This document is dedicated to the graduate students of the University of Florida.
ACKNOWLEDGMENTS

First, I would like to express my gratitude towards Dr. Stanley Y.W. Su, chair of my supervisory committee, for his continuous guidance, advice, and support throughout the course of my Ph.D. study, and for giving me the opportunity to work in the Database Systems R&D Center. My great appreciation also goes to Dr. Herman Lam, co-chair of my supervisory committee, for constantly providing me with valuable comments and suggestions during the dissertation work. I would like to thank my supervisory committee members--Dr. Abdelsalam Helal, Dr. Joachim Hammer and Dr. Richard Elnicki for their constant help, suggestions and time. Thanks also go to Sharon Grant for making the Database Center such a pleasant place to work.

My wholehearted gratitude goes to my parents and sister, for their unconditional love and continuous encouragement throughout my studies.

Finally, I thank all the colleagues and friends at the Database Systems R&D Center so enjoyable. I wish them all the best in their studies and their future career.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACKNOWLEDGMENTS</td>
<td>iv</td>
</tr>
<tr>
<td>LIST OF FIGURES</td>
<td>vii</td>
</tr>
<tr>
<td>ABSTRACT</td>
<td>viii</td>
</tr>
<tr>
<td>CHAPTER</td>
<td></td>
</tr>
<tr>
<td>1 INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>2 RELATED WORK</td>
<td>7</td>
</tr>
<tr>
<td>2.1 Security Models</td>
<td>7</td>
</tr>
<tr>
<td>2.2 Security Policy Specification Languages</td>
<td>8</td>
</tr>
<tr>
<td>2.3 Distributed Trust Management Systems</td>
<td>9</td>
</tr>
<tr>
<td>2.4 Reputation Management Systems</td>
<td>11</td>
</tr>
<tr>
<td>2.5 E-contract Technologies</td>
<td>12</td>
</tr>
<tr>
<td>2.6 Rule-based Knowledge Management Systems</td>
<td>13</td>
</tr>
<tr>
<td>2.7 Standard efforts on Security in the Web Service Infrastructure</td>
<td>16</td>
</tr>
<tr>
<td>2.8 Miscellaneous Related Work</td>
<td>17</td>
</tr>
<tr>
<td>3 REQUIREMENTS OF TRUST AND SECURITY MANAGEMENT</td>
<td>19</td>
</tr>
<tr>
<td>3.1 Security Threats in Collaborative Computing</td>
<td>19</td>
</tr>
<tr>
<td>3.2 Trust, Trustworthiness, and Trust Management</td>
<td>20</td>
</tr>
<tr>
<td>4 TRUST-BASED SECURITY MODEL (TSM) FOR ACCESS CONTROL</td>
<td>24</td>
</tr>
<tr>
<td>4.1 Definitions and Terms</td>
<td>24</td>
</tr>
<tr>
<td>4.1.1 Subject, Object, and Operation</td>
<td>24</td>
</tr>
<tr>
<td>4.1.2 Roles</td>
<td>25</td>
</tr>
<tr>
<td>4.1.3 Certificates</td>
<td>25</td>
</tr>
<tr>
<td>4.1.4 Memberships</td>
<td>27</td>
</tr>
<tr>
<td>4.1.5 Security Constraints</td>
<td>27</td>
</tr>
<tr>
<td>4.2 Trust-based Security Model (TSM)</td>
<td>28</td>
</tr>
<tr>
<td>4.3 Formalization of TSM</td>
<td>32</td>
</tr>
<tr>
<td>4.4 Trust Agreement Specification</td>
<td>36</td>
</tr>
<tr>
<td>4.4.1 Structure</td>
<td>38</td>
</tr>
</tbody>
</table>
4.4.2 Organizations ........................................................................................................38
4.4.3 Trust Policies ........................................................................................................40

5 A NON-REPUDIATION MESSAGE TRANSFER PROTOCOL ...................................45

5.1 Overview of Non-repudiation ..................................................................................45
5.2 Related Work ..........................................................................................................47
5.3 Non-repudiation Protocol Requirements ................................................................50
5.4 Background ............................................................................................................51
  5.4.1 Public Key Crypto Systems ............................................................................52
  5.4.2 Message Digest .............................................................................................52
  5.4.3 Dual Signature ...............................................................................................53
  5.4.4 Notation .........................................................................................................54
5.5 Secure Message Protocol for E-commerce .............................................................55
5.6 Analysis ..................................................................................................................58

6 ARCHITECTURE AND IMPLEMENTATION TECHNIQUE ........................................62

6.1 Distributed Network Architecture for Trusted Collaborative Computing ..........62
6.2 Overview of the Software Architecture ................................................................65
6.3 Implementation Details ..........................................................................................68
  6.3.1 Trust Agreement Specification Tool ............................................................70
  6.3.2 RBAC Specification Tool ............................................................................75
  6.3.3 Run-time Enforcement Engine .....................................................................76
  6.3.4 Protocol Implementation ..............................................................................82
6.4 Experiment ..............................................................................................................83

7 SUMMARY AND FUTURE WORK ........................................................................85

APPENDIX

TRUST AGREEMENT SPECIFICATION .....................................................................87
AN EXEMPLARY SPECIFICATION OF TRUST AGREEMENT .................................88
LIST OF REFERENCES ...............................................................................................89
BIOGRAPHICAL SKETCH .........................................................................................96
## LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-1</td>
<td>Traditional access control architecture.</td>
<td>2</td>
</tr>
<tr>
<td>1-2</td>
<td>Relationships among research objective.</td>
<td>5</td>
</tr>
<tr>
<td>3-1</td>
<td>Trust relationships in collaborative computing.</td>
<td>22</td>
</tr>
<tr>
<td>4-1</td>
<td>Trust-based security model.</td>
<td>30</td>
</tr>
<tr>
<td>4-2</td>
<td>Comparison with NIST’s RBAC methodology.</td>
<td>33</td>
</tr>
<tr>
<td>5-1</td>
<td>Third Party Authority (TPA)-based protocols.</td>
<td>48</td>
</tr>
<tr>
<td>5-2</td>
<td>Secure message transfer protocol for e-commerce.</td>
<td>56</td>
</tr>
<tr>
<td>6-1</td>
<td>Network architecture of a collaborative information system.</td>
<td>63</td>
</tr>
<tr>
<td>6-2</td>
<td>Trust-based security enforcement.</td>
<td>64</td>
</tr>
<tr>
<td>6-3</td>
<td>Software architecture of a trust-based security server.</td>
<td>66</td>
</tr>
<tr>
<td>6-4</td>
<td>General Web service.</td>
<td>69</td>
</tr>
<tr>
<td>6-5</td>
<td>Trust agreement specification tool.</td>
<td>71</td>
</tr>
<tr>
<td>6-6</td>
<td>Review of a trust agreement specification using the tool.</td>
<td>72</td>
</tr>
<tr>
<td>6-7</td>
<td>Editing screen shot of the trust agreement specification tool.</td>
<td>74</td>
</tr>
<tr>
<td>6-8</td>
<td>Editing a role authorization policy using the specification GUI.</td>
<td>74</td>
</tr>
<tr>
<td>6-9</td>
<td>Role based access control specification GUI.</td>
<td>76</td>
</tr>
<tr>
<td>6-10</td>
<td>The Enforcement-time architecture of trusted collaboration.</td>
<td>77</td>
</tr>
<tr>
<td>6-11</td>
<td>Implementation of three protocol components.</td>
<td>83</td>
</tr>
<tr>
<td>6-12</td>
<td>Typical use of trust agreement specification tool.</td>
<td>84</td>
</tr>
</tbody>
</table>
Security and privacy issues have long been investigated in the context of a single organization exercising control over its users’ access to resources. In such a computing environment, security policies are defined and managed statically within the boundary of an organization and are typically centrally controlled. However, developing large-scale Internet-based application systems presents new challenges. This is because we do not deal with just user authentication and access control of the resources of a single organization. Rather, we deal with a network of interconnected systems and the sharing of all types of resources that belong to these organizations. There is a need for a model, a language, and a framework for modeling, specifying, and enforcing the agreement established by collaborating organizations with respect to trust and security issues. This trust agreement is needed to establish inter-organizational security policies that govern the interaction, coordination, collaboration, and resource sharing of the collaborative community.
Our study conducted basic research on and developed application-level, trust-based security technologies to support Internet-based collaborative systems. It has four specific accomplishments. First, we introduced a way to define trust agreements and develop a language for specifying the agreements. A trust agreement establishes inter-organizational security policies and constraints regarding message exchanges and resource sharing, and enables collaboration among organizations, which are originally disjointed and have their own security policies and constraints. Second, we developed a security model to capture relationships among the concepts and modeling constructs of trust and the concepts and modeling constructs of a conventional access control model. By treating trust-related concepts and constructs as “first-class” security concepts and constructs, the model allows the specification of trust policies at the inter-organizational level, which is not supported in traditional security models. Third, we established a set of criteria for evaluating nonrepudiation protocols for B2B electronic-commerce; and developed a new protocol that meets the criteria. Fourth, we designed and implemented a prototype of a network-based trust and security management system to demonstrate the enforcement of inter-organizational security policies and constraints.
CHAPTER 1
INTRODUCTION

Internet-based technologies, such as the Web technology, distributed object technologies (RMI, J2EE, CORBA, EJB, COM) and the emerging Web service technology (UDDI, SOAP, WSDL) enable people and organizations to share all types of resources, such as data, software systems, application systems, hardware facilities, and human resources. These technologies have enabled the development of Internet-based systems to support applications such as Business-to-Consumer and Business-to-Business e-commerce, virtual enterprise management; supply chain management; biomedical information network; information grids; supply-chain management; homeland defense; and integrated military command, and control and communication systems. These application areas all involve a number of collaborating organizations sharing distributed and heterogeneous data, software, and other resources over the Internet. Here, collaborative computing refers to these types of distributed systems that achieve resource sharing among collaborating organizations.

A key requirement of collaborative computing is the management of trust and security. Security issues have long been investigated in the context of a single organization exercising control over its users’ access to resources. In such a computing environment, the focus has been on protecting the resources of an organization from malicious attacks, unauthorized access, and denial of services. User identity-based authentication and role-based access control for authorization, which are subject to an organization’s security and privacy requirements, have been shown to be very effective
However, these security techniques are static and centralized. For example, users have to be known to the system beforehand. Users are typically identified by account names and authenticated by passwords. Security policies are centrally controlled and governed by a single organization (that is, the resource owner or service provider). Thus, the traditional security mechanisms are tightly coupled, static, and not adequately responsive to changes.

Developing a large-scale, Internet-based collaborative system presents new challenges for security and trust management. This is because we do not deal with just user authentication and access control to the resources within a single organization. Rather, we deal with a network of interconnected systems and the sharing of all types of resources that belong to multiple organizations. We delineate the characteristics of collaborative computing over the Internet and their corresponding research challenges as follows:

- **Collaborative computing requires the establishment of application-level, inter-organizational security policies and constraints.** Collaborating organizations have their own security systems to enforce organizational security policies and constraints. When an organization decides to collaborate, it needs to negotiate with other organizations on what computing resources it should share, what rules it should use to authenticate legitimate interactions, and which protocol it should employ to securely exchange business documents. We refer to this process as the
establishment of inter-organizational security policies and constraints among these organizations, or trust agreement in short. Note that newly established policies and constraints should not conflict with existing organizational policies and constraints. Their enforcement mechanisms are different from, but can make use of, the security enforcement mechanisms at the infrastructure level. There is an increasing need for application-level security models, tools, and protocols to specify and enforce inter-organizational policies and constraints, such as confidentiality, authentication, access control, and non-repudiation.

- **Collaborative computing involves loosely coupled organizations participating in dynamic virtual communities.** Organizations collaborate for the purpose of achieving a common goal. Their collaboration is carried out in a loosely coupled manner. By loosely coupled, we mean that collaboration may be short-lived and may change at anytime, that is, dynamic. Also, service providers and users may come and go as their roles and responsibilities change. Therefore, it is hard to predetermine the user population and their privileges of computing resources offered by an organization. It has to trust that other collaborating organizations will grant the proper users the proper credentials to access its exposed resources. Therefore, trust and trust agreements among organizations must be dynamically adjustable as changes occur.

- **Communication between collaborating organizations may go through multiple intermediaries rather than direct communication.** To achieve resource sharing, collaborative organizations need to exchange messages such as making service requests, sending purchasing orders or requests-for-quote, reporting status, transmitting data, and so forth. Messages may have to go through several intermediaries at multiple network sites. For this reason, applications in collaborating computing are exposed to a higher risk of security threats. The trust dependency and the degree of trust on these intermediaries become critical trust management issues.

