.

Wang et al. [WLC08] define a formal notion of security in delegation. According to that definition, if the numbers of tasks a set of users can do with and without the help of delegation are same, then the access control model is secure. Wang et al. describe three different methods for enforcing security in workflow systems. These are static, dynamic and source-based enforcement. In static enforcement, the system does not create any delegation assertion rule that leads to a situation that violates the aforementioned property. There is no generic method to identify a situation like this. Unlike static enforcement, dynamic enforcement allows the delegation to happen and then controls the usage of delegation in different instances of workflow. For a particular instance of a workflow, a list of users is created who would participate in the workflow. Once the list is ready, the system checks if these users could perform the workflow without any delegation by satisfying all the security constraints. If they could not, the workflow is cancelled and the system rolls back. In source-based enforcement, security constraints are enforced during a workflow's runtime. The system allows any delegation event between users and enforces the security
when they attempt to execute tasks. When users request a task execution, they mention the source or delegator of their privilege if it is a delegated privilege. The system treats the delegator as the actual task invoker and checks if the task request violates any security constraint.

This thesis proposes a role-base delegation model where the security policies are applied to maintain separation of duty among the users and to prohibit delegation to certain visibilities.

2.5.7 Privacy preserving delegation

There is little previous work investigating data privacy in delegation. One of these models [GHH+05] is based on the identity management systems where data providers get services from different service providers by giving access to their data. To provide a service, service providers often rely on each other and therefore, they need to share the data. The model does not reveal the true identity of the data provider to any of the service providers. A trusted third party is used to maintain a pseudo identity of the data provider for each service provider. When a data provider wants to provide access to their data to a service provider, the third party issues a credential containing the pseudo ID of the data provider for that service provider. If the service provider wants to delegate the credential to another service provider, the third party again creates a new credential containing the pseudo ID of the data provider for the new service provider.

Wohlgemuth et al. [WM06] propose another delegation model for the identity management systems where a data provider interacts with service providers through proxies. Proxies are the agents that use the data provider’s credential to receive services. When data providers
need a service, they delegate a part of their credential to a proxy. A proxy can again delegate the credential to other proxies. Through the mediation of a trusted third party, the data provider creates an anonymous credential for the proxy.

To support privacy preserving use of credentials, the model allows a data provider to specify the usage restriction of the delegated credential. A data provider specifies which service provider, which service of the service provider should use the credential, the number of times a credential should be used, a valid time period for usage, and if the proxy can delegate it further to other proxies. These usage restrictions are encoded inside the credential.

In both of these models, data providers remain anonymous; the authors claim that the privacy is not violated as a result of delegation. However, hiding only identity information may not be enough for privacy protection. Exposing other information like address and date of birth can lead to the identification of a data provider [Swe02].

Both models attempt to impose a usage restriction by specifying a valid time period for using the delegated credential. This is an essential requirement for privacy protection. In addition, data providers specify what program of a service provider and which system call of that program can access data as their privacy preferences in the model proposed by Wohlgemuth et al.. However, it may not be feasible for a data provider to obtain such information about a service provider in practice.

In a very recent work, Bussard et al. [BN10] propose a model to facilitate privacy policy transfer from data providers to data consumers. Data consumers are the parties who access data (e.g., service providers in the previous two models). The authors use P3P for privacy
negotiation among providers and consumers. If both agree, then data is shared with the agreed privacy policy. Data consumers are obliged to follow the privacy policy when they use data. The work supports multiple data transfer where data consumers can also be data providers if they give the data to other parties. Privacy negotiation in all subsequent transfers is the same as the one in the first transfer.

Bussard et al. propose an XML based preference language where each preference consists of applicability, access control, and usage control constraints. Applicability is the type of data to which the access control and usage control constraints apply to. Access control constraints list the users who can use data. Usage control constraints set the allowed purposes and obligation for the use. Usage control constraints may contain another constraint called *downstream constraint* which states if the authorized users can further share data with other users. Downstream constraint can be a preference itself by containing access control and usage control preferences. The access control preference in downstream constraint identifies which users can receive the data and the usage control preferences list purpose and obligation for them. It is interesting to note that the usage control constraint may contain another downstream constraint. Therefore, a preference may contain several other preferences inside it.

Data providers encode their privacy preferences in this language. Data consumers also use the same language to encode their practices with data. According to the terminology of this model, both are called preferences. The preferences of data provider and consumer are compared to see if the provider’s preference is equally or more expressive than the consumer’s preference. The authors define that a preference $A$ is equally or more expressive
than another preference $B$ if the set of allowed users, purposes, downstream constraints and obligations in $B$ are contained in the sets of corresponding entities in $A$. If the data provider’s preference is equally or more expressive, then the data is shared by attaching the agreed privacy policy. Any subsequent transfer of the data by the current consumer will be accepted if there is a downstream constraint in the attached policy that allows the transfer.

In this model, the privacy agreement between the providers and the consumers is later used as the access control policies for data items. In my work, the privacy agreement and the access control policies are separated. It gives an organization better control over data use as it can create more restrictive privileges than the agreement. Another limitation of this work is that it does not consider hierarchical roles and purposes. We use both role and purpose hierarchies in policy specification and comparisons.

2.5.8 Grant/Transfer delegation

In transfer delegation [CK08-IJIS], delegators lose a delegated access right. They cannot use the access right unless it is revoked from delegatee. On the other hand, both delegators and delegatees can use a delegated access right in grant delegation. In the proposed delegation model, the delegators do not lose the delegated right which makes it a grant delegation model.
Chapter 3
Access control model: Platform for delegation

This chapter introduces a privacy model that formalizes privacy policies. It also explains the relationship of the privacy model to the access model. The chapter is organized as follows. Section 3.1 describes the privacy model. Section 3.2 relates the privacy model with the access control model. The access control model is described over Sections 3.2 and 3.3.

3.1 Privacy model

Privacy policy defines the data collector’s practice with the data. In this work, an enterprise model is followed where the data collector sets the necessary data usages to serve the data providers in their privacy statement. Data providers read these policies and accept them. The collector may ask the data providers’ preferences or even allow the providers to opt out from some policies. Examples of enterprise privacy policies include the privacy policies of Amazon.com [AMA10], Toys.com [TOY10], eBay.com [EBA10], etc.

Each privacy policy consists of data, action, purpose, condition and obligation. This definition is adapted from [AHK+03, NTB+07]. Here, data is the information being collected about the data provider. Usually, actions are read, write, etc. Purposes are the reasons for using the collected information. Conditions are Boolean expressions that are
used to validate contextual information necessary to enforce the policy. Obligations are the

task that one must do as a result of accessing data items. Consider an imaginary policy of a

website saying, “Every time we use data for marketing purpose, we will inform you by

Emails”. Here, the obligation of accessing data is to send emails. In short, a privacy policy
states that the authorized users are allowed to perform the action on the data only for the

specified purpose while satisfying the condition and obligation.

Let the sets of data, actions, purposes, conditions and obligations be denoted by \( D, ACT, P, C, \) and \( OB \), respectively. The sets of privacy policies, denoted by \( PPolicy \), is defined over

the following range:

\[
PPolicy \subseteq D \times ACT \times P \times C \times OB
\]

Data is accessed by different visibilities [BAB+09] which are parties involved in the

business operations. For example, Toys.com states in its privacy policy that a customer’s

information are accessed by its employees, service providers, business partners and

advertisers [Toy10], thereby defining four visibilities for the collected data. Here, one of

the visibilities collects the data and share them with other visibilities. Let the data collecting

visibility be termed as \textit{enterprise visibility}. Data usages by each visibility should be defined

\textit{i.e.}, no visibility can use data in cases other than the defined ones.

Let the set of visibilities be \( VIS \). As privacy policies may vary across visibilities, a separate

set of privacy policies for each visibility is defined. Let the privacy policy set for a

visibility \( i \) be \( PPolicy_i \) where \( i \in VIS \). The union of privacy policy sets for all the

visibilities should constitute the entire policy set for an enterprise.

\[
PPolicy = \bigcup_{i \in VIS} PPolicy_i
\]
An enterprise may allow data providers to specify their preferences for particular policies. For example, a privacy policy may state that the consent for using personal information for marketing purpose is optional and customers can opt out from such usage of their data. Data provider’s preferences and other related information necessary to enforce privacy policies are called privacy policy 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. Here, $\emptyset$ denotes power set.

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

**Example of the privacy model with a sample policy:**

In this example, the privacy model is mapped to a relational database that stores the privacy policies. The example uses a real life privacy statement taken from the website Toys.com that sells toys.

“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.”

The policy allows the website’s employees (mentioned as "R" Us Family members) to access the customers’ home addresses, telephone numbers and email addresses 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.
Table 3.1: Customer information

<table>
<thead>
<tr>
<th>Provider ID</th>
<th>Name</th>
<th>DofB</th>
<th>Gender</th>
<th>Home address</th>
<th>Telephone number</th>
<th>Email address</th>
</tr>
</thead>
<tbody>
<tr>
<td>235</td>
<td>Smith Jonas</td>
<td>02/23/1970</td>
<td>Male</td>
<td>60 Essex st., Toronto, ON</td>
<td>89450</td>
<td><a href="mailto:Smith78@search.com">Smith78@search.com</a></td>
</tr>
</tbody>
</table>

Table 3.2: Privacy policies, $PPolicy_{ws}$

<table>
<thead>
<tr>
<th>Policy ID</th>
<th>Data Name</th>
<th>Action</th>
<th>Purpose</th>
<th>Condition</th>
<th>Obligation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Name</td>
<td>Read</td>
<td>Promotion</td>
<td>$OptOut \neq Y$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>2</td>
<td>Home address</td>
<td>Read</td>
<td>Promotion</td>
<td>$OptOut \neq Y$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>3</td>
<td>Telephone number</td>
<td>Read</td>
<td>Promotion</td>
<td>$OptOut \neq Y$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>4</td>
<td>Email address</td>
<td>Read</td>
<td>Promotion</td>
<td>$OptOut \neq Y$</td>
<td>$\emptyset$</td>
</tr>
</tbody>
</table>

Table 3.3: Privacy metadata, $StatDS_{ws}$

<table>
<thead>
<tr>
<th>Provider ID</th>
<th>Policy ID</th>
<th>OptOut</th>
</tr>
</thead>
<tbody>
<tr>
<td>235</td>
<td>1</td>
<td>N</td>
</tr>
<tr>
<td>235</td>
<td>2</td>
<td>Y</td>
</tr>
<tr>
<td>235</td>
<td>3</td>
<td>Y</td>
</tr>
<tr>
<td>235</td>
<td>4</td>
<td>N</td>
</tr>
</tbody>
</table>

As the data is being used by the website’s employees, let the visibility be termed as website (denoted by the acronym ws). It is also the enterprise visibility in this data collection. Table 3.1 presents all the information collected about the customers with a sample record. Table 3.2 is an instance of the relation $PPolicy_{ws}$ that stores the above privacy statement by breaking it into several policies. Table 3.3 is an instance of the relation $StatDS_{ws}$ that
stores the preferences of the customers in a variable $OptOut$ regarding the policies. When the variable has the value ‘Y’ for a particular data provider and a privacy policy, it indicates that the provider has opted out from the policy.

A purpose hierarchy is considered in this work. Typically, purpose hierarchy is a partial order having the reflexive, transitive and anti-symmetric properties. In this thesis, purpose hierarchy is considered to be a tree.

Purpose hierarchy is $(P, \leq)$ where $\leq$ is a binary relation defined over the purposes $P$. In order to be a tree, the purpose hierarchy meets the following conditions [BRV04]:

1. There is a unique purpose $root \in P$ such that $(\forall p \in P \land root \leq^* p)$ where $\leq^*$ is the reflexive transitive closure of $\leq$.
2. Every purpose which is not the root has a predecessor i.e., $(\forall p \in P \land root \neq p \land \exists p' \in P \land p' \leq p)$.
3. $\leq$ is acyclic i.e., $\forall p \in P. \neg(p \leq^+ p)$ where $\leq^+$ is the transitive closure of $\leq$.

Here, $(p1 \leq p2)$ is read as $p2$ subsumes $p1$. The absence of relation between purposes is denoted by an operator $\circ$. If $(p1 \circ p2)$, then neither $(p1 \leq p2)$ nor $(p2 \leq p1)$ is true.

The use of purpose hierarchy makes the policies more expressive. Consider the privacy policy #4 in Table 3.2 which is $(Email\ address, read, Promotion, OptOut \neq Y, \emptyset)$. It allows accessing email address for purpose $Promotion$ as well as for any purpose that subsumes $Promotion$ in the purpose hierarchy. Figure 3.1 shows a sample purpose hierarchy where $Promotion$ is subsumed by $Contest$ and $eMarketing$. Condition and obligation for these purposes are the same as for $Promotion$. 
Figure 3.1: Purpose hierarchy

It can be concluded that the purpose mentioned in the privacy policy acts as the lower bound of allowed purposes when the purpose hierarchy is used. Purposes that are equal to the lower bound or subsume the lower bound can be used to access the data. A policy can be overridden by a more fine-grained policy. Consider the following policy which states that to use an email address for *eMarketing* purpose, two conditions must be true – data providers have not opted out from the policy and their age is greater than 18 years.

Privacy policy#5:

\[(Email\ address, read, eMarketing, OptOut \neq Y \text{ AND } Age > 18, 0)\]

This policy overrides policy #4 by increasing the data usage restriction for *eMarketing* purpose. However, to use email address for other purposes (e.g., *Promotion* and *Contest*), only one condition needs to be satisfied as stated by policy #4.
3.2 Privilege model

In a privacy preserving model, an enterprise creates privileges from the privacy policies. It then assigns privileges to the data users (e.g., its employees). When an enterprise creates privileges for the users, it may create privileges more restrictive than the privacy policies by choosing a subset of the allowed purposes of a privacy policy. To specify the subset, a purpose range is used in the privilege structure instead of a single purpose.

A purpose range is given by \( pr =< p_u, p_b > \) where \( p_b \leq p_u \). 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, \( dp \in DP \), \( dp =< Email \text{ address, read}, < Promotion, eMarketing > \) is a privilege based on policy #4 and policy #5 described in the previous section. 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.

3.3 Formal specification of the access control model

The access control model uses the following entities:

- \( VIS \) is a 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. For example, $dp. d$ denotes the data contained in the privilege $dp$.

Consider 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 $(R_i, \leq_{R_i})$ where $\leq_{R_i}$ is a binary relation defined over the roles $R_i$. In order to be a tree, the role hierarchy meets the following conditions:

I. There is a unique role $root \in R_i$ such that $(\forall r \in R_i \land root \leq_{R_i} r)$ where $\leq_{R_i}^*$ is the reflexive transitive closure of $\leq_{R_i}$.

II. Every role which is not the root has a predecessor i.e., $(\forall r \in R_i \land root \neq r \land \exists r' \in R_i \land r' \leq_{R_i} r)$.

III. $\leq_{R_i}$ is acyclic i.e., $(\forall r \in R_i \land !(r \leq_{R_i}^+ r))$ where $\leq_{R_i}^+$ is the transitive closure of $\leq_{R_i}$.

• 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 instantiates $PrivS$ with the privileges assigned to the user $u$ and returns true.
\[\text{hasRights}: U \times \wp(DP) \rightarrow \mathbb{B}\]

- The set of user-privilege assignments for all the users of visibility \(i\) is denoted by \(UADP_i\). This set is computed by using the predicate \(\text{hasRights}\) that finds privileges for each user. \(UADP_i\) is computed by combining all user-privilege assignments.

### 3.4 Access control

Notations used in access request validation are given in the following:

- A user submits data, action and purpose as the access request. Formally,

  Access request, \(oar = <d, a, p>\) where \(d \in D, a \in ACT, p \in P\).

- A predicate used in the access request validation is described as follows.

  \[\text{ComplianceAC}(dp, oar):\]

  The predicate accepts two arguments – privilege \((dp)\) and access request content \((oar)\).

  Let the purpose range of privilege \(dp\) is denoted by \(pr_{dp}\) where \(pr_{dp}.p_u\) be the upper bound and \(pr_{dp}.p_b\) be the lower bound of the range. The predicate evaluates the following logic expression and returns a Boolean value.

  \[
oar.d = dp.d \land oar.a = dp.a \land pr_{dp}.p_b \leq oar.p \land
  \]

  \[
oar.p \leq pr_{dp}.p_u
  \]

When the predicate returns true, we say that the privilege complies with the access request. Finally, access request evaluation is done using the following logic expression:

\[
\text{hasRights}(u, PrivS) \land \exists dp \in PrivS. \text{complianceAC}(dp, oar)
\]
The successful inference of the expression grants the access request. The expression tests if the requesting user is assigned a privilege which complies with the request.

The amount of data that is revealed to the user depends on the condition of the privacy policy. For instance, a condition $OptOut \neq Y$ will rule out the data of those providers who have opted out from the privacy policy. Therefore, another predicate is needed which will find the appropriate privacy policy for the access request and return the condition. In addition to enforcing condition, obligation must be performed. Details of condition and obligation enforcement are beyond the scope of this work. The research works [KZB+08, NBL08, OX] give more insight to these areas.

Note that in RBAC, session is a mapping of a user to an activated subset of the roles assigned to the user. To incorporate session in our work, we need an additional check before granting access request (also delegation request) if the privilege in question comes from an active role assigned to the requestor. In addition, delegation constraints like static separation of duty, visibility-conflict constraint and duplicate delegation constraint (described in the next chapter) are not affected if the session is considered. So we can conclude that the session doesn’t change the behaviour of our proposed model which is also the reason that we haven’t considered it in our model.
Chapter 4
Delegation

This chapter presents the proposed delegation model. Section 4.1 gives an overview of the delegation model including key definitions. Section 4.2 presents a formal specification of the model including delegation request and relations for storing delegation records. Delegation policies and constraints used in the model are described in Section 4.3. Section 4.4 includes the algorithms that show the entire delegation process in a step by step fashion. Finally, Section 4.5 describes how delegated access rights are revoked.

4.1 Introduction

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 and also sets the data usage guidelines. We also study how two users of a legitimate party share their access rights. The access control model described in the previous chapter groups users into different parties or visibilities. Delegation can occur among the users with the same or different visibilities. The visibilities of the delegator and delegatee are referred as source and destination visibility, respectively. A delegation event where the delegator and delegatee are from the same visibility, is called intra-visibility delegation. If the delegator and delegatee are from the different visibilities, the event is called inter-visibility delegation.
An access right which is about to be delegated from one visibility to another, contains the privacy policy for the source visibility. In inter-visibility delegation, the policy for the destination visibility may be different. Therefore, the policy of the access right should be updated so that the data usage in the destination visibility is guided by the appropriate policy. In contrast, the privacy policy of data is the same for all users within a visibility. Therefore, in intra-visibility delegation the privacy policy of the delegated access right remains the same as before.

Delegations among users are controlled using delegation policies. These are the rules that state what delegation operations are valid. Separate delegation policies are defined for intra- and inter-visibility delegation. All the delegation operations in this model are controlled by one of these policies. Besides the delegation policies, several constraints are applied to maintain separation of duty among the users, to prevent users from receiving duplicate privileges and to prevent users from delegating to certain visibilities.

Revocation is the process of removing delegated access rights from users. In this work, two types of revocation are considered. First, all delegated rights last for a specific time period and once the period ends, the rights are revoked automatically. Secondly, delegators are allowed to revoke the rights they delegated in the past. To support these revocations, two separate policies are applied.

The architecture of the delegation model can be given by the figure 4.1.
Chief privacy officer (CPO) uses the policy specifier to create a privacy policy database. Based on the policies, CPO assigns access rights through the policy specifier to data users which are stored into the authorization records. When a data user submits a delegation request to the delegation agent, the agent verifies the request by validating delegation policies. To validate the delegation policies, the agent needs the authorization records and privacy policies. If the request is valid, the agent requests the policy specifier to add a privacy policy for the delegatee and also, add an access right for the delegatee to the authorization records. In response to the request, the policy specifier checks if the new
policy is in conflict with any other policies in the policy database. If not, then it adds the policy and also, assigns the corresponding right to the delegatee.

We simplify the above architecture by removing some agents. In the actual architecture, we only keep the data user and remove the CPO by assuming that the privacy policies and the authorization records are already created. We also remove the policy specifer from the architecture. Previously, when a delegation takes place into the system, a policy is added to the privacy policy base as well as an authorization record is added for the delegatee. Since delegation is an assignment of access right, our architecture does not add the policy anymore, rather the delegation agent adds only the authorization record. By this simplification, we also avoid the check for policy conflict. We also add a history module into the architecture that logs the delegation related events into the system.

Figure 4.2 presents the simplified architecture that we have used in our work. The delegation agent processes delegation and revocation requests. If it is a delegation request, the agent retrieves the delegation policy and constraints that apply to the request 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 that are described in Section 3.3 and Section 4.2. These are the records of both regular and delegated user-access rights assignments. The privacy policies are the agreements between the data provider and the collector for using the data items. Delegation histories are records of the delegation events taken place in the system so far.
If a delegation request satisfies all the policies and constraints, a new record is added to the authorization record assigning delegatee to the delegated right. The delegation event is also logged in the delegation histories. The delegation process modifies the authorization records which are used for access control.

When a user submits a revocation request, the delegation agent retrieves the revocation policy (module DRP). The policy is verified using the delegation history. The agent responds to valid requests by removing the corresponding user-right assignment from the authorization records. Like other events, revocation is logged in the revocation histories (module DRH). Revoking access rights from users can also happen if the delegation time periods are over. Such event is not initiated by a user request, but by the system itself.

Figure 4.2: Delegation model
However, the policy inference and the event log are also done in this revocation like other events.

The delegation model will be described in detail over the next few sections.

4.2 Specification of the delegation model

This section defines the semantics of 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 called as full and partial delegations, respectively.

Since the privacy policies are the same for all the users with the same visibility, role delegation within a visibility is not affected by the privacy policy. Studying role delegation in this work would then be no different than existing literatures. Moreover, role delegation is addressed by most of the existing delegation models.

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 a role of the destination visibility.
This thesis studies partial delegation where a user can delegate individual privileges to other users but not the entire role.