The goal of this work is to conduct basic research on and to develop application-level, trust-based security technologies to support Internet-based collaborative systems.

The development of these technologies involves the integration of trust management with existing security technologies. The four specific objectives of this research are described below. Their relationships are shown in Figure 1-2.

- Introduce a way to define trust agreement and develop a specification language for defining inter-organizational security policies and constraints that govern the interaction, collaboration, coordination and resource sharing of collaborating organizations. Collaborating organizations need to agree on what subset of their resources they are willing to share, whom they would trust to authenticate the certificates of service requestors, and what authorization rules are to use to give
proper permissions. They also need to decide whom they should rely on to monitor their interactions and to meet additional security requirements like non-repudiation. In this research, we explore how trust agreement can be integrated with the key security functions such as authentication, access control and non-repudiation. We also identify security entities and modeling constructs that are relevant to inter-organizational security issues and develop a trust agreement specification language.

- **Develop a model for trust-based authentication and access control.** The role-based access control model is a well-established security model. In this work, we use it to model organizational security policies and constraints and integrate its modeling constructs with those of a trust model to form a new trust-based security model.

- **Identify additional requirements for evaluating non-repudiation messaging protocols and develop a new protocol for collaborative computing.** Interactions in collaborative computing (e.g., a web service request, an event notification, a certified mail delivery, an electronic software distribution, an electronic payment, a purchase order and a request-for-quote) can be abstracted as message transmissions and processing. Non-repudiation, with respect to the sending and the receiving of a message, is an important security issue. Several non-repudiation protocols have been proposed, and some qualitative evaluation criteria also exist. However, in the collaborative computing environment, additional requirements should be considered, and additional criteria should be introduced for evaluating the existing and new protocols.

- **Investigate a network-based security architecture and implementation techniques.** The network architecture must be distributed, scalable, reliable, and flexible. We design the components needed for trust and security management, and develop a prototype system to verify our research results. We investigate a specification-driven approach to trust and security management, which translates high-level trust agreement specifications into events, action-oriented rules and triggers. These events, rules, and triggers are then used by replicas of an event server and replicas of a rule server to enforce inter- and intra-organizational security polices and constraints.

To achieve the above objectives, we have developed a Trust-based Security Model (TSM) containing modeling constructs for inter-organizational trust and security (e.g., security policy agreement and certificate-based authentication) and for organizational security. The constructs for modeling organizational security are based on the well-established Role-Based Access Control model (RBAC) [1, 2, 3]. Our model defines the inter-relationships among these constructs in terms of mapping functions. It preserves...
the autonomy of collaborating organizations in maintaining their access control over the resources they share. We have formalized the model by adapting the National Institute of Standards and technology (NIST) methodology of the RBAC formalization. Based on the TSM, we have designed an XML-based trust agreement specification language, by which collaborative organizations can specify inter-organizational security policies and constraints.

![Diagram](image)

**Figure 1-2. Relationships among research objective**

For enforcing message-level security in a collaborative computing environment, we have identified some additional criteria for evaluating non-repudiation message transfer protocols. We have evaluated the existing non-repudiation protocols based on the new set of criteria and identified their limitations. We have also designed a new non-repudiation message transfer protocol that is better suited for the criteria.

This work also presents a network-based security system architecture and a prototype implementation. The implementation makes use of the Web service platform [4, 5, 6]. The non-repudiation message transfer protocol runs on top of the Simple Object
Access Protocol (SOAP) protocol. This work also introduces a *specification-driven* approach to trust and security management. In this approach, a high-level XML specification of a trust agreement is used to automatically generate security mapping data and an executable code for enforcing security constraints. Thus, a trust agreement on inter-organizational security and its modifications can be rapidly deployed. The TSM, the trust agreement specification language, the non-repudiation messaging protocol and the implementation technique presented in this dissertation are very general. They can be applied in many application domains that can be characterized as collaborative computing.

The organization of this dissertation is as follows. In Chapter 2, we summarize other research that is relevant to our work, explain how our work is different from other existing research projects, and point out our contributions. In Chapter 3, we address the security requirements for collaborative computing. The focus of the discussion is on how trust management can deal with the identified security requirements. We then present the Trust-based Security Model (TSM), its formalization, and the trust agreement specification language in Chapter 4. In Chapter 5, we turn to message security issues in collaborative computing and describe the non-repudiation message transfer protocol. In Chapter 6, we present the design and implementation of the key security components in the Web service environment. Finally, we give a summary and concluding remarks in Chapter 7.
CHAPTER 2
RELATED WORK

Several existing works have influenced our design and development of the Trust-based Security Model (TSM), the architecture, and the prototype implementation. We discuss them below.

2.1 Security Models

A wide range of security models has been proposed over the past several years to address the security needs of information systems. These models are categorized as either mandatory security models or discretionary security models, depending on supported policies [7]. A mandatory security model is designed to control the flow of sensitive information according to the users’ security clearance. The lattice-based access control model is an example of a mandatory security model. A discretionary security model is characterized by its flexibility in controlling data access based on the users’ identities. It allows users to grant authorization to other users. The security model used in operating systems and database systems follows this model. Recently, the Role-Based Access Control (RBAC) and the Task-Based Access Control (TBAC) have been studied [1, 2, 3, 8, 9]. They provide the high-level semantics for security specifications. Abstractions such as “role” and “task” are introduced to bridge the semantic gap between enterprise-level policies and low-level security rules. These concepts greatly reduce the intricacies of security administration. The RBAC model has shown its advantage in security management by managing the roles of users. On the other hand, TBAC was proposed to support dynamic security policies, which allow permissions to be checked-in
and checked-out in a just-in-time fashion. How to model constraints, such as separation-of-duties and chinese wall constraint, within RBAC were investigated in [10].

However, these models by themselves are not sufficient to define and enforce inter-organizational level security policies. This is because they were developed in the context of a single organization for controlling its users’ access to resources. It does not have enough constructs to represent inter-organizational security polices and constraints. We strongly believe that trust-related concepts and constructs such as certificates, certificate authority, membership, delegation, and trust agreement, should be integrated with those of existing security models (e.g., privilege, resource owner, ownership, security subject, etc.). One of our research tasks is to identify and integrate trust and security concepts to establish a formal trust-based security model. The model is also used to design a language for specifying Trust Level Agreements as opposed to Service Level Agreements (SLA) between collaborating organizations.

2.2 Security Policy Specification Languages

Several security policy specification languages were reported in the literature. Jajodia, Samarati, and Subrahmanian proposed an Authorization Specification Language (ASL) for defining authorization, conflict resolution, access control, and integrity constraint [11,12]. The language looks like a prolog program and provides constructs to specify constraints such as incompatible groups, incompatible role assignment, incompatible role activation, separation of duty, and Chinese wall constraints. Ponder is another language for security policy specification [13]. It is based on the object-oriented model and provides a declarative language for specifying polices of authorization, obligation, and refrain. Additionally, it provides constructs for organizing policies in a structured manner and constructs for defining roles, delegation, and relationships. It
allows for parameterized policies so that the policies can be customized and configured according to the deploying environment. These two languages are useful in defining intra-organizational security policies and constraints. Unlike these languages, our research focus is on the specification of inter-organizational security policies related to authentication, access control, and non-repudiation.

2.3 Distributed Trust Management Systems

Trust models and trust policies are often mentioned in the security literature [14, 15]. In the Public Key Infrastructure (PKI), a trust model is described as a hierarchical chain of certificate authorities [16, 17]. The concept of trust management was formally introduced in PolicyMaker [18]. The work demonstrated how security rules and digital credentials can be used for security policy enforcement in a distributed system. Similar to PolicyMaker, IBM developed the Trust Policy Language (TPL) [15] for defining trust policies. These policies specify the rules that map a Web service requestor to some predefined roles (or permissions) according to clients’ certificate and the certifying party. We also found other related works that describe the implementation of a trust model and trust policies [19, 20, 21].

Another type of research work on trust management is conducted in agent-based systems. Minsky took a distributed approach to security management [22], in which security policies are defined as laws. Laws govern the interactions in an agent community over the Internet. Recently, the work has been extended and a general mechanism was introduced to formulate and enforce a wide range of security policies based on the concept of law-governed interactions [23]. Distributed trust management in a supply chain management (SCM) system was also reported in [24, 20]. This work utilized security agents to enforce common policies for SCM. Policies are specified in Prolog
rules, which specify authorization. There are three types of agents in this framework; user agents, controller agents, and ticket distribution agents. User agents can make requests to perform certain actions by attaching digital credentials to the request messages. Controller agents make decision on access control. Ticket distribution agents correspond to certificate authorities in PKI.

Our approach to distributed trust and security management is different from these works in three major ways. First, instead of using an agent architecture/framework, we use replicas of servers in a peer-to-peer architecture to manage distributed events, triggers and rules, which implement trust-based security policies and constraints. Second, instead of building a network system from scratch (i.e., making no distinction between organizational and inter-organizational security rules and defining a common set of policies that all agents observe), we assume that collaborating organizations have their local security policy and enforcement mechanisms in place. Our task is to define and enforce inter-organizational security policies and constraints on top of the existing security systems. We have designed a trust agreement specification language and implemented an enforcement mechanism to demonstrate a specification-driven approach to trust management. Third, the referenced works did not deal with trust issues such as the degree of trust dependency on a third party authority and non-repudiation. We looked into trust issues of the access control and non-repudiation problem.

Another interesting work on trust management was reported recently. Winslett and et al. proposed an automated method for trust establishment between strangers (that is, parties from different security domains) using general-purpose credentials and negotiation strategies [25]. Trust establishment between strangers requires that they
exchange credentials so that they can make sure that they conduct business with the ones they want. The research problems in this context are 1) how these strangers know what credentials to exchange 2) how they determine whether to release a certain credential in spite of the possible presence of trust risk (e.g., privacy intrusion), and 3) what negotiation strategies are possible and what the architecture should be to implement the idea. In our study, the focus is not on how end users establish trust with service providers and how they determine what certificates to present. Instead, we look into how trust agreements between organizations can facilitate inter-organizational security management. We also investigate on how to rapidly deploy the trust agreement and inter-organizational security rules and constraints established by collaborating organizations.

2.4 Reputation Management Systems

Several recent works that manage reputation of peers in a peer-to-peer environment are very relevant to our work [21, 26, 27, 28, 29]. The common objective of these works is to assess the trustworthiness or reputation of peer agents by collecting some trust parameter values, such as satisfaction, complaint, context, evidence, user behavior and profile, feedback, and feedback source. These works treat all agents equally as opinion makers. Different from these works, we assume that third party authorities (TPA) are also participating in collaboration efforts, whose opinions and services are recognized as security services (that is, certification and non-repudiation services). The objective of our trust management is to establish, enforce, and monitor inter-organizational security policies regarding verification, validation, acceptance, distribution, evaluation of credential information (i.e., certificates, digital signatures, receipts, acknowledgements, etc.), and to control access to sharing resources based on the credentials and the trustworthiness of collaborating organizations.
2.5 E-contract Technologies

Internet-based collaborative applications involve inter-organizational interactions. In order to ensure the protection of the assets of all parties involved in e-commerce, interactions must be regulated by a contract, as is the case with traditional business interactions. A basic e-contracting architecture for B2B was proposed in [30], which includes key elements like a contract repository, contract notary, contract monitor, and contract enforcer. The responsibility of each element is as follows. The contract repository stores standard contract templates. Once two organizations choose a contract template and agree upon the content, the contract notary stores the contract. The compliance with contract terms is ensured by services provided by the contract monitor and the contract enforcer. They monitor, regulate and control all business interactions that have been agreed upon in a contract. Other related work in the area of e-contracting includes the EU-funded COSMOS project [31] and the CrossFlow ESPRIT project [32, 33].

Several e-contract works that propose agreement specifications for inter-organizational collaboration are relevant to our work. The Collaboration Protocol Agreement (CPA), a part of the ebXML [34], is a system-level agreement for data interchange between trading partners’ systems. Although it covers critical message security issues, such as encryption and non-repudiation, it does not have enough modeling constructs for specifying inter-organizational security policies and constraints. Agreements with respect to the resource accessibility and accountability of collaborating organizations cannot be expressed in CPA. Moreover, the handling of non-repudiation relies solely on the digital signature technology. The CPA does not address the involvement of third party authorities in a non-repudiation protocol. The Service Level
Agreement (SLA) from IBM [35] is another research effort that studies agreements with respect to qualities of services (QoS), such as throughput and downtime. The SLA specifies the QoS requirements. Different from this work, our research focus is on specification and enforcement of trust agreements with respect to inter-organizational security policies and constraints. We envision that our work will eventually be integrated with these technologies so that computer-aided collaboration design can become a reality.