**Delegation request:** A privilege in our access control model gives users access to a data item for a range of purposes. Users may want to delegate the privilege for a subset of the purposes that they have. Thus, a delegation request consists of data, action, and a purpose range. A delegation request is given by the tuple $odr = <d, a, pr>$ that requests performing action $a$ on data $d$ for purposes included in the range $pr$. Tuple $odr$ is the content of the delegation, referred as a *delegation object*. The complete request will contain additional information including delegator, delegatee, and *valid time*. All the delegated access rights in the proposed model last for a specific period of time. Valid time is a system time when a delegated access right will be removed from a delegatee. Let $Z$ be the set of times in the system clock. The complete delegation request is given by the following:

$$dr = <u, u', odr, etm>$$

Here, $odr$ is the content of delegation, $u$ and $u'$ are delegator and delegatee, respectively and $etm$ is the valid time where $\{u, u'\} \in U$ and $etm \in Z$.

A predicate called *compliance*, is used to validate delegation requests. It is similar to the predicate $complianceAC$ used in Chapter 3. The predicate $compliance$ is described in the following:

$compliance(dp, odr)$: The predicate tests if the delegation object $odr$ is contained by the privilege $dp$ which involves checking if both $dp$ and $odr$ have the same data and action
and the purpose range of $dp$ contains the purpose range of $odr$. The predicate returns true if the following comparisons are true.

1. $dp.d = odr.d$
2. $dp.a = odr.a$
3. $pr_{dp} \cdot p_b \leq pr_{odr} \cdot p_b \land pr_{odr} \cdot p_u \leq pr_{dp} \cdot p_u$

The structures of privilege and delegation request are the same. Therefore, instead of a delegation request, the predicate can also accept another privilege as an argument. In the predicate $compliance$, both arguments contain purpose range while in the predicate $complianceAC$, only one argument contains a purpose range. This is the difference between these two predicates.

Relations storing delegation records: In RBAC, users have direct relation with roles through the user-role assignment while they have indirect relation with privileges as the relation is derived through user-role and role-privilege assignments. In the proposed delegation model, a user may receive 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 that records user-privilege assignments including the valid time.
- Each delegation operation is logged by the system. In addition to information about the delegator, delegatee and access right, a log includes the time when a delegation takes place and its valid time.
Logging uses a relation $DH$ that has the following format:

$$DH \subseteq U \times U \times DP \times Z \times Z$$

An entry $(u1, u2, o, etm, stm) \in DH$ denotes that user $u1$ has delegated an access right $o$ to user $u2$ at time $stm$ which will expire at time $etm$.

### 4.3 Delegation policies and constraints

To control delegation between users, three types of rules are applied: delegation policies, security policies and duplicate delegation policies. Delegation policies are assertive rules that specify what types of delegation operations are allowed.

The separation of duty principle [SZ97] is applied as one of the security policies. Sometimes enterprises may want to prohibit its users to transfer access rights to certain visibilities; so another security policy is designed to prohibit the delegation between two visibilities. Finally, duplicate delegation policy prevents users from receiving access rights they already have through delegation. Security and duplicate delegation policies are designed as prohibitive rules. A delegation operation is permitted if it is allowed by a delegation policy and not declined by any security or duplicate delegation policy. The security and duplicate delegation policies are often referred as constraints to differentiate them from the delegation policies. The language for expressing all the policies and constraints is described in the following.

**Policy language:**

All the policies and constraints are written in the form of a logic programming clause. However, to make the policies easier to understand, we define the policy language as a
variant of First Order Logic. The language consists of a set of variables and constants. Let $v$ represents variables and $cn$ represents constants. 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 terms, then $pc(tm_1, tm_2,.., tm_n)$ is an atomic formula.

All the well-formed formulas are in the following Backus Naur Form (BNF):

$$
\phi ::= T | pc(tm_1, tm_2,.., tm_n)|(-\phi)|(\phi_1 \land \phi_2)|(\phi_1 \leftrightarrow \phi_2)|(\exists v. \phi)|(\forall v. \phi)
$$

Here, instead of $\phi_2 \rightarrow \phi_1$, we write $\phi_1 \leftrightarrow \phi_2$ for convenience.

All the formulas that are used to enforce delegation policies and constraints are written in the form of a definite program clause [Llo84].

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

Left and right side of the formula is called head and body, respectively. The semantics of the clause is that if the body is true, the head is true.

When the body of the clause is the Boolean constant $T$, it is called called unit clause [Llo84].

$$
\phi \leftarrow T
$$

The semantics of the unit clause is that the head is true. This type of clause is used to enter information into the system.

Note that First Order Logic uses the notion of function. While the predicate always returns Boolean values, the function can return values other than Boolean. In the language
described above, the notion of function is embedded into the predicates. For example, if \( hasRights(u, PrivS) \) is a predicate that returns true when user \( u \) has privileges as well as returns the privileges that the user has by instantiating the variable \( PrivS \).

The predicates used in this thesis 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 Bertino \( et al. \) [BFA99], the semantics are not the same. The semantics of the predicate classes are given 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 rules. Being used as the head of a rule, these predicates denote the consequence when they become true. Constraint predicates are used to enter security policies into the system. All the predicates used in this work are listed in Table 4.1-4.4. Here, \( \mathbb{B} = \{true, false\} \), \( N \) is the set of natural numbers and \( \emptyset \) denotes power set.
<table>
<thead>
<tr>
<th>Predicates</th>
<th>Functionality</th>
</tr>
</thead>
<tbody>
<tr>
<td>compliance:</td>
<td>compliance(dp, odr) returns true if the expression holds: dp.d = odr.d ∧ dp.a = odr.a ∧ $pr_{dp}p_b \leq pr_{odr}p_b \land pr_{odr}p_u \leq pr_{dp}p_u$</td>
</tr>
<tr>
<td>DP × ODR → ℕ</td>
<td>equality(dp, odr) returns true if the expression holds: dp.d = odr.d ∧ dp.a = odr.a ∧ $pr_{dp}p_b = pr_{odr}p_b \land pr_{odr}p_u = pr_{dp}p_u$</td>
</tr>
<tr>
<td>equality:</td>
<td>combine(SetA, odr, SetA') instantiates SetA' by doing the following operation: $SetA' = SetA \cup {odr}$</td>
</tr>
<tr>
<td>DP × ODR → ℕ</td>
<td>combine(SetA, odr, SetA') instantiates SetA' by doing the following operation: $SetA' = SetA \cup {odr}$</td>
</tr>
<tr>
<td>combine:</td>
<td>common(ST1, ST2, k) instantiates k with the number of privileges of the set ST1 that matches(^1) with one of the privileges in the set ST2. Pseudo code for the predicate is given Appendix A, Algorithm 6.</td>
</tr>
<tr>
<td>$\emptyset(DP) \times ODR \times \emptyset(DP) → ℕ$</td>
<td>expired(Z) returns true if the time lt is expired by the system clock. If the set Z of system times is totally ordered and ct is the current system time, the predicate returns true if $ct = lt + t$ where $t \in Z$.</td>
</tr>
<tr>
<td>common:</td>
<td>$\emptyset(DP) \times \emptyset(DP) \times N → ℕ$</td>
</tr>
<tr>
<td>$\geq: N \times N → ℕ$</td>
<td>$=: D \times D → ℕ$</td>
</tr>
<tr>
<td>$=: ACT \times ACT → ℕ$</td>
<td>$=: P \times P → ℕ$</td>
</tr>
</tbody>
</table>

\(^1\) Here, match does not mean total equality. The definition of match is explained in Section 4.3.2 and also in Appendix A.
### Table 4.2: Specification predicates

<table>
<thead>
<tr>
<th>Predicates</th>
<th>Functionality</th>
</tr>
</thead>
<tbody>
<tr>
<td>$vis: U \times VIS \rightarrow \mathbb{B}$</td>
<td>$vis(v, j)$ instantiates $j$ with the visibility of the user $v$.</td>
</tr>
<tr>
<td>$hasRights: U \times \varnothing(DP) \rightarrow \mathbb{B}$</td>
<td>$hasRights(u, PrivS)$ instantiates $PrivS$ with the privileges assigned to the user $u$. Pseudo code of the predicate is given in Appendix A, Algorithm 7.</td>
</tr>
<tr>
<td>$hasDelegation: U \times \varnothing(TUADP) \rightarrow \mathbb{B}$</td>
<td>$hasDelegation(v, TPD)$ instantiates $TPD$ with all the current delegation records where user $v$ is the delegatee.</td>
</tr>
<tr>
<td>$object: TUADP \times DP \rightarrow \mathbb{B}$</td>
<td>$object(tpd, dp)$ instantiates $dp$ with the privilege of the delegation record $tpd$.</td>
</tr>
<tr>
<td>$time: TUADP \times \mathbb{Z} \rightarrow \mathbb{B}$</td>
<td>$time(tpd, lt)$ instantiates $lt$ with the delegation time from the record $tpd$.</td>
</tr>
<tr>
<td>$grantor: TUADP \times U \rightarrow \mathbb{B}$</td>
<td>$grantor(tpd, u)$ returns true if the user $u$ is the grantor in a delegation event that generated the record $tpd$.</td>
</tr>
<tr>
<td>$dlookup: VIS \times ODR \rightarrow \mathbb{B}$</td>
<td>$dlookup(j, odr)$ returns true if the following expression holds. $\exists pp \in PPolicy_j \land odr.d = pp.d$ $\land odr.a = pp.a \land pp.p \leq pr_{odr}.p_b$</td>
</tr>
</tbody>
</table>

### Table 4.3: Decision predicates

<table>
<thead>
<tr>
<th>Predicates</th>
<th>Functionality</th>
</tr>
</thead>
<tbody>
<tr>
<td>$canGiveSV: U \times U \times DP \times ODR \rightarrow \mathbb{B}$</td>
<td>If $canGiveSV(u, v, dp, odr)$ is true, it denotes to the fact that a user $u$ can delegate a privilege containing $odr =&lt; d, a, pr &gt;$ to a user $v$ of same visibility. The eligibility of the user $u$ comes from having the privilege $dp$.</td>
</tr>
<tr>
<td>$canGiveDVC: U \times U \times DP \times ODR \rightarrow \mathbb{B}$</td>
<td>If $canGiveDVC(u, v, dp, odr)$ is true, it denotes to the fact that a user $u$ can delegate a privilege containing $odr =&lt; d, a, pr &gt;$ to a user $v$ from a different visibility. The eligibility of the user $u$ comes from having the privilege $dp$.</td>
</tr>
</tbody>
</table>
canGiveDVE:  
\[U \times U \times DP \times ODR \rightarrow \mathbb{B}\]  
If \(\text{canGiveDVE}(u, v, dp, odr)\) is true, it denotes that a user \(u\) can delegate a privilege containing \(odr = (d,a,pr)\) to a user \(v\) from a different visibility. The eligibility of the user \(u\) comes from having the privilege \(dp\).

error:  
\[U \times U \times ODR \rightarrow \mathbb{B}\]  
If \(\text{error}(u, v, odr)\) is true, it denotes that delegation of \(odr\) from user \(u\) to \(v\) is not permitted.

autoRevoke:  
\[\{SYS\} \times U \times ODR \rightarrow \mathbb{B}\]  
If \(\text{autoRevoke}(SYS, v, o)\) is true, it denotes that the system will revoke a privilege \(o\) from a user \(v\).

canRevoke:  
\[U \times U \times ODR \rightarrow \mathbb{B}\]  
If \(\text{canRevoke}(u, v, o)\) is true, it denotes that the system will revoke a privilege \(o\) from a user \(v\) on behalf of a user \(u\).

<table>
<thead>
<tr>
<th>Predicates</th>
<th>Functionality</th>
</tr>
</thead>
<tbody>
<tr>
<td>(aUserCannotHave):</td>
<td>If (aUserCannotHave(j, MDP, k)) is true, it denotes that a user cannot have (k) or more privileges from the set (MDP) from visibility (j).</td>
</tr>
<tr>
<td>(VIS \times \varnothing(DP) \times N \rightarrow \mathbb{B})</td>
<td></td>
</tr>
<tr>
<td>(conflictingVis):</td>
<td>If (conflictingVis(vis1, vis2)) is true, it denotes that delegation from (vis1) to (vis2) is not permitted.</td>
</tr>
<tr>
<td>(VIS \times VIS \rightarrow \mathbb{B})</td>
<td></td>
</tr>
</tbody>
</table>

**Table 4.4: Constraint predicates**

4.3.1 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 proposed a policy that states that a user cannot give an access right they do not have to other users; it is called the *principle of privilege attenuation* (POPA) [Bis02, Den76, Fon11]. Note that this principle imposes restriction only on delegators. The delegation policies defined in this work are motivated by this principle.
Policy for intra-visibility delegation:

The following policy is used to validate an intra-visibility delegation request:

*Users are eligible to participate in a delegation operation as a delegator if they have privileges that comply with the requested delegation object.*

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

\[
\text{canGiveSV}(u, v, dp, odr) \leftarrow \text{hasRights}(u, \text{PrivS})
\]

\[\wedge \exists dp \in \text{PrivS.compliance}(dp, odr)\]

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 privileges is then probed for the existence of a privilege \(dp\) that contains the delegation object \(odr\) using the predicate \(\text{compliance}\). The probe stops when such a privilege is found and then returns true or when there is no such privilege, false is returned. If the body returns 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 POPA principle as it requires that the delegators must have the privileges they want to delegate.

In intra-visibility delegation, both delegator and delegatee are from the same visibility. 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.
Policy for inter-visibility delegation:

In an inter-visibility delegation, access rights to data items are delegated from one visibility to another. To simplify the delegation model, it is assumed that 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 3.1 where the enterprise visibility is the website itself. All other visibilities receive access rights from the enterprise visibility.

Inter-visibility delegation are divided into two sub-classes: collaboration and exchange. In collaboration, users of the source visibility delegate access rights to the users of the destination visibility so they can collaborate to achieve the same goal. For example, an organization allows its employees and an external service provider to access its customer’s information for doing market research. However, the source and destination visibilities have separate business operations in exchange; but they share their customer’s information for their mutual benefits. For example, an organization shares their customer information with their parent and subsidiary companies.

Delegation policy for collaboration:

As two visibilities collaborate to achieve the same goal, it is expected that data is visible to both of the visibilities with the same privacy policies. It also allows the delegation policy to follow the POPA principle. So the delegation policy for collaboration is the following:

Users are eligible to participate in a delegation operation as a delegator if they have privileges 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 POPA principle that asks that the delegator must have a privilege containing the requested object. The second part of the policy asks that the delegation object must be supported by a privacy policy of the destination visibility. The rule validating a delegation request with this policy is given in the following.

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

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 any 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.

\[ \forall l \in PPolicy_d \exists l' \in PPolicy_s. (l.d = l'.d \land l.a = l'.a \land l.p = l'.p) \]

Here, \( PPolicy_s \) and \( PPolicy_d \) 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 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 delegation, no users from the source visibility will be involved.

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 entire 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.

\[
\text{canGiveDVE}(u, v, dp, odr) \leftarrow \text{hasRights}(u, \text{PrivS})
\]

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

The rule first checks if the delegator has a privilege \(dp\) among the privileges of the set \(\text{PrivS}\) that contains the requested data \((odr. d)\). Next, using the predicate \(\text{vis}\) 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 \textit{i.e.}, if the following expression holds.

\[ \forall l \in PPolicy_d \exists l' \in PPolicy_s. (l.d = l'.d) \]

4.3.2 Security policies

Delegation allows users in a system to share their access rights without the mediation of a system administrator. Malicious users can use delegation to abuse the system. More on this issue is described in Section 2.5.6. To protect the system’s security, delegation should be constrained by security policies. This section describes the security policies used in this thesis.

\textbf{Separation of duty (SoD) policy:}

SoD policies are used to distribute tasks among multiple users to prevent fraud. The policies can be divided into three classes: static separation of duty (SSoD) policy [LTB07], dynamic separation of duty (DSoD) policy [SZ97] and Chinese Wall policies [BN89]. DSoD and Chinese Wall policies use execution histories of the system. Rather than preventing users from being assigned to access rights, these policies prevent users at runtime from using access rights that conflict with some of the access rights they used before. Since delegation is a process of assigning access rights to users, only SSoD policies are considered in this work and other policies are left for runtime enforcement. SSoD policies are briefly described next.
Static separation of duty (SSoD) policy [LTB07]: When a sensitive task is comprised of \( n \) steps, SSoD policy requires the cooperation of at least \( k \) (for some \( k \leq n \)) different users to complete the task.

Consider an example SSoD policy that says at least three users together can have the privileges \( \{dp_1, dp_2, dp_3, dp_4\} \). The policy can be written in the format \(<\{dp_1, dp_2, dp_3, dp_4\}, 3\>\).

Li et al. [LTB07] show that the problem of enforcing SSoD policies cannot be solved in polynomial time. An SSoD policy involves \( k \) users where \( k \) can be an arbitrarily large number. Li et al. show that if the policy can be converted to one or more sub policies and if each sub policy involves a single user, then the problem can be solved in polynomial time. Let the sub policies 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 \(<\{dp_1, dp_2, dp_3, dp_4\}, 3\>\) that says at least 3 users can have the privileges altogether. Suppose, \(<\{dp_1, dp_2, dp_3\}, 2\>\) is an SSoD' policy stating that no user can have 2 or more privileges from the specified set. Enforcing this SSoD' will lead to a state where three or more users will have these privileges \( \{dp_1, dp_2, dp_3, dp_4\} \) which also satisfies the SSoD policy. The idea of using SSoD' policies for maintaining separation of duty is adapted in this work. The rest of this section describes how SSoD' policies can be expressed using the policy language of this model.
General format for a SSoD’ policy is \(< MDP: \{dp_1, dp_2, \ldots, dp_f\}, k >\) that says a user cannot have \(k\) or more privileges from the set \(MDP\). The following rule is used to configure the system with the SSoD’ policies for a visibility \(j\).

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

Since the body of the rule is true, the head is also true and the system will accept the information entered using the predicate \(aUserCannotHave\). The rule that enforces the entered policy is defined below. Let \(u\) be the delegator, \(v\) be the delegatee, and \(odr\) be the delegation object.

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

The rule works as follows. Using the predicates \(vis\) and \(aUserCannotHave\), all the SSoD’ policies of the delegatee’s visibility are returned. Until the rule becomes true for one of the security policies, the following things are performed for each policy. The privileges assigned to the delegatee are returned as the set \(PrivS\) using the predicate \(hasRights\). 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. (A more detail description of the predicate \(common\) can be found in Appendix A,
Algorithm 6. If the number of matched privileges is $k$ or more, the head of the rule becomes true and the delegation will be denied.

Visibility conflict policy:

In this delegation model, only the users of the enterprise (or data collecting) visibility are allowed to delegate to the users of other visibilities. The visibility conflict policy is used by the administrator of the enterprise visibility to block the transfer of any access rights to a particular visibility. For example, an organization does not want its employees to delegate any privilege to the users of the visibility thirdparty. To do that, information about the prohibited visibility should be entered into the system first. A rule is then defined to raise an error if the delegatee of a delegation request is from the prohibited visibility. Formally,

$$conflictingVis(enterprise, thirdparty) \leftarrow T$$

$$error(u, v, odr) \leftarrow vis(u, uVis) \land vis(v, vVis) \land conflictingVis(uVis, vVis)$$

The first rule configures the system with the information that thirdparty is a prohibited visibility. The head of the second rule becomes true if the delegatee is from the prohibited visibility.

4.3.3 Duplicate delegation policy

This policy prevents a user from receiving access rights they already have. The policy is defined with the following rule:

$$error(u, v, odr) \leftarrow hasRights(v, PrivS)$$

$$\land \exists dp \in PrivS.compliance(dp, odr)$$
The rule checks if the delegatee has a privilege that complies with the delegation request. The fact that one of the delegatee's privileges complies with the delegation request indicates that the delegatee is already assigned the requested access right and delegation is not necessary.

4.4 Delegation process

The delegation agent (Figure 4.2) processes delegation requests using two algorithms: delegation decision and delegation assignment algorithms. The delegation decision algorithm validates the delegation request against a set of rules stored in module DRP. To verify the rules, the agent uses the authorization records stored in module AR and the privacy policies stored in module PP. Finally, the process of assigning delegated access rights to a delegatee is described by the delegation assignment algorithm which updates the authorization records and the delegation histories (module DRH).

The rules stored in module DRP include the delegation policies and constraints discussed in the previous section (Section 4.3). In the decision algorithm, intra-visibility delegation policy is denoted by the predicate canGiveSV while the predicates canGiveDVC and canGiveDVE represent the policies for the inter-visibility delegation of type collaboration and exchange, respectively. All the constraints which consists of the security and duplicate delegation policies are stored in a set denoted by SCB. The dot operator is used to denote a particular predicate of a constraint. As an example, scb.error denotes the predicate error of the constraint scb where scb ∈ SCB.
Description of the delegation decision algorithm:

Inputs $u$ and $v$ represent the delegator and the delegatee. $A$ is a set of delegation requests where each element of $A$ is a delegation object. Another input of the decision algorithm is a time $etm \in Z$ which is the valid period for using the delegated access rights. A delegatee loses the rights once the time expires. $A'$ is an empty set that will be used to store the processed requests.

**Algorithm 1: Delegation decision**

<table>
<thead>
<tr>
<th>Line</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$i := \text{vis}(u), j := \text{vis}(v)$</td>
</tr>
<tr>
<td>2</td>
<td>$A' := \emptyset, dp := \emptyset$</td>
</tr>
<tr>
<td>3</td>
<td>For each $odr \in A$:</td>
</tr>
<tr>
<td>4</td>
<td>If $(i = j)$ then</td>
</tr>
<tr>
<td>5</td>
<td>If $(\neg \text{canGiveSV}(u, v, dp, odr))$ then</td>
</tr>
<tr>
<td>6</td>
<td>return $false$</td>
</tr>
<tr>
<td>7</td>
<td>If $(i \neq j &amp; i = &quot;Enterprise&quot;)$ then</td>
</tr>
<tr>
<td>8</td>
<td>If $(\neg (\text{canGiveDVC}(u, v, dp, odr) | \text{canGiveDVE}(u, v, dp, odr)))$ then</td>
</tr>
<tr>
<td>9</td>
<td>return $false$</td>
</tr>
<tr>
<td>10</td>
<td>For each $scb \in SCB$:</td>
</tr>
<tr>
<td>11</td>
<td>If $scb.\text{error}(u, v, odr)$ then</td>
</tr>
<tr>
<td>12</td>
<td>return $false$</td>
</tr>
<tr>
<td>13</td>
<td>End For each</td>
</tr>
<tr>
<td>14</td>
<td>If $(\text{equality}(dp, odr))$ then</td>
</tr>
<tr>
<td>15</td>
<td>$A' := A' \cup dp$</td>
</tr>
<tr>
<td>16</td>
<td>Else then</td>
</tr>
<tr>
<td>17</td>
<td>$dp' := odr$</td>
</tr>
<tr>
<td>18</td>
<td>$DP_i := DP_i \cup dp'$</td>
</tr>
<tr>
<td>19</td>
<td>$A' := A' \cup dp'$</td>
</tr>
</tbody>
</table>
Based on the delegator’s and the delegatee’s visibilities, each delegation object $odr$ of the set $A$ is tested against either intra- or inter-visibility delegation policies. Successful inference of a delegation policy also returns the delegator’s privilege as $dp$ that complies with the delegation request (line 4-9). The request is then checked against the constraints of the set $SCB$ to see if any of them is violated if the request is granted (line 10-13).