Our concept of a trust agreement resembles the concept of certified contracts for regulating e-commerce [36]. However, different from the certified contract approach, a trust agreement in our approach is signed and distributed to the replicated servers. The server then generates the enforceable rules and configuration data from the agreement specification. Our approach supports the transparency property of distributed systems in that it does not require end-users to make an explicit effort to obtain and maintain (possibly multiple) contracts needed for accessing services. Another difference from the certified contract approach is that our approach makes a clear separation between the global policy and the local policy to support local autonomy; whereas, the certified contract approach does not distinguish them. Local autonomy is an important requirement in designing a trust-based security model for supporting collaborative computing.

### 2.6 Rule-based Knowledge Management Systems

Three general types of rule systems have been developed in academic research and the commercial world: logic rule systems [37], production rule systems [38], and event-condition-action (ECA) rule systems [39]. The first two types do not allow the specification and processing of events in an explicit manner. ECA rules have been used in active database management systems [39, 40, 41, 42], including our own work on an object-oriented knowledge base management system [43]. They are used in several
commercial systems for business applications (e.g., Vitria’s Automator, Haley’s Enterprise rule system, Blaze Software’s Advisor, and products by Business Rule Solution, Rule Machines, Netron, and Ontogenics.com).

An attempt to apply the trigger concept in active database systems to security enforcement was reported in [43]. The basic idea of this work is to specify when and how a workflow system can restrict the assignment of tasks to agents using authorization triggers (expressed in ECA rules). It shows that the following categories of security authorization constraints can be represented by ECA rules: dependency (time-dependency, instance-dependency, and history-dependency), scope (global, local), and verification time (static, dynamic). Examples of authorization constraints include separation of duties, binding of duties, restricted role membership, task cooperation, restricted activation, sensitive data filtering, and sensitive data management.

In our previous work, the ETR Server is developed based on an Event-Trigger-Rule (ETR) paradigm reported in [44, 45, 46]. Unlike the ECA paradigm, events and rules are defined separately. Triggers are specifications that link events to rules. This allows different organizations to define their own rules, which are triggered by the occurrences of events in a distributed computing environment. When an event occurs, distributed systems that have subscribed to the event will be notified through a notification mechanism. Distributed triggers that are associated with the event will then activate rules for processing.

A rule is a small granule of control and logic in a high-level language. It consists of a condition specification, an action specification, and an alternative action specification. Based on the result of the evaluation of the condition, either the action or the alternative
action specification is executed. Different from the existing ECA-type of rule systems, our system allows a rule definer to specify a network structure of rules, which represents a large granule of control and logic. A rule can appear in multiple rule structures and can post event(s) to trigger other rule structures.

The specification and processing of event history (or composite events) is also supported. An example of a knowledge specification based on event history is “When E1 or E2 occurs, verify if E3 and E4 have also occurred within a specified time window (event history). If so, activate a structure of rules.”

In several previous projects, we have used the ETR Server for the enforcement of business rules in the contexts of collaborative e-business environment, Internet-based knowledge networks, automated business negotiation and dynamic workflow management [45, 46, 47, 48]. The security server implemented in this work makes use of the implemented Event-Trigger-Rule Server as an underlying policy enforcement mechanism to meet the dynamic, adaptive, and rapid re-configuration security requirements (i.e., due to the contract revision, annulment or revocation of authority). We adopted the event-driven and rule-based approach to enforce authorization constraints because the event-driven and rule-based paradigm is very flexible in terms of policy specification and enforcement. Moreover, in some cases, we may want to specify a complex security rule, which takes some actions conditionally (i.e. sensitive data filtering, query modification before processing request, and cryptographic actions) along with authorization decisions. Traditional authorization specifications do not allow this type of specification.
2.7 Standard efforts on Security in the Web Service Infrastructure

There are several standardization efforts to secure the web service infrastructure. Basically, the efforts provide security tools. The Simple Object Access Protocol (SOAP) Security Extension [49] of W3C describes the syntax and the processing rules of a SOAP header to include a digital signature within the SOAP Envelope. The XML Encryption WG is developing an XML-based encryption/decryption technology to provide confidentiality of data elements that are represented as XML documents [50]. The XML Key Management Specification (XKMS) [51] is an XML-based PKI service to distribute and manage the keys that are necessary for ensuring end-to-end communication security. The PKI interoperability issue is addressed by adopting XML as a medium for electronic communication. XKMS describes a standard-based approach to adding PKI-based trust processing (digitally signing and/or encrypting/decrypting XML documents) to XML applications. The Registry Security Proposal of ebXML [34] identifies the security requirements and addressed security aspects of service registry (or broker). SAML [52] investigates a standardized way to securely exchange authentication, authorization, and profile information between trading organizations regardless of the security systems or platforms in use. Its objective is to promote a secure e-business transaction across company boundaries by the use of trust assertions, which convey trust statements on any subject, including financial transaction and authenticated data as well as public keys.

Recently, IBM, Microsoft, Verisign, and RSA have collaborated to propose a security roadmap for Web services [53]. The proposal consists of several sub-specifications. As of December 2002, the sub-specification that is relevant to this dissertation is the “Web Service Trust Language.” Another planned specification related to this dissertation is the “WS-Federation,” which has yet to be published. Using the Web
Service Description Language (WSDL), the Web Services Trust Language (WS-Trust) defines messages and operations for the issuance, exchange and validation of security tokens. Although the specification includes the description of a general message model for trust establishment through security token exchange, it does not cover how collaborating organizations come to an agreement and establish inter-organizational security policies (that is, authentication, authorization and non-repudiation), and how the agreement enables the collaboration between these organizations.

2.8 Miscellaneous Related Work

In this section, we will summarize the research prototypes that incorporate security technologies: a digital library and a distributed computing environment.

The Digital Library Authorization Model (DLAM) was proposed as a part of the digital library project [56]. It shows four interesting points that are relevant to our trust-based security model. First, the proposed model identifies an individual subject by its qualifications and characteristics (the so-called credential) rather than by its identity. The model introduces the notion of “credential” as an abstract collection of the subject’s properties. Its credential specification provides modeling constructs for expressing complex conditions of credential qualification and for specifying relationships among different credential types. Second, based on the credentials of an individual, authorization decisions are made on what kinds of contents can be accessible. Third, the model provides a language for specifying the granularity of authorization. Fourth, the paper also points out a basic distinction between a role and a credential. A credential is characterized by a set of attributes, thus easily expressing the qualification or characteristics of an individual subject. Unlike DLAM, our model determines an individual subject’s qualification (or credential) based on a trust relationship among
collaborating organizations and an individual subject’s certificate certified by trusted collaborating partners. Another difference is that we specify authorization rules by linking a credential with a role, while DLAM links a credential with a conceptual object extracted from a digital content.

Another interesting security research was carried out in the Oasis project [55], which is targeted for an open distributed environment. The authors proposed that a subject can be classified into named roles, initially by each service provider. Besides, subjects’ other named roles can be additionally identified based on the relationships between the named roles. Here, the named roles correspond to composite entities that combine a membership entity and a role entity of our model. The relationship definitions between the named roles are similar to membership derivation in our model. In our case, we consider membership and role objects as separate entities because memberships and roles are managed by different organizations in the Internet-based collaborative computing environment. Oasis also makes use of the delegation concept. Through delegation, subjects can have additional named roles. Our model also supports delegation but in a different way: the delegation in our model is done at the certification authority level rather than the delegation of rights between subjects.
CHAPTER 3
REQUIREMENTS OF TRUST AND SECURITY MANAGEMENT

In this chapter, we begin with a discussion of security threats in the collaborative computing environment and identify their corresponding security requirements. This chapter also discusses some trust concepts and the trust management issues to provide a background on a Trust-based Security Model (TSM) in the next chapter.

3.1 Security Threats in Collaborative Computing

Collaborative computing is subject to various security threats and attacks because it exposes enterprise resources to the public, and it involves exchange of sensitive data through a relatively unsecured public network: the Internet. All Internet-based collaborative systems need to satisfy the general security requirements; i.e. network connections should be secure and trustworthy in order to prevent any possible data interception and modification during data transmission. Furthermore, policy-based security mechanisms must be in place to protect resources and services against unauthorized use. This work covers these two important issues: “access control” and “communication security” in Internet-based collaborative systems.

Unlike the conventional security management in client/server systems, in which security policies are defined and centrally managed according to a single organization’s regulation, the characteristics of Internet-based collaborative computing present unique challenges. This is because we do not deal with just user authentication and access control to the resources of a single organization. Instead, we deal with a network of interconnected systems and the sharing of all types of resources that belong to these
organizations. In the collaborative computing environment, the requirements of trust and security management are quite different from those of the client-server environment. We shall delineate some new requirements as follows:

- **Requirement 1**: In the collaborative computing environment, an organization cannot predetermine the users of its resources and their access privileges. Instead, collaborating organizations need to establish a trust agreement among them and manage and enforce the agreement. The establishment, management and enforcement of trust agreements represent a new dimension of collaborative computing.

- **Requirement 2**: Collaborative computing is the joint responsibility of organizations that interact and collaborate. No single organization can dictate what security policies should be enforced across organizational boundaries. Policies often need to be negotiated and agreed upon by participating organizations. A collaborative computing system should be able to enforce not only individual organizations’ local policies but also those global policies.

- **Requirement 3**: An organization may participate in multiple virtual communities based on different needs and contexts of collaboration. Its membership in these communities can be short-lived and may constantly change (i.e., dynamic). Also, the user population of a virtual community is dynamic in that its users may change their roles and responsibilities. Furthermore, changes may occur in organizational relationships, security/privacy/safety policies and constraints, contextual information, and resources. The enforcement of security and constraints cannot be static and tightly coupled to applications. A collaborative computing system must be dynamic and adaptive to account for these changes without having to modify the existing applications.

- **Requirement 4**: Communication between collaborating organizations may be established through multiple intermediaries rather than directly. In such a case, the trust dependency and the degree of trust on these intermediaries and other security issues need to be addressed in the architecture and implementation.

### 3.2 Trust, Trustworthiness, and Trust Management

Trust is an abstract concept, which is described as a relationship between/among persons or organizations. It is closely related to concepts of reliance, dependence, promise, confidence, and/or belief. Trust is essential in reducing risk and uncertainty when a person has to work in an environment over which he has no control. The Internet-based collaborative computing environment is such an environment in which
collaborating parties may have to rely on intermediaries’ security services to meet security requirements such as confidentiality, access control and non-repudiation. With respect to security and trust management, we identify the following important properties of trust and trustworthiness:

- **Trust is associated with risk:** As stated previously, putting trust in another person or organization creates vulnerability. We need to consider risk factors when evaluating the trustworthiness of a transaction with an entity. Conceptually, we can say that trustworthiness is evaluated as \( f(\text{confidence}, \text{risk}) \), where \( f \) is an arbitrary evaluation function.

- **Trust is dynamic and transient:** Experience and knowledge about a business entity is accumulated with time. As a result, the degree of trust in the entity is constantly re-evaluated and changes with time.

- **Trust and trustworthiness is subjective:** Trust is not an objective property of an entity, but a subjective degree of belief in the entity [56]. It is based on the truster’s prior experiences and knowledge. The degree of trust ranges from complete distrust to complete trust. There is also a case where we are ignorant of an entity, thus we simply have no opinion about the matter of trust in the entity.

The source of knowledge on an entity may come from outside. As noted in Web Services Trust Model of WS-Trust [53], collaborative computing needs some trust services (e.g. certification, non-repudiation, and service evaluation and rating). In the WS-Trust, trust services are referred to as “security token services.” In the large-scale collaborative computing systems, ‘Third Party Authority (TPA)’ usually provides trust services. A TPA is an independent authority trusted by collaborating organizations and individuals. Its security service is trusted because it is fair and open. For example, collaborating parties may rely on TPA for certification and non-repudiation requirements, as shown in Figure 3-1.

The most well-known type of TPA is Certification Authority (CA). Several commercial CA(s) are currently doing business in the Internet. A CA verifies public keys and identities and issues certificates using public key cryptography. The acceptance of a
certificate is a matter of trust because the certificate is accepted and honored only if there exists a trust relationship between an organization that authenticates the certificate and the authority that issues the certificate. Other types of TPAs may provide different security services such as key management and non-repudiation.