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 request $odr$ is compared with the privilege $dp$. Note that privilege $dp$ is returned by the delegation policy. If it is equal to $odr$, it can be delegated directly, so, it is added to the delegation set $A'$. Otherwise, a new privilege is created based on $odr$ for the delegatee’s visibility and added to the set $A'$ (line 14-19). Privileges in the output set $A'$ are assigned to the delegatee using the delegation assignment algorithm.

**Description of the delegation assignment algorithm:**

This algorithm assigns each privilege of the set $A'$ separately to the delegatee $v$ (line 4). Each assignment is followed by adding an entry to the delegation history $DH$ (line 5). The entry consists of delegator, delegatee, delegated privilege, start and end time for delegation. Start time $stm$ is the current system time obtained using the predicate $CurrentSystemTime$. End time, $etm$, is a part of the delegation request.
Algorithm 2: Delegation assignment

<table>
<thead>
<tr>
<th>Input: ((u, v, A', etm)) Output: None.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 ( j := \text{vis}(v) )</td>
</tr>
<tr>
<td>2 ( \text{stm} := \text{CurrentSystemTime()} )</td>
</tr>
<tr>
<td>3 For each ( dp' \in A' ):</td>
</tr>
<tr>
<td>4 ( TUADP_j := TUADP_j \cup (v, dp', etm) )</td>
</tr>
<tr>
<td>5 ( DH := DH \cup (u, v, dp', etm, \text{stm}) )</td>
</tr>
<tr>
<td>6 End of For each</td>
</tr>
</tbody>
</table>

4.5 Revocation

Revocation is the process of removing delegated access rights from a delegatee. In this thesis, two methods of revocation are considered. In the first method, delegators revoke any access right that they delegated in the past. This type of revocation is called *forced revocation*. In the second method, a delegated access right is revoked automatically when its valid time expires. This type of revocation is called *auto revocation*. Like delegation, revocation is also performed by the delegation agent of Figure 4.2. The revocation policies used by the agent are formally described in the following.

**Forced revocation policy:**

\[
\text{canRevoke}(u, v, dp) \leftarrow \text{hasDelegation}(v, TPD) \land \exists tpd \in TPD
\]

\[
\land \text{object}(tpd, dp) \land \text{grantor}(tpd, u)
\]
Consider that a revocation request consist of a delegator $u$, a delegatee $v$ and a delegated privilege $dp$. It means that user $u$ wants to revoke the delegated object $dp$ from user $v$. To validate the request, the delegation agent uses the above rule where the predicate $\text{hasDelegation}$ returns all the delegation records as $TPD$ where the user $v$ was the delegatee. Of these delegation records, the rule finds a delegation event using the predicate $\text{grantor}$ where the user $u$ was the delegator. If there is no such event, the rule returns false and the revocation request is denied. The successful inference of the rule returns the privilege $dp$ which would be revoked later.

**Auto revocation policy:**

\[
\text{autoRevoke}(SYS, v, dp) \leftarrow \text{hasDelegation}(v, TPD) \land \exists tpd \in TPD
\]

\[
\land \text{object}(tpd, dp) \land \text{time}(tpd, lt) \land \text{expired}(lt)
\]

This type of revocation does not require any request for initiation. The delegation agent periodically goes through the delegated access rights of the users and revokes the expired rights. The agent follows the above rule when it performs revocation. In the rule, the predicate $\text{hasDelegation}$ returns all the delegation records as $TPD$ for a user $v$. Of these records, the rule finds the delegation events whose valid time is expired by the system clock using the predicates $\text{time}$ and $\text{expired}$.

Successful inference of either of the revocation policies will initiate the following cleanup task that would finally remove the delegated object from the delegatee.

\[
TUADP_j := TUADP_j/v, dp,.
\]
The delegation agent will update the authorization records by removing the access right \( dp \) from the user \( v \). Consider the visibility of the user \( v \) as \( j \). Dot is used on the position of valid time in the entry \((v, dp, .)\) to indicate that the cleanup task is not affected by its valid time. Any record in the relation \( TUADP_j \) containing \( v \) and \( dp \) will be removed. Once the record is removed, the revocation event is logged in the revocation histories (module DRH in Figure 4.2). The log record is stored using a relation \( RVH \subseteq \{U \cup \{"sys"\}\} \times U \times DP \times Z \). An entity drawn from this set \(<u, v, o, tm>\) denotes that a privilege \( o \) was revoked from a user \( v \) at the system time \( tm \) on the request from a user \( u \). In case of auto revocation, \( u = "sys" \).

In this delegation model, a delegator can create a new privilege or use an existing one for delegation. Therefore at the time of revocation, it is necessary to check if the delegated privilege is still being used by other users. If it is not, then the privilege was created only for delegation and after revocation, it should be deleted from the system. Future extension of this model will examine this issue in detail.
Chapter 5

Case study

An online shopping website *EShop* is imagined as the data collector in this case study. The website sells products online. It collects data about the customers who visit the site or purchase any product from it. The website uses the data for its business operation. To complete the business operation, it sometimes relies on external organizations. Therefore, it gives those organizations access to its customers’ data. This case study will show how delegation is used to share access rights within an organization and between organizations.

The following schemas show what data the website *EShop* collects about its customers.

\[ DInfo \ (\text{CustomerID}, \text{Name}, \text{DofB}, \text{Home address}, \text{Email address}, \text{Phone}) \]

\[ SystemInfo(\text{CustomerID}, \text{IP address}, \text{Cookie information}) \]

\[ CProfile(\text{CustomerID}, \text{PurchasedItem}, \text{ItemType}, \text{SystemTime}) \]

Customers’ demographic information is stored in the schema *DInfo*. Information about the purchased products are stored in the schema *CProfile*. The schema *SystemInfo* stores the customers’ IP addresses and cookie [HTT] information. In the privacy policy and privileges, data is mentioned using the format `Schema.Attribute`. For example, *DInfo.Email address* represents the attribute *Email address* of the schema *DInfo*. The format `Schema.*` denotes all the attributes of a schema.
*E*Shop uses the collected information for different business purposes. Consider that purposes fulfilled by *E*Shop are limited to the set given in Figure 3.1.

To fulfill some of these purposes, *E*Shop uses external organizations as service providers, business affiliates, etc. *E*Shop may have one or more organizations listed as each of these categories. For example, *E*Shop may hire one company, say *C*1, for shipping the products to the customers and another company, say *C*2, for surveying the customers about the services they get from *E*Shop. Both companies are its service providers. Such relationship between visibilities can be captured by introducing a partial ordering among the visibilities.

From the previous example, if three visibilities are considered: *service provider*, *C*1 and *C*2, then both *C*1 and *C*2 are ordered with respect to *service provider*.

The delegation operations described in Chapter 4 are orthogonal to the partial relations of visibilities. The relationship between visibilities was not defined at that time for simplicity. However, to better explain each step of the case study, the partial ordering among visibilities is considered.

The partial order \( \leq_{vis} \) is a reflexive, transitive and anti-symmetric relation defined over the set *VIS*. (The definition of the relation is same as the ones defined for purposes in Section 3.2 and roles in Section 3.3.) An order between two visibilities naturally introduces a relation between the privacy policies of these visibilities. Consider two visibilities *vis*\( A \) and *vis*\( B \) having the relation (*vis*\( A \leq_{vis} \) *vis*\( B \)), then \( PPolicy_{visA} \subseteq PPolicy_{visB} \). According to this, the privacy policies for either of the companies *C*1 or *C*2 is a subset of the policies for the visibility *service provider* in the previous example.
In other words, the privacy policies of a visibility \( \text{vis}B \) is a superset of the privacy policies of all the visibilities \( b \) that satisfy \( b \preceq_{\text{vis}} \text{vis}B \). The relation is described formally in the following:

\[
\text{vis}B \in \text{VIS}
\]
\[
V = \{b | b \in \text{VIS} \land b \preceq_{\text{vis}} \text{vis}B\}
\]
\[
\bigcup_{b \in V} PPolicy_b \subseteq PPolicy_{\text{vis}B}
\]

Delegation policies and constraints considered in this case study are described as follows. These policies are explained in Chapter 4. Note that visibility conflict policy and duplicate delegation policy are not used in this case study.

Policy I: Delegation policy for intra-visibility delegation

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

Policy II: Delegation policy for inter-visibility delegation of type exchange

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

Policy III: Security policy

\[
\text{error}(u, v, odr) \leftarrow \text{vis}(v, j) \land aUserCannotHave(j, MDP, k)
\]
\[
\land \text{hasRights}(v, \text{PrivS}) \land \text{combine}(\text{PrivS}, odr, \text{PrivS}')
\]
\[
\land \text{common}(\text{PrivS}', \text{MDP}, k) \land k' \geq k
\]
Case#1: Intra-visibility delegation

Scenario: *EShop* plans to run a promotional contest among the customers and eventually, reward some of the participating customers. The task of running the contest can be divided into two subtasks: marketing about the contest and selection of the winner. *EShop* creates privacy aware privileges that allow accessing customer records for these tasks and assigns these privileges to its employees. One of the employees involved with the contest is going for a vacation. Before leaving, he attempts to delegate his authorization to another employee.

The website *EShop* has a privacy agreement with data providers about how their information would be used by *EShop* itself *i.e.*, by the employees of *EShop*. The privacy agreement is denoted by *PPolicy_{enterprise}* as *EShop* is the enterprise visibility in this data collection. Consider that Table 5.1 lists the policies of the set *PPolicy_{enterprise}.*

<table>
<thead>
<tr>
<th>PolicyID</th>
<th>Data</th>
<th>Action</th>
<th>Purpose</th>
<th>Condition</th>
<th>Obligation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>DInfo.ID</td>
<td>read</td>
<td>Billing</td>
<td>∅</td>
<td>∅</td>
</tr>
<tr>
<td>2</td>
<td>DInfo.Name</td>
<td>read</td>
<td>Billing</td>
<td>∅</td>
<td>∅</td>
</tr>
<tr>
<td>3</td>
<td>DInfo.Email address</td>
<td>read</td>
<td>Billing</td>
<td>∅</td>
<td>∅</td>
</tr>
<tr>
<td>4</td>
<td>CProfile.*</td>
<td>read</td>
<td>Billing</td>
<td>∅</td>
<td>∅</td>
</tr>
<tr>
<td>5</td>
<td>DInfo.Email address</td>
<td>read</td>
<td>Promotion</td>
<td>∅</td>
<td>∅</td>
</tr>
<tr>
<td>6</td>
<td>DInfo.Customer ID</td>
<td>read</td>
<td>Contest</td>
<td>enteredTS</td>
<td></td>
</tr>
</tbody>
</table>

These policies allow *EShop* using data for the purposes *Billing*, *Promotion*, and *Contest*. However, only policies 5 and 6 are relevant in this case. Based on these policies, the administrator of *EShop* creates two privileges.
\[ dpX_1 = < DInfo. Email address, read, Promotion - eMarketing > \]

\[ dpX_2 = < DInfo. CustomerID, read, Contest - Contest > \]

\[ DP_{enterprise} := DP_{enterprise} \cup \{dpX_1, dpX_2\} \]

The first privilege allows a user to promote the contest among the customers by email while the second privilege allows using the IDs of the customers to run a draw among them who participated in the contest. Assume that for security reasons, these two privileges are mutually exclusive; no single user can have them both. The idea behind this is that the user who is responsible for marketing should not be involved in the contest draw. The administrator enters the required information in the system using policy IV. Policy IV states that no user can have both of these privileges \(\{dpX_1, dpX_2\}\).