Figure 3-1. Trust relationships in collaborative computing

Each organization has its own view on who are the trusted authorities, which may change with time (based on its trust experience), and defines its own trust policy that determines which certificates it would accept. Managing the level of trust on TPAs is therefore a key security requirement. Trust management consists of trust establishment, enforcement, and monitoring. We refer to the agreement on inter-organizational security policies as “trust establishment.”

There are various ways to establish a trust agreement. For example, a trust agreement can be negotiated if the collaboration is among peers. It can be specified as an e-Contract in XML [31, 32, 33, 34] and later can be exchanged and modified by collaborating organizations. A trust agreement can also be specified by one party (e.g., a service provider) and accepted by another (e.g., a consumer of the service). A trust agreement can also be declared by a controlling entity (e.g., global policies declared by the project office of a joint venture).
A trust agreement, once established, is deployed (that is, translated into executable security rules in security systems) to enforce the inter-organizational security policies. It is important that none of the trust agreements compromises or conflict with an existing individual organization’s policies and constraints. The collaboration effort should complement rather than replace existing local security policies and constraints.

Last but not least, the trust agreement should be monitored at each organization. For each collaboration environment, a number of useful trust evaluation parameters can be defined. An example parameter for evaluating trustworthiness is the frequency that the user violates a particular security constraint. A parameter for measuring the trustworthiness of a Web service is the reliability of the service or the latency that data are returned by the service. Other non-trust/security-related parameters such as financial condition, credit, payment record, and trust parameters, as proposed in [27, 28, 56], can also contribute to the trustworthiness of a security subject. The monitoring of these parameters not only affects the currently effective trust agreements, security/privacy/safety rules, and the run-time states and data of a collaborative system, but also triggers a counter-measure automatically if some constraints are violated repeatedly.

Based on the concepts and issues related to trust, trustworthiness and trust management discussed here, we present a Trust-based Security Model (TSM) in the next chapter.
CHAPTER 4
TRUST-BASED SECURITY MODEL (TSM) FOR ACCESS CONTROL

In this chapter, we present a Trust-based Security Model (TSM) for collaborative computing. First, we give the definitions of the basic security entities used in our model, which have been used in the literature on security. Then, we give an informal description of the Trust-based Security Model with a diagram. The informal description of the model is then formalized. Based on the formalized model, we present a trust agreement specification language and its usage in a scenario.

4.1 Definitions and Terms

4.1.1 Subject, Object, and Operation

A subject is an end-user entity (that is, a real end-user, agent or application acting on behalf of a user or a company) that initiates operations on a resource. It has a unique identifier with a set of security attributes (such as its clearance and membership). Database management systems authenticate each subject by a password. Once authenticating a subject, systems retrieve the subject’s profile that contains associated security attributes. In collaborative computing, subjects may also carry digital certificates, which certify their associated security attributes.

An object refers to a resource entity under access control. Examples of objects are HTML/XML documents, database objects (tables, views, database itself), and the Web-service objects. They may be organized in a directory structure so that access rules and constraints can be specified in terms of object types [11, 12]. An object may also be associated with a security attribute (top secret, secret, confidential). Each object may
have one or more access points (called “operations”), with which information
encapsulated within an object can be manipulated. Methods of the conventional object-
oriented model are considered as *operations*.

### 4.1.2 Roles

A role is a very general term having different semantics, depending on the context. For example, in the context of workflow management systems, a role represents organizational responsibilities and functions (that is, service providers, service requestors, service brokers, etc.). In an access control model, two definitions of a role are found in the literature [57]: 1) a role is a named collection of users and permissions and possibly other roles; and 2) a role is a named collection of permissions and possibly other roles. The difference is whether users are considered in the definition of a role. In our work, since we propose to deal with users and group management separately from role management, we choose the second definition. A role collects a set of access rights (or permissions) into a single entity to simplify authorization.

Our approach, which separates the role specification from the management of role authorization, has the following two benefits: 1) it allows a role definer to define a role without having to be concerned about who will actually play the role; and 2) it allows for a distributed administration of access control because the decision on who can play a role can be negotiated, agreed, and managed by collaborating organizations.

### 4.1.3 Certificates

A certificate [58] is a data record or document about a subject (an individual, company or server), digitally signed by a trusted entity (e.g., a Certificate Authority (CA)). It is used to assert and prove a subject’s attributes, such as distinguishable properties (name, address, public key), demographic information (age, sex), transactional
information (credit card number, credit limit, available credit), and relationship
information (group membership, relationship to other groups). A certificate is referred to
as “credential assertion” in the SAML project [52]. As the CA uses its private key to sign
certificates and the CA’s public key is well-known, the integrity of certificates can be verified using the public key cryptography. Different collaborating organizations may choose different certification services, provided by either third-party Certification Authorities (CA) or in-house built-in CA, depending on the degree of trust, reputation and partnership.

Certificates are employed in making many applications secure. SSL/TLS, a security technology, require that certificates be exchanged for mutual authentication. A service requestor verifies the identity of the service provider by reviewing the server-side certificates. Conversely, a service provider also does the same to establish a secure connection in PKI (public key infrastructure). Recently, we have observed some variations of certificates (i.e., attribute certificates and smart cards) used in real-world applications to authenticate a service requestor’s security attributes (membership, role, security clearance, identity, etc.) [58]. The use of attribute certificates for access control in large-scale Internet-based applications depends on the existence of public-key certificates and the public key authentication protocol (for example, SSL/TLS). This is because public keys are used to authenticate each other in Internet-based applications and each attribute certificate contains attribute information associated with the corresponding public key. Thus, authentication of public keys is a prerequisite to the use of attribute certificates for authorization.
A certificate may go through three different stages: requested, valid and invalid. A request for certification creates a skeleton of a certificate that has yet to be signed. Then the skeleton is sent to a CA for certification. Once a certificate is signed, it becomes valid. Later, a certificate becomes invalid for two explicit reasons: 1) when the current date is not within the valid period stated in the certificate; and 2) when the certificate holder is no longer entitled to have the certificate. In the latter case, the certificate is explicitly revoked by a Certificate Authority.

4.1.4 Memberships

We also make use of the well-established notion of membership. Generally speaking, a membership represents a state of being a member of a group, which is usually associated with certain privileges. By presenting a certificate or a smart card, an individual subject can prove its membership. The membership concept is useful in defining authorization rules because we can assign a set of privileges to a group of people instead of giving authorization to individual users one by one. This reduces the complexity of managing authorization.

Another important property of membership is that an individual subject’s membership is not static because membership represents a state, and the state of the subject’s membership can change in a collaborative computing environment.

4.1.5 Security Constraints

A security constraint, in its general usage, refers to a statement that restricts someone from doing something. It is intended to maintain system integrity. It is also defined to describe exceptional security rules, such as temporal restrictions. The constraint may check the trustworthiness of a requestor based on information stored in the auditing database. It may also evaluate the trustworthiness of a transaction by considering
the location, time, and risk associated with the transaction. In a sense, security constraints are used to detect an un-safe state. In the Trust-based Security Model (TSM) that we shall present in Section 4.2, security constraints are expressed in terms of conditional statements that specify the inter-relationships between entity types. The condition part of a security constraint makes reference to the contextual information accessible to and verifiable by a security system.

The violation of security constraints can be handled in different ways. The simplest approach is to just disable (or un-activate) the inter-relationship between entity types defined in TSM and reject the service request. The violation can also be handled by raising exceptions or events, which trigger some counter-measure rules. These rules then perform actions, such as sensitive data filtering, query modification before processing requests, and cryptographic actions.

4.2 Trust-based Security Model (TSM)

In order to define and enforce inter-organizational security policies, we need a new formal security model that allows the security policies or rules to be defined in terms of trust relationships among collaborating organizations. Quite a number of security models have been proposed over the past several years to address the security needs of authentication and access control in information systems [59]. However, these models are not adequate to meet the inter-organizational access control requirements. In order to define inter-organizational access control requirements and policies, a security model must integrate the trust-related concepts, such as certificate, certificate authority, user membership, delegation and trust agreement, with those of the existing security models, such as permission, role, operation, security object, security subject, resource owner and ownership. Trust model and trust policies are often mentioned in the security literature
[7, 15, 18–21, 52]. However, there is still no formal treatment that captures the security concepts in collaborative computing and their semantic relationships. In our work, we identify the security constructs from a well-established organizational security model (that is, Role-Based Access Control (RBAC) model), and the trust constructs from trust management, certificate-based authentication, and constraint specification. We then define their inter-relationships to form an integrated Trust-based Security Model (TSM). We took this approach instead of attempting to invent a brand-new model because collaborative computing is designed on top of existing security technologies. The collaborating organizations will still maintain their autonomy in deciding which subset of their resources they are going to share under what constraint. We also incorporate the constraint specification into our model so that polices can be adjusted easily, if necessary. Our model is trust-based in that the policies/constraints governing the authentication and access control are negotiated, agreed, and enforced.

Figure 4-1 shows the design of TSM [60, 61], which consists of three parts: role-based access control model (RBAC), trust-based authentication, and trust establishment. Access control in TSM is based on the role-based access control (RBAC) model [1, 2, 3]. As shown on the right side of the figure, a resource owner can own many resource objects (RO) (i.e., 1-to-n cardinality). A resource object has many operations (OP) and an operation can be performed on many resource objects (m-to-n cardinality). A set of such associations defines a privilege (or permission). A role (R) can acquire one or more permissions and a permission can be acquired by one or more roles (m-to-n). To simplify the diagram, we do not represent the cardinalities but will discuss them when we formalize the model in Section 4.3.
The authentication part of TSM incorporates two established concepts: membership [55, 57, 62] and certification-based authentication [15, 58, 63], as shown in the dotted-line box on the left side in Figure 4-1. They are added to TSM to support the distributed and dynamic nature of Internet-based applications. Unlike the traditional method of authentication (i.e. verifying pre-assigned userids and passwords), a Certificate Authority (CA) is used to issue certificates that certify subjects’ membership.

Figure 4-1. Trust-based security model (TSM)

Note that membership entities are defined independently of role entities. We model this way because in collaborative computing, different organizations manage roles and memberships. Roles and associated privileges are usually managed by service provider
organizations; whereas memberships are independently verified, certified, and managed by Certificate Authorities (CA). Role authorization is not embedded into certification, and thus allows role management to be loosely coupled with membership management. Since the management of roles (or privileges) is de-coupled from the management and certification of users and their membership, change to a role or user membership will have isolated impact on each administration.

The membership of a subject can be determined from a digital certificate, as stated earlier. An individual user would obtain its certificate(s), which includes membership information and additional attribute information, from a trusted Certificate Authority (CA). From a resource owner’s perspective, a CA is an information provider that provides information about an individual. The acceptance of endorsement is a matter of trust on a CA. Our model captures this trust concept by defining a CA as an entity, which is trusted by the resource-providing organization or is one whose authority has been delegated by another trusted CA. Furthermore, the membership of a subject can also be determined (or derived) from other relevant memberships of the subject (m-to-n). This is analogous to the situation where a user proves his financial stability by using a number of bank statements. The traditional group-based access control approach [64], which organizes groups in a hierarchical manner, is a special kind of membership-to-membership relation. Basically, a group is organized in a hierarchical manner according to generalization/specialization relationships. Since relationships between memberships in a collaborative computing environment are not necessarily hierarchical, we decide to capture membership derivation instead of group hierarchy.
A trust agreement, shown on the top of Figure 4-1, represents relationships between collaborating organizations regarding security and trust policies. To establish a trust agreement, a resource provider organization (RPO) and a resource requestor organization (RRO) would negotiate with each other to define a set of security policies and constraints that they mutually agree to enforce. The negotiated trust agreement contains, among other points, rules such as which CA should provide the certification service, which membership should be mapped to which particular role, and what constraint should be associated with the mapping (e.g., a subject with membership M can only play the role R during working days).

The TSM also includes a constraint construct for defining a variety of conditional restrictions. A security expert can model security constraints as conditional mappings between the entities of the entity types defined in the model (shown in Figure 4-1 by arrows with “bubbles”). The conditional statements are specified in terms of contextual information accessible to a virtual enterprise.

4.3 Formalization of TSM

We formalize the Trust-based Security Model (TSM) by adopting the methodology of the National Institute of Standards and Technology (NIST). Like the formalization method of NIST’s RBAC, we organize the definitions of entities and relationships in the layered way. It is layered in that the definitions of the lower level are used to define the entities and relationships in the upper layer. The layers are the basic TSM, the role hierarchy/membership derivation support, the security constraint support, and the trust agreement. The layer of basic TSM identifies security entity types and defines inter-relationship types among these entity types. On top of the basic layer, the role hierarchy and the membership derivation are defined. Then security constraints can be defined in
the next layer. Note that constraints can be defined either with or without the definition of role hierarchy and membership derivation. Based on entities and relationships defined in the lower level, the trust agreement is defined at the top layer.