Policy IV:

\[ aUserCannotHave(\{dpX_1, dpX_2\}, 2) \leftarrow \top \]

Consider that EShop has three employees \(u_1, u_2\) and \(u_3\) and two of them are assigned privileges.

\[ u_1, u_2, u_3 \in U_{enterprise} \]

\[ (u_1, dpX_1), (u_3, dpX_2) \in UADP_{enterprise} \]

Now user \(u_3\) is going away for vacation, so he wants to delegate \(dpX_2\) to another person until the system time \(etm\) where \(etm = 12/01/2012\ 09:00:00\). He attempts to delegate to user \(u_1\) with a delegation request \(< DInfo. CustomerID, read, Contest - Contest >\) which is equivalent to the privilege \(dpX_2\).
The request is allowed by the intra-visibility delegation policy (Policy I) as the request is compliant to the delegator’s privilege $dpX2$. However, the security policy (Policy III) raises an error. The security policy uses the information entered using Policy IV which states that no user can have both privileges $dpX1$ and $dpX2$. Since the delegatee $u1$ is already assigned the privilege $dpX1$, he cannot receive $dpX2$. The delegation request is rejected and the delegator and the delegatee are informed that delegation was not successful.

The user $u3$ attempts to delegate to the user $u2$ with the same request. This time, delegation is allowed by policy I with no error raised by the security policy. Since the privilege $dpX2$ is same as the delegation request, $dpX2$ is assigned to the delegatee instead of creating a new privilege. The delegation is also logged by adding an entry in DH.

$$TUADP_{\text{enterprise}} := TUADP_{\text{enterprise}} \cup (u3, dpX2, etm)$$

$$DH := DH \cup (u3, u2, dpX2, stm, etm)$$

Here, the delegation expiry time, $etm$, is 12/01/2012 09:00:00 and suppose, the current system time, $stm$, is 13/01/2010 09:00:00. Finally, the delegator and the delegatee are informed that the delegation was successful.

**Case#2: Inter-visibility delegation**

Scenario: $EShop$ has a subsidiary company $FamilyShopping$ which also sells different products online. $FamilyShopping$ is currently promoting several brands online. $EShop$ has a huge customer base. $FamilyShopping$ plans to use the customers’ data of $EShop$. To make the promotion effective, it plans to send customized advertisements to the customers by
email. To produce customized advertisements, it requires the customers’ purchase history. Therefore, *FamilyShopping* needs access to the customers’ ID, email address, and the type of products they purchased from *EShop*. An employee of *FamilyShopping* requests access to the necessary records from an employee of *EShop*.

It is assumed that *EShop*’s privacy agreement with its customers allows it to share their information with *EShop*’s corporate parents and subsidiary companies. Table 5.2 presents the agreement *PPolicy*_\(_{ps}\) where *ps* is the visibility of *corporate parents and subsidiaries*, containing policies that allow using customers’ IDs, email addresses and the types of the items they purchased from *EShop*. Here, ID is a unique number given to the customers by *EShop* when they register.

Table 5.2: Privacy policy for *EShop*’s subsidiaries and parent organizations, *PPolicy*_\(_{ps}\)

<table>
<thead>
<tr>
<th>PolicyID</th>
<th>Data</th>
<th>Action</th>
<th>Purpose</th>
<th>Condition</th>
<th>Obligation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>DInfo.CustomerID</td>
<td>read</td>
<td>eMarketing</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>2</td>
<td>DInfo.Email address</td>
<td>read</td>
<td>eMarketing</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>3</td>
<td>CProfile.CustomerID</td>
<td>read</td>
<td>eMarketing</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>4</td>
<td>CProfile.ItemType</td>
<td>read</td>
<td>eMarketing</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
</tr>
</tbody>
</table>

Consider that the visibility *FamilyShopping* is denoted by *fs* and has the relationship (*fs* $\prec_{vs}$ *ps*) in the visibility hierarchy. It is also assumed that *FamilyShopping* is given all the privileges that are allowed for the visibility *corporate parents and subsidiaries*, so *PPolicy*_\(_{fs}\) = *PPolicy*_\(_{ps}\). The rest of the case study describes a delegation operation where an employee of *EShop* delegates several access rights to an employee of *FamilyShopping*. 
Consider two users from these visibilities: \( v \in U_{fs} \) and \( u5 \in U_{\text{enterprise}} \). In this case, user \( v \) is the delegatee and the user \( u5 \) is the delegator. Being an employee of EShop, \( u5 \) is responsible for billing the customers who purchase goods. The privileges assigned to \( u5 \) are following which are created based on policies 1-4 in Table 5.1.

\[
(u5, dpY1), (u5, dpY2), (u5, dpY3), (u5, dpY4) \in UADP_{\text{enterprise}}
\]

\[
dpY1: < DInfo.ID, read, Billing – Billing >
\]

\[
dpY2: < DInfo.Name, read, Billing – Billing >
\]

\[
dpY3: < DInfo.Email address, read, Billing – Billing >
\]

\[
dpY4: < CProfile.*, read, Billing – Billing >
\]

The employee of FamilyShopping, \( v \), requests the user \( u5 \) for access to the customers’ IDs, email addresses and their purchase types. In response to the request, the user \( u5 \) submits a set \( A \) of delegation requests. The contents of \( A \) include four access rights that would give access to the requested records.

\[
A = \{odr1: < DInfo.CustomerID, read, eMarketing – eMarketing >, 
odr2: < DInfo.Email address, read, eMarketing – eMarketing >, 
odr3: < CProfile.CustomerID, read, eMarketing – eMarketing >, 
odr4: < CProfile.ItemType, read, eMarketing – eMarketing >\}
\]

The user \( u5 \) has access to all the requested data items. However, the requested purpose is different than the purpose \( u5 \) uses the data for.
The exchange policy (policy II) requires that in order to delegate, the delegator should have access to the requested data and in order for the delegatee to receive access rights, the privacy policy of the delegatee’s visibility should support the delegation request. In this case, for each delegation request of the set $A$, the delegator $u5$ has a privilege containing the requested data and also, there is a privacy policy in $PPolicy_f$ supporting each request.

Assuming that no security policy applies to this case, the delegation requests are validated successfully. Four privileges created based on the delegation requests of the set $A$.

$$dpZ_1 := odr1, \ dpZ_2 := odr2, \ dpZ_3 := odr3, \ dpZ_4 := odr4$$

$$dpZ := \{dpZ_1, dpZ_2, dpZ_3, dpZ_4\}$$

$$DP_f := DP_f \cup dpZ$$

Privileges are then assigned to the delegatee $v$. The delegation event is also logged.

Suppose the requested period for using the delegated privileges $etm$ is 12/01/2014 09:00:00 and the current system time $stm$ is 14/01/2010 09:00:00. For each $dp \in dpZ$, the following entries are entered into the system:

$$TUADP_f := TUADP_f \cup (v, dp, etm)$$

$$DH := DH \cup (u5, v, dp, stm, etm)$$

The above example shows how the inter-visibility delegation of type exchange is performed in the proposed model.
Chapter 6

Complexity analysis

This section gives the complexity analysis of the proposed delegation model. First, we analyze the complexity of individual policies and then compute the complexity of the entire delegation process. In this study, we assume that comparison is more expensive than all other operations (e.g., assignment). Table 6.1 contains the semantics of the notations used here.

Table 6.1: Notations used in the complexity study

<table>
<thead>
<tr>
<th>Notations</th>
<th>Meaning</th>
<th>Notations</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>$N$</td>
<td>Total number of users in the system</td>
<td>$nP$</td>
<td>Number of purposes fulfilled by a data collector</td>
</tr>
<tr>
<td>$nU$</td>
<td>Number of users in a visibility</td>
<td>$nPP$</td>
<td>The number of privacy policies for a visibility</td>
</tr>
<tr>
<td>$nR$</td>
<td>Number of roles in a visibility</td>
<td>$nSoD$</td>
<td>Number of the separation of duty policies exist in a visibility</td>
</tr>
<tr>
<td>$nDP$</td>
<td>Number of privileges in a visibility</td>
<td>$nVCPOL$</td>
<td>Number of the visibility conflict policies exist in the system.</td>
</tr>
</tbody>
</table>

6.1 Complexity of the delegation policies and constraints

Intra-visibility delegation:

\[ canGiveSV(u, v, dp, odr) \]

\[ \leftarrow hasRights(u, PrivS) \land \exists dp \in PrivS.compliance(dp, odr) \]
In this rule, the predicate hasRights returns all the privileges that are assigned to a user $u$ as the set $\text{PrivS}$. (The pseudo code for this predicate is given in Appendix A, Algorithm 7.) The number of comparisons needed to run the predicate is $nR^2$ where $nR$ is the number of roles in a visibility. An elaboration of this estimate is as follows.

**hasRights:** The predicate first retrieves the roles assigned to the user $u$. A structure array, UserRole, is used to store the user-role assignments. The array is indexed by another structure array, UserRoleIndex, that contains the starting and ending record references for $u$ in the array UserRole. In the worst case, the entire index array needs to be searched for finding the reference of the user $u$ which would incur the cost $nU$. Once the reference is found, we assume that the cost of building the set $Ru$ which records all the roles assigned to the user $u$ is negligible. Thus, the cost for retrieving the set of roles assigned to a user is $nU$.

When role hierarchy is used, a role may subsume other roles and is allowed to have privileges of the subsumed roles. So, we find out what other roles are subsumed by the roles of the set $Ru$.

Role hierarchy is stored using a structure array, RoleHierarchy. The structure contains two members which are roles. The first role subsumes the second role. The array is indexed by another structure array, RoleHierarchyIndex. Using the index, the complexity of finding the subsumed roles for all the roles of the set $Ru$ is $nR^2$. All the subsumed roles are added to the rear end of the set $Ru$.

We next find the privileges assigned to the roles of the set $Ru$. We use the similar storage techniques for role-privilege assignments as we did for user-role assignments. Therefore,
the complexity to find all the privileges is $nR^2$. The total complexity for the predicate hasRights is $O(nU + nR^2 + nR^2)$ which is equivalent to $O(nR^2)$. □

The user $u$ can delegate if one of the retrieved privileges is compliant to the delegation request. In the worst case, we need to run the compliance predicate for all the privileges of the user $u$. If we assume that the user can be assigned all the privileges of its visibility, we need to run the compliance predicate $nDP$ times. In the compliance predicate, we do three comparisons with data, action and purpose. With the exception for purpose, there is no hierarchy for data and action, so the data and action comparisons incur negligible cost. However, the cost of comparing purposes is $nP$ which is the number of purposes in the system. (The routine to compare purposes is given in Appendix A, Algorithm 3.) Therefore the cost of finding a privilege that complies with the delegation request is $(nDP \cdot nP)$.

Then, the total complexity of validating the canGiveSV policy is $O(nR^2 + (nDP \cdot nP))$. If we assume that $(nDP >> nR), (nP > nR)$, and so $((nDP \cdot nP) > nR^2)$, then the complexity is $O(nDP \cdot nP)$.

Inter-visibility delegation: Collaboration

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

The cost of the first part of the rule that checks if a delegator has a privilege containing the delegation object is $O(nR^2 + nDP \cdot nP)$ as explained for the intra-visibility delegation policy.
The second part of the rule checks if the privacy policy of the destination visibility supports the delegation request using the predicates \( \text{vis} \) and \( \text{dlookup} \). The predicate \( \text{vis} \) returns the visibility of the delegatee \( v \). An array can be used to store the visibilities of all the users where one column stores the users and another column stores their visibilities. In such a setting, we may need to search the entire array in the worst case to find a user as well as its visibility. Thus, the cost of the predicate \( \text{vis} \) is \( N \) where \( N \) is the total number of users in the system. Once the delegatee’s visibility is retrieved, the predicate \( \text{dlookup} \) is used to find a privacy policy that supports the delegation request. The predicate \( \text{dlookup} \) returns true if the following expression holds.

\[
\exists pp \in PPolicy_j \land \text{odr}.d = pp.d
\]

\[
\land \text{odr}.a = pp.a \land pp.p \leq \text{pr}_{\text{odr}}.p_b
\]

We consider the cost for comparing purposes and ignore the cost for comparing data and action. In the worst case, we need to traverse through all the privacy policies for a visibility and for each policy, purposes are compared. Thus, the complexity of the predicate \( \text{dlookup} \) is \( O(nPP \times nP) \).

By summing up all the costs of the previous steps, the total complexity of the \( \text{canGiveDVC} \) policy is \( O(nR^2 + (nDP \times nP) + N + (nPP \times nP)) \) which is equivalent to \( O(nDP \times nP) \) under the assumptions that \( (nDP > nPP) \), and \( ((nDP \times nP) > (nPP \times nP)) \).

**Inter-visibility delegation: Exchange**

\[
\text{canGiveDVE}(u, v, dp, odr) \leftarrow \text{hasRights}(u, PrivS)
\]

\[
\land \exists dp \in PrivS. (dp.d = odr.d) \land \text{vis}(v, j) \land \text{dlookup}(j, odr)
\]
The first of the rule checks if the delegator has a privilege that contains the requested data. As explained in the previous section, the number of comparisons needed to run the predicate \textit{hasRights} is \((nR^2)\). The user \(u\) can delegate if one of its privileges contains the requested data \(i.e., (dp.d = odr.d)\) is true. The cost of this comparison is 1. The worst case requires this comparison for all the privileges of the user \(u\). If we assume that the user can be assigned all the privileges of the visibility, the number of comparisons for this step is \(nDP\) and the total cost up to this step is \(O(nR^2 + nDP)\).

Similar to the collaboration policy, the second part of the rule checks if the privacy policy of the destination visibility supports the delegation request using the predicates \textit{vis} and \textit{dlookup}. Therefore the cost of this check is \(O(N + (nPP * nP))\)

By summing up all the costs of the previous steps, the total complexity of the \textit{canGiveDVE} policy is \(O(nR^2 + nDP + N + (nPP * nP))\) which is equivalent to \(O(nPP * nP)\) under the assumptions that \((nPP > nR), (nP > nR)\) and \(((nPP * nP) > nR^2)\).

\textbf{Security constraints: Separation of duty policy}

\[error(u,v,odr) \leftarrow vis(v,j) \land aUserCannotHave(j,MDP,k)\]

\[\land hasRights(v,PrivS) \land combine(PrivS,odr,PrivS')\]

\[\land common(PrivS',MDP,k) \land k' \geq k\]

Prior to delegation, a set of separation of duty policies for each visibility are entered into the system. The above rule uses those policies to validate a delegation request. In the rule, the predicate \textit{aUserCannotHave} returns one SoD policy at a time and all the remaining
elements are tested for that policy. The cost of the predicate \textit{hasRights} is \(nR^2\). The cost of predicate \textit{combine} is negligible and ignored here. The cost of the predicate \textit{common} is \((nDP^2*nP)\). (The pseudo code of the predicate is given in Appendix A, Algorithm 6.) The complexity analysis of the predicate is explained as follows.

\textit{common}: The predicate finds the common elements between two privilege sets. Two loops are used to traverse and compare the elements of these sets. There are two If conditions nested inside the loops. The total cost of these conditions is \(nP\) as one of these compare purposes. Each loop will iterate \(nDP\) times in the worst case where both of the input privilege sets have the size of \(nDP\), so the total cost of the predicate is \((nDP^2*nP)\).

The total cost of validating one security policy consists of the costs of the predicates, \textit{hasRights}, and \textit{common}, which is \(O(nR^2 + (nDP^2*nP))\). It is equivalent to \(O(nDP^2*nP)\). Here, we ignore the cost of the predicate \textit{combine}. The estimated cost is the number of comparisons for testing one separation of duty policy. If the number of policies entered into the system is \(nSoD\), the complexity becomes \(O(nSoD * nDP^2*nP)\).

\textbf{Security constraint: Visibility conflict policy}

\[\text{error}(u, v, odr) \leftarrow \text{vis}(u, uVis) \land \text{vis}(v, vVis) \land \text{conflictingVis}(uVis, vVis)\]

This policy checks if the delegation is prohibited between the visibilities of the delegator and the delegatee. The total cost of the two \textit{vis} predicates is \((2*N)\). Once the visibilities of the delegator and delegatee are retrieved, the rule goes through all the visibility conflict policies that exist in the system and attempts to match the visibilities. Therefore, the total
cost of the policy is $O(2 \times N + nVCPOL)$ which is equivalent to $O(N)$ under the assumption that $(N \gg nVCPOL)$.

**Duplicate delegation policy:**

\[
\text{error}(u, v, odr) \leftarrow \text{hasRights}(v, PrivS)
\]

\[
\land \exists dp \in PrivS. \text{compliance}(dp, odr)
\]

The rule first returns the privileges assigned to the delegatee $v$ using the predicate $\text{hasRights}$ with the cost $nR^2$. If one of the returned privilege complies with the delegation request, the head of the rule returns true. In the worst case, the predicate $\text{compliance}$ is tested for all privileges which would incur the cost $O(nDP \times nP)$. The total cost of the policy is $(nR^2 + (nDP \times nP))$ which is equivalent to $O(nDP \times nP)$.

**6.2 Total complexity of the delegation decision algorithm**

Delegation requests are processed by the delegation decision algorithm. (The algorithm is given in Section 4.4.) The algorithm first tests the request against all the delegation policies and constraints. If the request passes all these checks, the algorithm uses the predicate $\text{equality}$ to verify if the request is same as the privilege returned by the delegation policy. We consider that the cost of the predicate $\text{equality}$ is negligible.

To validate an intra-visibility delegation request, the decision algorithm uses the intra-visibility delegation policy, the separation of duty policy, the visibility conflict policy and the duplicate delegation policy. So, the complexity of the algorithm is the addition of the individual complexity of these policies.
The complexity of validating an intra-visibility delegation request

\[ = O((nDP \times nP) + (nSoD \times nDP^2 \times nP) + N + (nDP \times nP))\]

\[ \approx O(nSoD \times nDP^2 \times nP)\]

The inter-visibility delegation request should be validated against the collaboration or the exchange policy. The decision algorithm first tests the collaboration policy. If the request is not valid by that policy, the algorithm then tests the exchange policy. In the worst case, both policies will be tested for a single inter-visibility delegation request.

(Note that the operators | and || work in two different manners in C language. Though the symbol | is used throughout the thesis, it is assumed that the operator | will not evaluate the remaining operands of a compound expression when it finds one operand as true)

So, the complexity of validating an inter-visibility delegation request is the addition of the complexities of the collaboration policy, the exchange policy, the separation of duty policy, the visibility conflict policy and the duplicate delegation policy.

The complexity of validating a delegation request

\[ = O((nDP \times nP) + (nP \times nP) + (nSoD \times nDP^2 \times nP) + N + (nDP \times nP))\]

\[ \approx O(nSoD \times nDP^2 \times nP)\]

This estimate is the cost of validating a single delegation request. When there are multiple requests, the cost is multiplied by the number of the requests. From this analysis, we find that the separation of duty policy enforcement is the most expensive of all the policies in both intra- and inter-visibility delegations.
We do not measure the complexity of the delegation assignment algorithm as it does only a few assignment operations whose cost is negligible. However, we plan to investigate the performance of the revocation process of our model in future.
Chapter 7

Conclusion

The number of Internet users has grown very fast over the last decade. The success of Internet can be attributed to the huge number of websites and a wide range of services they offer. While most of these services are free, users disclose their personal information by using them. Some information are required by a website to provide the service, some are collected automatically by monitoring the users’ activities and some are provided voluntarily by a user. Considering the amount and varieties of personal information shared, internet users have a higher risk of privacy breach. Inappropriate of use private information can harm our 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 [DHF09].

7.1 Contribution

Data can be accessed by multiple parties and a data provider may have different privacy preferences about how the data should be used by each party. An effective way to protect a data provider’s privacy is to control the use of their data through privacy preferences.

As a contribution of this thesis, a privacy model is proposed to formalize data providers’ preferences into privacy policies for each party who are interested to access data. An access
control model is adapted which uses the privacy policies to create access rights containing appropriate privacy restrictions for each party.

Delegation is a process of sharing access rights in access control models. It brings flexibility into the access control models. This thesis also introduces a delegation model that allows sharing access rights in the proposed access control setting. By applying two delegation policies, it is ensured that a delegated access right is constrained by the appropriate privacy policy for the receivers. Several security and efficiency constraints are applied to the delegation model. All these policies and constraints are expressed using a declarative logic language.

7.2 Future works

We would like to investigate the following issues as future works.

Duty delegation: To complete a task, users need a set of privileges rather than a single privilege. In the proposed delegation model, when users need a set of privileges, privileges are assigned individually. In a bid to improve the management of the delegated privileges, we plan to group the requested privileges and then assign the group to the delegatee. In the new setting, revocation will require only step which will remove the users from the delegated privilege group.

We would like to test the assumption that a single task requires privileges with the same purpose. If it is true, privileges with the same the purpose will be put into one group and group will be termed as duty. A privilege group created for one delegation request can also be reused in future delegation events.
Flexibility in inter-visibility delegation policy: In our current proposal, the policy for inter-visibility delegation is more flexible than the one for intra-visibility delegation. In inter-visibility delegation, delegators create access rights from the privacy policies of the destination visibility and the rights are then assigned to the delegatee.

In intra-visibility delegation, delegators delegate access rights that are assigned to them. These access rights are created by a system administrator. The administrator can create equal or more restrictive access rights than the privacy policy of the data. So there is an implicit administrative control that overrides the privacy policy in intra-visibility delegation. We would like to study this difference in more detail in future.

Multi-step delegation: Another limitation of the proposed model is that in inter-visibility delegation, the source visibility should always be the data collecting organization. That is, once an external organization gets access to data, they cannot share it with other external organizations. One of the future works would be extending the model so that it allows an external organization to share the access rights with other parties. At the same time, data collecting organization would still have a way to control the sharing.