Figure 4-2. Comparison with NIST’s RBAC methodology

At the basic layer, we identify a set of basic entity types (i.e. Certificate Authority (CA), OWNER, Membership (MS), Operation (OP), Object (OJB) and Subject (SUBJ)), and define two composite entity types (i.e. Privilege (PRV), and Certificate (CTR)). In addition, the basic layer defines a set of relationship types in terms of mappings. The layer of basic TSM is defined as follows:

**Definition 1:** The basic TSM

The security entity types

- **Primitive entity types:** $CA$, $OWNER$, $MS$, $ROLE$, $OP$, $OBJ$, and $SUBJ$, which stand for certificate authorities, resources owners, memberships, roles, operations, resource objects and subjects, respectively.

- **Composite entity types:** $PRV$, and $CTR$ where
  
  $PRV = 2^{(OP \times OJB)}$: a set of privileges
  
  $CTR = CA \times SUBJ$: certificate types

The inter-relationship types

- $DETERMINE \subseteq MS \times CTR$ is a certificate-to-membership relation, whose instances are defined by a 1-to-1 mapping function: $certified\_membership(CTR) \rightarrow MS$. For $c \in CTR$, $certified\_membership(c) = m$ where $m \in MS$, and $(m, c) \in DETERMINE$

- $PLAY \subseteq MS \times ROLE$ is a membership-to-role assignment relation, whose instances are defined by a many-to-many mapping function: $assigned\_roles(MS) \rightarrow 2^{ROLE}$. For $m \in MS$, $assigned\_roles(m) = \{ r \in ROLE | (m, r) \in PLAY\}$
Note that we consider a role as “a named collection of privileges” at the basic layer.

This will be extended in the next layer (role hierarchy) so that a role can also be defined by an inheritance from another role, not just by a set of privileges. As stated in Section 4.1.2, we consider a role as “a named collection of permissions, and possible other roles” [57].

Based on the definitions of the basic layer, we then define role hierarchy and membership derivation. We made a slight modification to the NIST’s definition of role hierarchy. The TSM’s role hierarchy represents a partial order, which defines a seniority relationship between roles, whereby a senior role acquires the privileges of its juniors. The difference is that no consideration of users is needed in the definition of TSM’s role hierarchy. We take this approach because role authorization in collaborative computing needs to be negotiated and agreed upon by collaborating organizations, and thus roles should be defined independently of who will actually play the roles.

- **GET ⊆ PRV × ROLE** is a privilege-to-role assignment relation, whose instances are defined by a many-to-many mapping function: assigned_privilege(ROLE) → $2^{PRV}$. For $r \in ROLE$, assigned_privileges(r) = \{ p \in PRV \mid (p, r) \in GET \}

- **OWN ⊆ OBJ × OWNER** is a resource-to-owner ownership relation, whose instances are defined by a 1-to-m mapping function: owned_resources (OWNER) → $2^{OBJ}$. For $owner \in OWNER$, owned_resources (owner) = \{obj \in OBJ \mid (obj, owner) \in OW \}.

- **DELEGATE ⊆ CA × CA** is a CA-to-CA trust relation, whose instances are defined by a 1-to-many mapping function: delegate(CA) → $2^{CA}$. For $ca \in CA$, delegate(ca) = \{ ca' \mid ca' \in CA, ca \neq ca', (ca, ca') \in DELEGATE \}
We formalize the membership derivation as follows. A derivation of membership represents a relationship among memberships.

**Definition 2a:** The Role Hierarchy in TSM:

- \( \text{INHERIT} \subseteq \text{ROLE} \times \text{ROLE} \) is a partial order on \( \text{ROLE} \) called the inheritance relation, written as \( r_1 \succeq r_2 \) if and only if all permissions of \( r_2 \) are also permissions of \( r_1 \). That is, \( r_1 \succeq r_2 \Rightarrow \text{authorized_permissions}(r_2) \subseteq \text{authorized_permissions}(r_1) \).

- \( \text{authorized_permissions} (\text{ROLE}) \rightarrow 2^{\text{PRMS}} \) is the mapping of a role \( r \) onto a set of permissions in the presence of a role hierarchy. Formally: For \( r \in \text{OWNER} \), \( \text{authorized_permissions}(r) = \{ p \in \text{PRV} \mid r \succeq r', ([p], r') \in \text{PLAY} \} \)

We formalize the membership derivation as follows. A derivation of membership represents a relationship among memberships.

**Definition 2b:** Membership (MS) Derivation in TSM:

- \( \text{DERIVE} \subseteq \text{MS} \times \text{MS} \) is a MS-to-MS relation whose instances are determined by a many-to-1 mapping function \( \text{derive}(2^{\text{MS}}) \rightarrow \text{MS} \). Formally, \( \text{derive} (\{m_i \mid m_i \in \text{MS}\}) \rightarrow m_j \in \text{MS} \) where \( m_i \neq m_j \), \((m_i, m_j) \in \text{DERIVE}\).

We define the security constraint in the next layer. A security constraint, in its general usage, refers to a statement that restricts someone or some organization from accessing resources or playing a certain role and so forth. Security constraint is defined as follows:

**Definition 3:** The Definition of Security Constraint:

- A security constraint is a conditional mapping function \( \text{Constraints}(A, C) \rightarrow B \), where \((A,B)\) represents any relations defined at the lower layer and \( C \) is a set of contextual statements that return Boolean values. An instance of the mapping from one entity type in \( A \) to another entity type in \( B \) is enabled if a contextual statement in \( C \) is evaluated to be true.

In TSM, security constraints are defined in terms of conditional inter-relationships between entity types. They are defined to detect un-safe states of a collaborative computing system. The violation of security constraints may raise exceptions, which trigger actions such as sensitive data filtering, query modification before processing a
request, and cryptographic actions. A contextual statement $C$ is defined on contextual information accessible to a virtual enterprise (contextual data), which may include information on a Web session, access history, communication status, IP address, events/state of virtual enterprise, and so forth.

With the definitions of entities, relationships in the basic layer, role hierarchy, membership derivation, and security constraints, we are ready to define trust agreement for inter-organizational service access control. A trust agreement specification contains at least instances of DETERMINE, PLAY, and CONSTRAINT.

**Definition 4**: Trust Agreement for inter-organizational access control:

- A **trust agreement** is a set of entities and relations containing \{MS, ROLE, CA, CTR, DETERMINE, PLAY, CONSTRAINTS\} that are agreed upon.

### 4.4 Trust Agreement Specification

Based on TSM, we have designed a high-level specification language for describing trust agreements. The need for this specification language is clear, as mentioned in Chapter 1. Collaborating organizations need their agreement to be specified explicitly in terms of what subset of their resources they are willing to expose to whom, and how they can protect messages from any kind of threat, especially at the application level. Note that in this work the trust agreement specification addresses only the security-related issues (i.e., certificate-based authentication, role authorization and non-repudiation). Other types of inter-organizational policies, such as monitoring or prevention of non-compliance and punishment of policy violation, are important but beyond the scope of this dissertation.

We will use an example scenario to illustrate the key constructs of the trust agreement specification. In this scenario, we assume that “ORG-S”, a supplier, exposes
its order processing system as a Web service (or any other remote service invocation technology, such as RMI, grid computing) and defines the “OrderRequestor” role internally for role-based access control. Now a buyer organization called “ORG-B” decides to make use of the ORG-S’s services to order parts and products for its departments. A policy negotiator, Bill, who works for ORG-B, is asked to establish a trust relationship with ORG-S. While he gathers the background information to prepare for negotiation, he quickly realizes that most of the department managers in ORG-B have already obtained digital certificates. Their certificates were mostly issued from the certificate authority FEDERAL_CA, except a few of them were issued from the certificate authority FLORIDA_CA. In order to save time and money, he decides to reuse the existing certification infrastructure. He also notices that order processing requires signature verification and the tracking of receipts. He knows that a third party called “ReceiptDistributor” is trustworthy for this non-repudiation requirement from his previous experience. With this background, he writes the following set of trust policies:

“Our company has a user group called ‘manager,’ to whom we want to give the authorization to access your ordering system. Most of them have certificates from FEDERAL_CA, while a few of them are still using their certificates from FLORIDA_CA. Please accept the latter as a delegate of the former for a while. And, our company policy requires the using of the non-repudiation service provided by ReceiptDistributor for communication security”

Bill specifies these policies in a specification document and then sends the document to the supplier ORG-S. Upon receiving the document, ORG-S assigns the reviewing task to Alice. Alice uses a tool to browse and evaluate the document and add a
few additional conditions. She suggests the following constraints to the document and returns it to Bill.

“Since FEDERAL_CA issues other types of certificates and issues to other organizations, let us consider only certificates issued to your company and your company’s managers. And your managers who use our order processing systems must be very trustworthy (say, with a measure greater than 0.7 out of 1).”

Bill reviews the modified document and agrees with the modified trust agreement specification. Alice then deploys the stated policies in Org-S. This scenario is simple but serves to illustrate the agreement-based trust establishment through negotiation.

4.4.1 Structure

We have designed an XML-based language for defining trust agreements. The complete DTD and a specification document for the above scenario are included in the Appendix. At the top level, a trust agreement specification consists of two sections: 1) description of the organizations and the parties involved in the agreement and 2) a set of trust policies that have been agreed upon, as shown below.

```
<!ELEMENT trustagreement (organizations, policies) >
......
<!ELEMENT organizations (collaboratingParty+, ca+, tpa+) >
<!ELEMENT collaboratingParty (contact, ServiceDefinition, exportedRoles) >
......
<!ELEMENT ca (publickey, revocationRepository) >
<!ELEMENT tpa (wsdl, contact) >
......
<!ELEMENT policies (membership+, delegation+, msderivation+, rolegrant+, nonrepudiation?) >
......
```

4.4.2 Organizations

The organizations part describes the parties involved in an agreement. Depending on its role in collaboration, an organization can be one of the following types:

collaborating party, certificate authority, third-party authority. A collaborating party is
the major organization that shares its resources with other collaborating organizations and/or makes use of another organization’s resources. The modeling construct includes the contact information of the organization in a URL and its service interface in WSDL (Web service technology) or in IDL (CORBA technology). Moreover, the modeling construct, if it describes a service provider, includes role privileges associated with the exposed service(s). Role privileges, defined by service providers, are referenced in the construct so that role-based authorization policies can be specified later. In this scenario, both supplier ORG-B and buyer ORG-S are collaborating parties. The following example shows the specification that ORG-S exposes its resource (i.e., orderProcessing.wsdl) and a role privilege (i.e., Order-Requestor).

```xml
<collaborating Party id ="ORG-S">
  <URL>  "http://www.org-s.com" </URL>
  <ServiceDefinition>  "https://www.org-s.com/orderProcessing.wsdl" </ServiceDefinition>
  <ExportedRolePrivileges> Order-Requestor <ExportedRolePrivileges>
</collaborating Party>
```

A Certificate Authority (CA) organization (or party) is an authority that is trusted for certification. Different types of certificates (for instance, public key certificates, membership certificates, or attribute certificates), certified by different CAs, can be specified. Note that the CA has the responsibility for information it certifies, but it is up to the organizations in agreement to determine how information in certificates is to be used for security enforcement. The trust policy part of an agreement specifies role authorization for this purpose.

The CA construct has two attributes: 1) a location (or URI) to a CA’s public key and 2) the CA’s repository that stores revoked certificates. The CA’s public key is needed for verification of certificates and their integrity. Information about the CA’s revocation
repository is also needed because a certificate could have been revoked or become invalid before it reaches its expiration day. CA periodically publishes a list of revoked certificates, which can be cached by the verifying organizations to reduce communication overhead. As an example, the CA “FEDERAL_CA” in our scenario is described below.

```xml
<ca id="FEDERAL_CA">
  <public key> http://ca.virtual.com/pk </public key>
  <revocation_repository> "http://ca.virtual.com/revoke/list" </revocation_repository>
</ca>
```

Another organizational entity is the Third Party Authority (TPA). A TPA is an independent authority, trusted by collaborating organizations, that performs some fair and open security service(s). It may involve the monitoring of collaborative activities. For example, a TPA may monitor the protocol used by communicating parties to keep track of digital evidence or monitor communications for determining the quality of service (QoS). Note that we have separate constructs for CA and TPA even though a TPA can play the role of CA. The reason is that a certifying authority does not have to be a third party. In other words, depending on the relationship between a service provider and a requestor organization, an agreement may include the CA of the partner organization, instead of a third party. Shown below is a description of TPA called ReceiptDistributor, which monitors a non-repudiation protocol.

```xml
<tpa id="ReceiptDistributor">
  <serviceDefintion> http://www.receipt.com/axis/non-repudiation.wsdl </serviceDefintion>
  <contact >http://www.receipt.com/axis/ReceiptDistributor</contact>
</tpa>
```

### 4.4.3 Trust Policies

Once we identify the parties in a trust agreement, we may specify trust policies. Currently, our specification language supports three types of trust policies that are
relevant to inter-organization security issues: membership acceptance policies, role
authorization policies, and non-repudiation policies.

Membership acceptance policies specify how to authenticate service requestors’
membership and other security attributes. In a large-scale internet-based collaboration
system, the membership of service requestors needs to be authenticated in addition to
their identification and other security attributes. This is because a security policy is
usually defined in terms of a general entity like “a manager having access to a service”
rather than “a specific person, Jane, is allowed to access the service.” Authentication is a
prerequisite for correct access control. Actually, authentication and authorization are
inseparable; the result of authentication carries the data that are used for making an
access control decision. For instance, in our scenario, the supply organization ORG-S
should be able to authenticate if a service requestor is actually a manager working for the
partner organization, ORG-B, and his trust level is greater than 0.7.

Our survey of previous works [55, 60, 61] uncovers three widely recognized
mechanisms for authenticating subjects’ membership: direct membership certification,
delegation, and derivation. Obviously, the requestor can show its membership by
presenting the membership certificate that is obtained directly from a trusted CA.
Similarly, presenting a membership certificate issued by the delegate of the trusted CA,
the second method, is also acceptable. Finally, a subject can prove membership \( ms \) by
presenting a set of other memberships that are closely related to \( ms \). To support these
three mechanisms, our specification language includes three policy types: \textit{membership},
\textit{delegation}, and \textit{membership_derivation}. We will describe them with examples below.
A membership policy is defined in terms of who is (or are) trusted to issue what membership certificates and what constraints are associated with the certificates. In our scenario, we have a trust policy saying that ORG-S agrees to accept ORG-B managers’ membership certificates issued by FEDERAL_CA (that is, the subject must work for ORG-B and his/her job title is ‘manager’). The policy is specified as follows. Here, “this.organizations” is a keyword referring to the list of organizations (ORG-S and ORG-B in the example) that are bound to the agreement:

```
<membership id="manager">
  <ca> FEDERAL_CA </ca>
  <attributes> text:job_title, text:company_name, double:trust_level </attributes>
  <condition> this.organizations.contains(company_name) AND (job_title == 'manager') </condition>
</membership>
```

A delegation policy is another way to recognize requestors’ membership. Certification Authority (CA) may be delegated from one certifying authority (CA) to another. The delegation relationship should therefore be considered when checking membership certificates. From the run-time point of view, the delegation seems to be the same as having another CA in the CA list. However, there are some cases in which organizations want to explicitly represent a delegation relationship between certificate authorities. For example, the delegation might be accepted on a temporary basis. The delegation is also considered when the security policy definer wants to define an explicit trust chain so that the deletion of a CA from a trusted CA list would automatically disable the delegated authorities in a cascaded manner. The following example demonstrates a policy stating that FLORIDA_CA plays the delegate role of FEDERAL_CA for issuing certificates to managers.
The attribute `furtherDelegate` shown above specifies the propagation property of delegation. We may use “*” to mean any level of further delegation. Otherwise, we may use an integer to specify the number of times that the delegation can be further delegated. In this example, “1” means that delegation stops at Delegate_CA.

Another way to determine the membership of requestors, which is not given in the scenario, is by the other membership(s) that one holds currently. For example, a student with ACM student membership can be recognized as a college student. Another example is that a system may recognize the requestor as an IT engineer specialized in computer engineering if the requestor has a degree in computer science and a few patents in the IT field.

```
<delegation id="delegation_relationship1">  
  <delegator> Federal_CA </delegator> 
  <delegatee> Florida_CA </delegatee> 
  <authorityTobeDelegated>manager</authorityTobeDelegated> 
  <furtherDelegate> 1 </furtherDelegate> 
</delegation>
```

```
<membershipDerivation id="ms_link1">  
  <have>ACM_Student_Membership</have> 
  <willhave>College_Student</willhave> 
</membershipDerivation>
```

Based on authentication policies (that is, how to determine requestors’ membership), authorization policies can be specified in terms of membership-to-role mappings. The specification states explicitly how membership is related to authorization. Role authorization is considered as a trust policy from the perspective that the role granting organization (or the resource provider organization in the model) grants a set of role privileges to a certain membership holder on the basis of its trust in membership certification. A role authorization policy may be enabled or disabled depending on the constraints defined on a mapping relation. For example, a policy may state that the buyer
organization ORG-B’s managers are able to play the OrderRequestor role defined by a supplier organization ORG-S. The policy is disabled if the trust level of the certificate presented is less than 0.7.

\[
\text{<roleGrant id="Authorization1">}
\text{ <membershipID>manager</membershipID>}
\text{ <roleID>OrderRequestor</roleID>}
\text{ <condition> (manager. trust_level > 0.7) </condition>}
\text{ </roleGrant>}
\]

Last but not least, our specification language includes a construct for describing a secured non-repudiation message protocol that is to be used in message transfer. Non-repudiation is an important requirement. In case the protocol relies on the security service of a third-party organization, the name of the third party needs to be specified. The following example shows a policy specification that the non-repudiation service provided by “ReceiptDistributor” is to be used in message transfer.

\[
\text{<non-Repudiation id="Comm_Protocol">}
\text{ <protocol> UF_Non_repudiation</protocol>}
\text{ <tpa>ReceiptDistributor</tpa>}
\text{ </non-Repudiation>}
\]
CHAPTER 5
A NON-REPUDIATION MESSAGE TRANSFER PROTOCOL

Non-repudiation is an important issue in all types of e-applications. Quite a number of non-repudiation protocols have been proposed, and criteria for qualitative evaluation of these protocols also exist. However, there are additional requirements in collaborative computing that should also be considered when evaluating these protocols. In this chapter, we analyze the existing non-repudiation protocols with respect to these requirements and propose an improved protocol.

5.1 Overview of Non-repudiation

In B2C or B2B e-commerce, organizations/people exchange resource requests, data, business documents, agreements, payments, contracts, acknowledgments, and so forth. These exchanges can be abstracted as message transfers among members (users or automated systems) of a virtual community. Non-repudiation in message transfers is a key security issue. A sender or a receiver should not be able to deny that a message has been sent or received if the message transfer actually took place. Non-repudiation is a security service, which creates, collects, validates, and maintains cryptographic evidence of an electronic transaction to support the settlement of a possible dispute [65].

Many non-repudiation protocols have been proposed in the literature and some criteria for evaluating these protocols have been proposed [65, 66, 67, 68, 69]. In his book, Zhou [65] compares the merits and weaknesses of eleven non-repudiation protocols qualitatively in terms of the third-party involvement (e.g., inline, online, or offline), communication overhead (high, medium, or low), privacy protection (good, average, or
poor), and timely termination (yes, possible, or no). In the context of e-applications, additional evaluation criteria are required. For example:

- **Fairness**: Depending on who can control the execution of a messaging protocol, the protocol can be biased to either the sender or the receiver, or can be fair to both. For example, in order to protect a message sender from the receiver’s repudiation of the receipt, a protocol can be designed in such a way that the message sender can control the commitment of the messaging protocol by not releasing the encryption key until he gets a receipt from the receiver. Such a protocol is in favor of the sender and is not so fair to the receiver. In B2B e-commerce, business organizations can negotiate to determine the non-repudiation protocol that should be used. The fairness of a protocol in terms of control over commitment of transactions can be an important consideration in the decision process.

- **Trust dependency on a third party**: Different messaging protocols can exhibit different degrees of trust dependency on a third-party authority (TPA). For example, a protocol may allow a TPA to have a key to an encrypted message and the message itself, thus trusting the TPA with the contents of the message (i.e., a high degree of trust dependency). Another protocol may use a TPA’s service to accomplish the message transfer, but does not allow the TPA to see the message contents. Such a protocol can be said to have a lesser degree of dependency on the TPA.

- **Existence dependency**: A protocol may produce a TPA’s signature on the delivery of critical information (i.e. decryption key), in which case the TPA plays an arbitrator role. Another protocol may produce enough digital evidence from both the sender and the receiver so that a subsequent dispute settlement does not depend on the existence or availability of the TPA. The choice is analogous to whether we keep a delivery receipt of a mail service provider (i.e. post office) or keep a receipt signature of mail recipients.

If we take the above three evaluation criteria into consideration, we may find that some existing protocols show some limitation. For example, the protocol proposed by Zhou [66] is biased to the message sender in that the message receiver has to keep on pulling for the encryption key from the third party until the sender posts the key. The protocol also has a high degree of trust dependency on the third party in the sense that the third party is entrusted with the encryption key. The third party can potentially use the key to decrypt the sensitive information transmitted in a message. Furthermore, the presence of the third party is required for dispute settlement even long after the
transaction has been committed. Ideally, at the end of a protocol, each party involved in a transaction should have a signature from each other instead of a delivery signature of a third party whose business may no longer exist at the time of dispute resolution. A non-repudiation protocol, that is fair to all parties, has the lesser degree of trust dependency on the third party and does not rely on the existence of the third party, is needed for collaborative e-business. In this work, we developed such a non-repudiation protocol.

5.2 Related Work

The existing studies on handling digital signature and evidence in electronic transactions have been reported in the context of the non-repudiation problem [65]. For different application areas (messaging systems, certified mail systems, electronic software distributions, payment systems, and so forth.), researchers have proposed different non-repudiation protocols. Here, we briefly review the ones that are closely related to ours.

In his book, Ford suggested the use of a trusted third party for non-repudiation service [70]. A service requestor S sends a request message to a service provider R through a third party authority (TPA). The TPA is responsible for the message transfer and the confirmation of its delivery. It becomes a witness in any future dispute. This approach greatly depends on TPA’s scalability. The TPA not only plays the role of the message deliverer but also as the witness who keeps track of all the transactions between S and R. It needs to maintain a large and secured database to record all the transactions and to play an arbitrator’s role in case of any dispute. Since all messages go through TPA, it may potentially become a performance bottleneck. A protocol must be designed so that it minimizes the involvement of TPA.
Zhou and Gollmann proposed a “Fair non-repudiation Protocol” [66]. The protocol is fair in the sense that the partial evidence generated during the execution of the protocol does not give any advantage to anyone. The sequence of actions is shown in Figure 5-1A.

![Diagram of TPA-based protocols](image)

**Figure 5-1.** Third Party Authority (TPA)-based protocols. A) Zhou’s, B) Abadi’s

In step 1, a message sender S creates a cipher text C by encrypting a plaintext M with an encryption key K. Then, it sends the ciphered text C to a recipient R with its digital signature. R, then, is supposed to acknowledge its receipt of the ciphered text C by returning a digital receipt to S in step 2. After receiving the receipt, S publishes the key K to TPA in step 3, where R retrieves the key in step 4 and S retrieves a confirmation ticket in step 5. The soundness of the protocol was discussed in terms of dispute resolution for each repudiation case. However, as pointed out in [67], the protocol has some drawbacks. First, it is advantageous to the sender because the successful execution of the protocol depends on whether the sender submits the key K to TPA as expected. The recipient has to keep on pulling to check if the key is available at TPA. In terms of the control over the commitment of a transaction, the protocol is not fair to message recipients. In Internet-based applications, especially e-commerce, we believe that the fairness with respect to the control over the commitment of a transaction needs to be considered. Second, the encryption key K is visible to TPA, thus, there is a risk of
violation of message security/privacy. Anyone who can access the key K at TPA can read the content of the message M.

Kim reported an extension of Zhou’s protocol to address the above two problems [67]. The sender sets the time limit t1 and included the information in step 1 of Figure 5-1A. The recipient also sets the time limit t2 (where, current time < t2 < t1) to let the sender know the deadline to submit the key. The protocol assumes a global time synchronization among senders, recipients, and TPA. In order to secretly transfer decryption keys, the protocol uses the Diff-Hellman algorithm. However, the extended protocol still requires the recipient to pull the decryption key from TPA until t2, which may incur several rounds of communication overhead. Furthermore, it needs the existence of the third-party authority for dispute resolution long after a transaction has been committed.