In intra-visibility delegation, only single step delegation is studied. It means that a delegatee cannot share the access rights any further. In future, we plan to investigate multi-step delegation where delegatees can in turn become delegators and share the rights they received with other users. Multi-step delegation poses challenges for revocation as revoking a privilege from a user can start a series of revocations from the delegates who received the privilege from the first user.
Delegation in health care: Delegation is highly practiced in health care sectors where accesses to patients’ data are exchanged among the professionals of different health organizations. The privacy concept is slightly different in health care sectors than the one used in our work. For example, the relationship between a data provider and a data user is an important part of the privacy. Consider a sample policy that says that patients’ data can be accessed only by their primary physicians. We plan to extend our delegation model for health care sectors as a future work.
Appendix A

Algorithm 3: Operator, ≤

<table>
<thead>
<tr>
<th>Input: ((P_k, P_{k+1}))</th>
<th>Output: Boolean value</th>
</tr>
</thead>
</table>

//This is comment

//Partial order of purposes is stored in a global array PurposeHierarchy[]

//Index for the above array is stored in a global array PurposeHierarchyIndex[]

//nP is the number of purposes in the set P

For \(t = 0\) to \(nP\):

If PurposeHierarchyIndex \([t][0] = P_{k+1}\) then

\[ SIndexofp = PurposeHierarchyIndex \[t][1] \]

\[ EIndexofp = PurposeHierarchyIndex \[t][2] \]

Break;

For \(l = SIndexofp\) to \(EIndexofp\):

If PurposeHierarchy\([l][1] = P_k\) then

Return true;

Return false

This routine returns true if \((P_k \leq P_{k+1})\). To understand how the routine works, we first need to know how the purpose hierarchy is stored. The purpose hierarchy or the partial ordering of the purposes can be presented as a tree like in Figure A.1.
We store the partial orders into a 2D array. As an example, the array storing the purposes of Figure A.1 would look like Table A.1.

Table A.1: Partial orders among purposes of Figure A.1

<table>
<thead>
<tr>
<th>Left side</th>
<th>Right side</th>
</tr>
</thead>
<tbody>
<tr>
<td>P4</td>
<td>P1</td>
</tr>
<tr>
<td>P4</td>
<td>P2</td>
</tr>
<tr>
<td>P4</td>
<td>P4</td>
</tr>
<tr>
<td>P2</td>
<td>P1</td>
</tr>
<tr>
<td>P2</td>
<td>P2</td>
</tr>
<tr>
<td>P3</td>
<td>P1</td>
</tr>
<tr>
<td>P3</td>
<td>P2</td>
</tr>
<tr>
<td>P1</td>
<td>P1</td>
</tr>
</tbody>
</table>

This routine returns true if it finds an entry in the array that has $P_{k+1}$ as the left side and $P_k$ as the right side. To facilitate the searching, an index structure is used that stores the references of each purpose in the hierarchy array. The index structure is itself another array. Table A.2 is a sample index of Table A.1.

Table A.2: Index for the purpose hierarchy of Table A.1

<table>
<thead>
<tr>
<th>Purposes</th>
<th>Start Index</th>
<th>End Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>P2</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>P3</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>P4</td>
<td>1</td>
<td>3</td>
</tr>
</tbody>
</table>
The routine first retrieves the index for $P_{k+1}$ from the array $PurposeHierarchyIndex$ (which is Table A.2 in the above example). The routine then searches for the entry including $P_k$ in the array $PurposeHierarchy$ (which is Table A.1 in the above example) between the references. It returns true once it finds the entry. The index array can have $nP$ entries where $nP$ is the number of purposes.

**Algorithm 4: Operator, $<$**

Input: $(P_k, P_{k+1})$  
Output: Boolean value.

\[
\text{Return } ( (P_k \leq P_{k+1}) \& (P_{k+1} \neq P_k) )
\]

This routine returns true in the case when $P_{k+1}$ subsumes $P_k$ and $P_{k+1}$ is not same as $P_k$.

**Algorithm 5: Operator, $\circ$**

Input: $(P_k, P_{k+1})$  
Output: Boolean value.

\[
\text{Return } \neg ( (P_k \leq P_{k+1}) | (P_{k+1} \leq P_k) )
\]

This routine returns true in the case when there is no relation between $P_{k+1}$ and $P_k$. 
Algorithm 6: predicate Common

Input: \( (ST1, ST2, k) \)  
Output: Boolean value.

\[
k := 0 \\
n := countElement(ST1), m := countElement(ST2) \\
shadowST2 := \{0,0,...,0\}_m \\
For \( l = 1 \) to \( n \): \\
dp1 = ST1[\( l \)], pr1 = ST1[\( l \)].pr \\
For \( q = 1 \) to \( m \): \\
dp2 = ST2[\( q \)], pr2 = ST2[\( q \)].pr \\
If \( (dp1.d = dp2.d \& dp1.a = dp2.a \& \neg (pr2.U < pr1.L \mid pr1.U < pr2.L \mid pr1.L \circ pr2.L \mid (pr1.L < pr2.L \& pr1.U \circ pr2.U) \mid (pr2.L < pr1.L \& pr2.U \circ pr1.U)) ) \) then \\
    If \( (shadowST2[q] \neq 1) \) then \\
        \( k = k + 1 \) \\
        \( shadowST2[q] = 1 \) \\
End of For \\
End of For \\
Return true
\]

The predicate finds the number of privileges in the set \( ST1 \) that have a match with one of the privileges in the set \( ST2 \). A privilege in the set \( ST1 \) is said to have a match if there
exists a privilege in the set $ST2$ such that both privileges have same data and action and finally, the intersection of their purpose ranges are not null.

No privilege of the set $ST2$ is counted twice. An array $shadowST2$ is used to mark the already counted privileges which are not considered in the next iterations.

Two For loops are used for iterations. Inside the loops, there are two If conditions. The first condition checks if two privileges from $ST1$ and $ST2$ have the same data and action. Then it checks if the purpose ranges of these privileges have any purpose in common using a negated Boolean expression. The Boolean expression is true if the ranges do not have any common element.

The control moves to the next If condition provided that the first condition is true. The new condition examines if the privilege of $ST2$ is already matched to a privilege of $ST1$. In such case, we do not count the privilege. Otherwise, we count it by incrementing $k$ and update the shadow array that the privilege just got counted. Thus this privilege will not be counted in future iterations. When the iterations stop, the value of $k$ gives the number of privileges of the set $ST1$ that matched with the ones of the set $ST2$.

**Algorithm 7: Predicate hasRights**

```
Input:{u, PrivS} Output: Boolean value

/* Visibilities are assigned integer values using enumerators

enum VIS{enterprise,parentSubsidiaries,serviceProviders,...} */

/* NU, NR and NDP are the maximum number of users, roles and privileges for all the
visibilities */
```
/* User-role assignments are stored using the following structure. */

struct SUR{char user[20]; char role[20]; UserRole[nVIS][NU * NR];}

It is assumed that the variable UserRole is already instantiated for all the visibilities */

/* Index for UserRole is stored using the following structure */

struct SURIndex{char user[20]; int s; int e;} UserRoleIndex[nVIS][NU];

It is assumed that UserRoleIndex is already instantiated for all the visibilities */

j := vis(u)

// nU, nR and nDP are the maximum number of users, roles and privileges for visibility j

nU = numU(j)
nR = numR(j)
nDP = numDP(j)

// empty set Ru of size nR and empty set PrivS of size nDP

Ru:=[null, null, ..., null]_nR

PrivS:=[null, null, ..., null]_nDP

// Finding the roles assigned to user u and store them in Ru

For t = 0 to nU:

    If UserRoleIndexes[j][t].user = u then

        SIndexofu = UserRoleIndexes[j][t].s

        EIndexofu = UserRoleIndexes[j][t].e

        Break

End of For

nRu = 0

For t = SIndexofu to EIndexofu:
\[ Ru[nRu] = UserRole[j][t].role \]

\[ nRu = nRu + 1 \]

End of For

\[ nRu = nRu - 1 \]

//Finding the roles in the role hierarchy that are subsumed by the roles in \( Ru \)

/*Role hierarchy is stored using the following structure.

\[
\text{struct SRH} = \{ \text{char role1[20]; char role2[20]; RoleHierarchy[nVIS][nR * nR];} \}/n\]

/*Index of RoleHierarchy is stored using the following structure.

\[
\text{struct SRHI} = \{ \text{char role[20]; int s; int e; RoleHierarchyIndex[nVIS][nR];}\}/n\]

For \( s = 0 \) to \( nRu \):

For \( t = 0 \) to \( nR \):

If \( \text{RoleHierarchyIndex[j][t].role = Ru[s]} \) then

\[ SIndexofr = \text{RoleHierarchyIndex[j][t].s} \]

\[ EIndexofr = \text{RoleHierarchyIndex[j][t].e} \]

Break

End of For

For \( l = SIndexofr \) to \( EIndexofr \):

\[ nRu = nRu + 1 \]

\[ Ru[nRu] = \text{RoleHierarchy[j][l].role2} \]

End of For

End of For

//Finding the privileges assigned to the roles of \( Ru \)

/* Role-privilege assignments are stored using the following structure

...
struct SRDP { char role[20]; char dp[20]; } RolePriv[nVIS][nR * nDP]; */

/* Index for SRDP is stored using the following structure */

struct SRDPI { char role[20]; int s; int e; } RolePrivIndex[nVIS][nR]; */

nDPu = 0
For s = 0 to nRu:
    For t = 0 to nR:
        If RolePrivIndex[j][t].role = Ru[s] then
            SIndexofr = RolePrivIndex[j][t].s
            EIndexofr = RolePrivIndex[j][t].e
            Break;
        End of For
    For l = SIndexofr to EIndexofr:
        PrivS[nDPu] = RolePriv[j][l].dp
        nDPu = nDPu + 1
    End of For
End of For
Return true

The predicate finds the privileges assigned to the user u. To do that, it first retrieves the roles assigned to the user by searching a 2D structure array called UserRole[nVIS][nU * nR] that contains two members- user and role. Here, nVIS is the number of visibilities, nU and nR are the number of users and roles in each visibility. The array can be visualized as a
1D array for each visibility; each element of the 1D array contains a user and the role assigned to that user.

To facilitate the search, the array is indexed by another structure array called $UserRoleIndex[nVIS][nR]$. For each visibility, the first member of the structure contains roles, the second and third member contain the start and end references of the entries of the roles in the array $UserRole$.

The roles that are assigned to the user $u$ are stored in $Ru$. Next, the predicate finds all other roles of the visibility that are subsumed by the roles of $Ru$ according to the role hierarchy. The subsumed roles are also added in $Ru$. The role hierarchy is stored in a structure array called $RoleHierarchy[nVIS][nR \ast nR]$. For each visibility, the first member of the structure contains roles and the second member contains subsumed roles. To facilitate the search, the array is indexed by another structure array called $RoleHierarchyIndex[nVIS][nR]$.

In the final step, the privileges assigned to the roles of the set $Ru$ are retrieved and stored in the array $PrivS$. It is done by searching the role-privilege assignments that are stored in a structure array called $RolePriv[nVIS][nR \ast nDP]$. The array is also indexed by another structure array called $RolePrivIndex [nVIS][nR]$.
Bibliography


[EBA10] EBay.ca. Privacy policy (as of Sep 20,2010).
DOI=http://pages.ebay.ca/help/policies/privacy-policy.html