Abadi proposed another protocol shown in Figure 5-1B. The target application of the protocol is certified e-mail systems [71]. E-mail systems require sending messages in a send-and-forget manner. Moreover, mail senders need digital evidences of deliveries to prove that mails are actually delivered. The protocol was designed to meet these requirements. The protocol works in the following way. In step 1, the sender encrypts the message, encrypts the key with the Third Party Authority (TPA)’s public key, and sends them to the recipient. The recipient then forwards the encrypted key to TPA to retrieve the key in step 2. The TPA returns the key after decrypting the encrypted key with its private key in step 3 and sends a confirmation of the key delivery in step 4. This protocol has the following drawbacks. First, the protocol allows TPA to have access to the encryption key. It assumes that TPA is totally trustworthy and will not intentionally
violate the privacy policy. The protocol has a high degree of trust dependency on TPA. Second, from the non-repudiation perspective, the protocol is not secure because there is no evidence exchanged except the receipt of key delivery from TPA. The sender can repudiate the sending of a message because the protocol does not require the sender to write his signature. And, TPA’s confirmation of the key delivery cannot be accepted as proof of a recipient’s receipt of the message because the sender can intentionally send an encrypted key that cannot decrypt the message. We argue that TPA’s confirmation of key delivery is not equal to the evidence of message delivery.

Ray proposed a non-repudiation protocol that does not use TPA, avoiding the possible single-point-of-failure and availability issues [69]. However, the e-applications can have any number of TPAs. Replication techniques (that is, transparent request distribution and policy-based server selection) introduced in [72] can be used to replicate TPA’s services in the e-commerce environment. Also, communication between collaborating organizations may go through multiple intermediaries rather than direct communication between message senders and recipients.

5.3 Non-repudiation Protocol Requirements

As with other protocols [66, 72], we assume that the communication channel between parties involved in message transfer is reliable (that is, messages will not be lost). In addition, we assume that there is no single-point-of-failure or the availability issue with respect to the service provided by TPA, possibly using replication techniques.

Based on these assumptions, which eliminate the problems in executing the protocol correctly, we identify the following requirements regarding non-repudiation in e-commerce. We will show that our protocol satisfies these requirements in Section 5.6.
The protocol must protect both parties (that is, the sender and the recipient) from security threats such as message interception, modification, and replay attacks. This principle could be easily compromised in collaborative e-business because the communication channel may go through multiple intermediaries rather than through direct communication.

The protocol must ensure the confidentiality of transactions so that except the intended receiver, no one else including the third party authority (TPA) involved in the protocol is able to see any part of the transmitted messages. Although TPA collects transactional evidence for settling future disputes, it should not misuse its authority to monitor and collect transactional details.

The protocol must prevent the message recipient from reading the content of message until he has confirmed that the message has been received correctly.

The protocol must prevent the message sender from sending an invalid message or denying the sending of a message. The protocol should require the digital signature of the message sender not only for message authentication but also for message integrity.

The protocol must ensure that no communicating party can gain any advantage for having some partial evidence. The result of the protocol should be one of the following two: 1) the recipient having obtained the message with the sender’s signature and the sender having obtained digital evidence; 2) neither of them having obtained any useful information.

The settlement of a dispute for a committed transaction should be based solely on the digital signatures of transaction parties. For a committed transaction, the involved parties should not have to rely on the existence of a third party for dispute settlement because the third party’s business may be transient. The third party’s responsibility should be limited to facilitating a fair transaction to take place but should not have any further responsibility after the transaction commitment.

The protocol should be able to satisfy all the above requirements without causing too much overhead with respect to the number of communication channels needed, transaction delay and scalability.

5.4 Background

In this section, we will briefly go over the cryptographic tools we used in designing our protocol. Although this discussion is basic to cryptography researchers, without a basic knowledge of these tools, it is very hard to convince readers of how the protocol works. We shall therefore summarize them before describing our protocol.
5.4.1 Public Key Crypto Systems

In a public key or asymmetric encryption system, each entity K has a pair of keys (Pₖ, Sₖ), a public key and a private key [73]. The Pₖ is called the public key because it is published and used by others. The system is called “asymmetric” because different keys are used for encryption and decryption. Each key does only half of the encryption and decryption process. The keys operate as inverses, meaning that one key undoes the encryption provided by the other key. To support this asymmetric property, the system needs a special pair of mathematic algorithms: an encryption algorithm E and a decryption algorithm D, which are known to all collaborating parties. The RSA algorithm is one of them. The eclipse curve algorithm has gotten recent attention because of its cryptographic operation speed.

Using the asymmetric property, entities in a public key crypto system can exchange encrypted documents and signatures. For example, when Alice wants to send a secret message m to Bob, she computes a ciphertext c = E(P_{Bob}, m) and sends c. Since Bob alone knows S_{Bob}, he can read m by computing m = D(S_{Bob},c). No one else can read m. In case Bob wants to verify that a message m really comes from Alice, he may ask her a digital signature. She can do this by computing s = E(S_{Alice}, m). Note that Alice’s private key is used to generate a signature s. Bob can then check the origin of the message m by computing m’ = D(P_{Alice}, s) and checking m = m’. Actually, the implementation of message encryption and digital signature generation may employ a hash function to reduce the computational cost on encryption.

5.4.2 Message Digest

A Message Digest (MD) of a plaintext m is a fix-length (for example, 128 bits) data produced by using a one-way hash function, which takes the message m as the input [74].
It is significant that the hash function has the property of being one-way. From the message digest, no one can restore the original plaintext. Furthermore, no two different plaintexts can produce the same message digest. In secured communication protocols, the message digest is used as a basic tool for verifying the integrity of a received message. The sender attaches the message digest to the message. Then the recipient calculates the message digest of the received message. If two digest values match, then the recipient can be sure that the message has not been altered during the transmission.

In our protocol, the message digest function is used for checking the integrity of messages to be exchanged. We also take advantage of the one-way property of the message digest to hide the details of a message interchange. We design the protocol in such a way that the message digest is enough for dispute resolution. A third party involved in the protocol is able to access message digests but is not able to determine the original message content.

5.4.3 Dual Signature

The dual signature is a verification technique used in the Secure Electronic Transaction (SET) to link a purchase order and the purchase authorization with a credit card [74]. In the SET protocol, a purchase order message from a customer to a merchant consists of two parts: 1) the main content containing the details of the purchase order, and 2) the authorization code containing the card number of the customer. The latter is usually sealed to protect the customer’s credit card number from the merchant. The merchant then gets the main content of the purchase, whereas the credit card service provider receives the authorization code. The protocol needs a way to prove that these two parts (the purchase order and the authorization code) are actually linked for the settlement of possible future disputes. For instance, the authorization code used to
purchase product M should not be misused to authorize for purchasing product N. A dual signature is a customer’s signature on the concatenation of these two parts to prevent them from being used separately.

We use the same idea to make a link between an encrypted message and a sealed decrypting key. The message sender certifies the linkage by providing the dual signature to the recipient. Our protocol uses the dual signature technique for the following three purposes. First, the recipient can use the signature to check the integrity of the received message because it contains the message digests of both the message content and the key information. Second, it is the sender’s certification about the linkage between the encrypted contents and the secret key information. This is needed to prevent the sender’s misbehavior. The sender cannot send the incorrect decryption key information because it will not match with the dual signature. Third, the dual signature also prevents the recipient’s misbehavior. The recipient cannot generate the sender’s dual signature that links the key and the message. The recipient therefore cannot claim that a key provided by the sender cannot decrypt a message by swapping the key information in two transactions from the same sender. For instance, if the sender sends two transactions, \( t_1 (m_1, k_1) \) and \( t_2 (m_2, k_2) \), without the technical support of dual signatures, the recipient can say that \( m_1 \) cannot be decrypted with \( k_2 \) (Here \( t_1 (m_1, k_1) \) stands for the transaction \( t_1 \) containing the encrypted message \( m_1 \) and the decryption key \( k_1 \)).

5.4.4 Notation

The following notation adopted from Zhou’s paper [66] will be used in the remaining part of this chapter to present our non-repudiation protocol.
In our discussion, the term “encrypted key” is used to mean a secret key that is encrypted with the message recipient’s public key. The sender does this encryption to make sure that only the recipient can use the key. The recipient will decrypt the encrypted key using its private key and decrypt the content of a message using the secret key. We also use the term “double-encrypted key” to mean a twice-encrypted secret key that is encrypted with the recipient’s public key first and then with the public key of a third party authority (TPA) involved in the protocol. The sender creates the double-encrypted key to ensure that if and only if the recipient performs an obligation, he is entitled to access the secret key. The TPA will be responsible for monitoring the fulfillment of the recipient’s obligation (in other words, collecting the recipients’ signatures).

### 5.5 Secure Message Protocol for E-commerce

In this section, we explain our approach to address the requirements identified in Section 5.1. Figure 5-2 gives a high-level sketch of the new non-repudiation protocol without going into details. To simplify the figure, we omit the transaction ID, and message type $i$ means the contents of the message exchanged in step $i$.

In step 1, the sender generates a secret key randomly and uses it to encrypt the message. It then double-encrypts the secret key (dek: encrypted with the recipient’s public key and then with the third party authority’s public key). The secret key is encrypted twice because the sender depends on the third party authority to check the key releasing policy, however, the sender does not want the authority to access the key. The
dual signature is also created by concatenating the message digest of the ciphered text (em: the encrypted content), the message digest of the double-encrypted secret key, and the sender’s signature on these two message digests. All this information is sent to the recipient.

Figure 5-2. Secure message transfer protocol for e-commerce

When receiving the message of step 1 (that is, \( \text{tid} \parallel S \parallel \text{em} \parallel \text{dek} \parallel \text{dual_signature} \)), the recipient checks the integrity of both the encrypted main content \( \text{em} \) and the double-encrypted key \( \text{dek} \) by comparing them with the dual signature. Note that...
only when the integrity is preserved, the recipient initiates the next step. The progress to step 2 implies the recipient’s confirmation of receiving both the encrypted content and the double-encrypted key correctly. Thus, the recipient cannot claim later that he had received the wrong encrypted message content.

In step 2, the recipient forwards the double-encrypted key to the third party authority (TPA), along with its signature to acknowledge the correct receipt of the message content. The recipient is required to send his digital signature on the cipher text \( em \) in order to have access to the key. The recipient’s signature provides significant digital evidence that the recipient had attempted to access the secret key. The TPA will store the signature temporarily for dispute resolution and for signature distribution at the end of the protocol. Note that the recipient cannot write a signature on a cipher text \( em' \) (where \( em \) is actually what the sender had sent and \( em' \) is not equal to \( em \)) because he/she cannot construct the sender’s dual signature that contains \( em' \) which is needed if there is a lawsuit.

In step 3, the Third Party Authority (TPA) sends a ‘prepare_commit’ command, asking the recipient to commit to the current transaction of the protocol and return a signature. The TPA does not release the encrypted key at this stage because the recipient can deny receiving the key if the TPA does so. To prevent this case, we apply the two-phase protocol (2PC) to get a commitment from the recipient before releasing the key.

In step 4, the recipient generates a signature on the ‘prepare_commit’ command and returns the signature. After this step, the recipient will be entitled to get access to the key.
In step 5, TPA decrypts the double-encrypted key and releases the encrypted key to the recipient. Note that TPA is still unable to access the secret key because it is still sealed by the recipient’s public key. Only the recipient can access the secret key (wrapped inside the encrypted key). In case the key delivery fails due to a communication error, TPA will make it available to the recipient so that he can pick it up at anytime.

Lastly, the protocol ends with TPA forwarding two signatures at step 2 and 4 from the recipient. These two signatures represent the recipient’s receipts of the encrypted ciphertext and the commitment to getting the secret key, respectively. The TPA collects and forwards these signatures so that the sender does not need the existence of TPA after the transaction is completed.

5.6 Analysis

In this section, we give an informal analysis on how our protocol satisfies the requirements identified in Section 5.3. By this analysis, we want to clarify the implicit logic and the resolution scheme, which was not described in Section 5.5.

**Requirement 1:** *The protocol protects the involved parties from well-known message security threats such as message interception and modification and replay attacks.*

**Argument:** To protect from message interception and modification, we use message digest and encryption techniques. The integrity of the message can be checked with the message digest value and the confidentiality of the message is protected through encryption. No one but the recipient can read the message content. To protect from replay attacks, the protocol generates a fresh transaction id (TID) every time.
**Requirement 2:** The protocol ensures the confidentiality of transactions so that, except the recipient, no one else including the third party authority (TPA) involved in the protocol is able to understand the contents of a transmitted message.

**Argument:** The only way to understand the message between the sender and the recipient is through the secret key that encrypts the message. The secret key is encrypted twice to prevent the third party authority (TPA) and other intermediaries from getting access to the key. And, in step 4, the recipient signs on the message digest of the secret key, but not on the secret key itself. Thus, TPA does not have access to the key, even though he facilitates the key exchange. Note that a message digest is one-way so that it is impossible to reconstruct the original content from a message digest.

**Requirement 3:** The protocol must prevent the message recipient from reading the content of a message until he/she has confirmed that the message has been received correctly.

**Argument:** Our protocol allows the recipient to read the entire message only after he has returned the signature that he has received the encrypted message and he has committed to the transaction. Thus, the recipient cannot read the message without giving these two signatures

**Requirement 4:** The protocol prevents the message sender from sending an invalid message or denying sending a message.

**Argument:** The sender can obtain a receipt only after step 5. However, step 5 cannot be reached if the sender A has sent an invalid message. Recipient B would not give the first signature at step 2 if he did not receive the encrypted message correctly. Recipient B can check this with the sender’s dual signature and can also prove the
sender’s cheating (i.e. sending a wrong key) if he cannot read the encrypted message with
the key received from TPA. In court, recipient B can demonstrate his position by showing
that key K cannot decrypt the message and key K corresponds to the key part of the dual
signature received in step 1.

Sender A cannot deny having sent a message M (containing \textit{em}, \textit{dek}, and \textit{dual
signature}) because of the dual signature. It is only the sender who can generate the
signature. If the sender denies having sent either \textit{em} or \textit{dek} to recipient B and claims
having sent a different message \textit{em’} (where \textit{em’} is not equal to \textit{em}) or \textit{dek’} (again, \textit{dek’} is
not equal to \textit{dek}), recipient B can refute that claim by showing the sender’s dual signature
on \textit{em} and \textit{dek} that has been received.

\textbf{Requirement 5:} \textit{The protocol must ensure that no communicating party can gain
any advantage for having some partial evidence.}

\textbf{Argument:} If the protocol ends at step 1, even if recipient B has the sender’s dual
signature, the recipient cannot take any advantage because he/she has no way to access
the message content. If it ends at step 2 or step 3, sender A cannot claim anything
because recipient B has yet to sign the commitment of the transaction. If it ends in step 4,
the recipient still can retrieve the signature from TPA and read the message. If it ends
right before step 5, the sender also can retrieve the recipient’s signatures from TPA.

\textbf{Requirement 6:} \textit{Any dispute for a committed transaction must be resolved solely
based on the digital signatures of transaction participants. For a committed transaction,
both parties should not rely on the existence of a third party for dispute resolution.}

\textbf{Argument:} At the end of the protocol, the recipient ends up having the sender’s
dual signature and the sender having the recipient’s signature. Thus, they do not need the
third party’s presence in court. Signatures of both parties are enough to resolve any dispute.

**Requirement 7:** The protocol should be able to satisfy the previous requirements without causing too much overhead with respect to the number of communication channels needed, transaction delay, and scalability.

**Argument:** Our protocol requires six message exchanges, which is one more than Zhou’s protocol. This can be justifiable because our protocol aims at the lesser degree of trust dependency on the part of the third party and does not rely on the existence of the third party to settle disputes. Our protocol exchanges signatures between the sender and the receiver, instead of the TPA’s signature on the delivery of the key. In terms of transaction delay, our protocol does not require any transaction delay. The message recipient (in most cases, service providers) can retrieve the key in step 5, without having to wait for the sender to push the key to the TPA, which is the case in Zhou’s protocol.

From the scalability perspective, in order to avoid the bottleneck problem when using TPA, we propose to replicate the TPA’s services. The same replication approach can also be used to implement Zhou’s protocol to achieve scalability.
CHAPTER 6
ARCHITECTURE AND IMPLEMENTATION TECHNIQUE

Based on the research results presented in the previous chapters, we have designed and prototyped a distributed network architecture and its security software components needed for trust-based security management. We have also investigated a specification-driven approach for system implementation. This chapter describes the network architecture and the specification-driven approach to enforce trust agreements.

6.1 Distributed Network Architecture for Trusted Collaborative Computing

The overall network architecture for a collaborative system is shown in Figure 6-1. We envision that the architecture consists of a network of Trusted Collaboration (TC) nodes, which interact as peers in the network. A TC node is a set of hardware and software under the administration and control of an organization. Physically, a TC node is protected by using advanced router and firewall technologies, which mediate and control the traffic flow into and out of the TC node. It enforces the security policies and constraints that are consistent with the security objectives and requirements of an organization. It also achieves secured sharing of its protected resources based on its established trust relationships with the TC nodes of its collaborating partners. Each Trusted Collaboration (TC) node is capable of establishing trust and contractual relationships with others without resorting to a centralized controller. A TC node keeps a list of all TC nodes with which it establishes trust relationships and the terms and conditions of collaboration. This trust information will be used to make authentication and authorization decisions for service requests. A user in a TC node can have access to
the protected resources in another TC node, possibly through multiple intermediary TC nodes. Similarly, collaborating organizations’ applications and software systems (the clients), which are connected to these TC nodes through service adapters, are allowed to access collaborating organizations’ resources.

![Network architecture of a collaborative information system](image)

Figure 6-1. Network architecture of a collaborative information system

Inside a Trusted Collaboration (TC) node, there are a number of servers that provide various services for supporting collaborative e-business (e.g., negotiation services, workflow management services, brokering services, security services, etc). The servers in a TC node can be replicated and installed at many sites in the Internet just like replicated Web servers to achieve scalability, reliability and expandability. Among these servers, the trust-based security server, which is responsible for security and trust management, is the focus of this dissertation.

Figure 6-2 shows how the trust-based security server is different from the traditional security server. The main difference is that a trust agreement made between TC nodes will be taken into account in performing security functions. The server is responsible for 1) authenticating the service requestor’s credentials according to the
agreement, 2) evaluating the trustworthiness of the requestor based on authenticated credentials, 3) evaluating the trustworthiness of a transaction based on local security policies and contextual environment (such as network location, connection time, separation of duty, etc), and 4) finally granting the proper level of role privileges.

Figure 6-2. Trust-based security enforcement

Note that trustworthiness of a transaction is evaluated against local security rules and the contextual environment before the transaction is being authorized. Most organizations have their own policies for security and privacy, independent of any collaboration effort. These rules are defined to guard against any possible risk associated with transactions. They need to be checked and evaluated against the contextual environment of the network that provides run-time states and/or values. The contextual environment includes temporal context (e.g., user session and time), computing context (e.g., protected resource status, network connectivity, and availability of secure channel), access history context (auditing data), and exceptional events. For example, assuming
that there is a risk of a multiple role-playing employee being engaged in some unlawful actions (e.g., creating a bank check statement and clearing the check), a “separation of duty” policy can be designed and implemented into the security system. Another example is a policy for the protection of privacy from incremental access. Let us assume that a single datum may not reveal the protect information, but there is a risk that a set of data together may reveals the sensitive information just like the clues to a mystery. Even though the service requestor is trusted, the transaction is checked against the privacy protection rule to evaluate the associated risk and trustworthiness. Only those transactions that are not violating any local security rule will be considered as trustworthy.

6.2 Overview of the Software Architecture

We have designed the software architecture of a trust-based security server, which has been briefly described in the previous section. The security server takes high-level trust agreement specifications, integrates them with local security policies and finally translates them into events, action-oriented rules, and triggers. The security server, replicated at each organization’s gateway, also enforces inter- and intra-organizational security policies and constraints by making use of the ETR technology [46]. The server consists of two parts: a specification-time architecture and an enforcement-time architecture, as shown in Figure 6-3.

The specification-time architecture, shown in the upper part of Figure 6-3, contains a set of visual tools and a deployment tool. Collaborating organizations, through negotiations or other means, come to an agreement on inter-organizational (global) security policies and constraints. The resulting agreement is signed and distributed (in an XML document) to the servers of the collaborating organizations. A tool is provided to
aid the specification and distribution of a trust agreement, as shown on the top left side of Figure 6-3.

Figure 6-3. Software architecture of a Trust-based security server

Apart from global policies, local security policies and constraints are also specified, as shown on the upper right side of Figure 6-3. We separate the tool for the local security specification from the tool for the trust agreement specification in order to stress our point that the former is a joint effort of collaborating organizations and the latter is the task of individual organizations. The translation, verification and deployment tool then takes both the trust agreement specification and the local security specification, verifies the policy consistency, and translates them into security configuration, events, and condition-action rules. The verification of policy consistency between inter-organization
security policies and organizational security policies is important, but is beyond the scope of this dissertation. Research into this issue is part of our future work.

The enforcement-time architecture, shown in the lower oval of Figure 6-3, enforces security rules and constraints during the processing of service requests and replies. The architecture consists of software components that implement the protection mechanisms, such as certificate-based authentication, role-based access control, and constraint checking. To meet the dynamic, adaptive, and rapid re-configuration security requirement (i.e., due to the contract revision and annulment or revocation of authority), it takes advantage of an implemented Event-Trigger-Rule Server [46], which is not shown in Figure 6-3, as the underlying mechanism to enforce the trust and security rules and policies. The rules and configuration data are generated based on an inter-organizational trust agreement specification. The Event-Trigger-Rule Server uses these rules and data to enforce the trust and security policies and constraints specified in the trust agreement.

The software architecture outlined previously has the following advantages. First, inter-organizational security policies and constraints can be specified using a high-level specification language (i.e., the trust agreement specification) or a GUI tool, instead of being hard-coded in applications. This facilitates the design and modification of inter-organizational security policies and constraints; it is easy to understand and make changes to security rules if the policies and constraints are specified in a high-level specification language. Changes made to policies and constraints due to the dynamic nature of a virtual community can be made and redeployed quickly with our approach. Second, we provide a mechanism that generates executable rules and data from specification documents to quickly deploy the policies and constraints. By generating
events, condition-action rules and triggers, and installing them in replicas of the Event Server and the ETR Rule Server from high-level specifications of trust and security policies and constraints, our approach can invoke multiple security rules that suit a particular computing environment. This enables distributed and flexible deployment of trust agreements. Third, the event-driven and rule-based enforcement of policies and constraints allows the integration of loosely coupled systems and the formation of a secured virtual community. Data relevant to security (e.g., update of certificate revocation list, modification to a trust agreement, recommendation about trustworthiness of a new Certificate Authority (CA) or existing CA, etc.) can be exchanged through the event notification mechanism and be used to coordinate the activities of the components within a TS server and the components of its replicas across the Internet.

To summarize, the three key features of a proposed software architecture are: 1) the provision of high-level tools for security and trust specifications; 2) the specification-driven approach (that is, generating data, code and rules automatically from high-level specifications of the security policies and constraints); and 3) the event-driven, rule-based enforcement of security constraints to support the dynamic, adaptive, and rapid deployment of trust and security management in a collaborative computing environment. Next section 6.3 covers these details.

### 6.3 Implementation Details

The trust-based security model, the non-repudiation protocol and the software architecture proposed in the work are very general. They can be applied to different collaborative computing environments. Since Web service technology has drawn much attention recently, we have implemented the trust-based security network software
architecture and the non-repudiation message transfer protocol in the Web service platform.

![Diagram of Web service interactions]

**Figure 6-4. General Web service**

Web service technology provides a systematic and standard-based approach (e.g., UDDI, SOAP, WSDL, WSFL) to enable application-to-application integration [4, 5]. It provides basic building blocks for collaborative computing. Figure 6-4 shows the general Web service model [75], which shows the interactions among three roles: Service Provider, Service Registry, and Service Requestor. In the publish-phase of the model, a service provider, which represents an organization that provides its resources as Web services, describes its services using WSDL (Web Service Definition Language) and publishes the services to a service registry using UDDI (Universal Description, Discovery and Integration). In the discover-phase, a service requestor, also using UDDI and WSDL, queries the registry to find the required service and to obtain the information required to contact the service provider. In the bind-phase, the service requestor contacts a service provider to dynamically bind and invoke a Web service application by sending a SOAP (Simple Object Access Protocol) message via HTTP.
We have implemented a set of graphical user interfaces and a deployment tool that are running as a Web application. They are a Trust Agreement specification tool, a RBAC specification Tool, and a Deployment Tool. These tools help a policy maker define a set of trust policies in a high-level specification, and generate security metadata (mapping and event definitions in our case) and executable rules from the specification. We have also implemented a run-time enforcement engine that enforces security policies and constraints using the generated data and rules. The engine is an extension of a Web server. Furthermore, we have implemented the non-repudiation message transfer protocol running in the Web service environment. The protocol takes any message from applications, generates encrypted/signed SOAP messages, and requires generating the recipient’s signature. This section describes the details of each component in turn.

### 6.3.1 Trust Agreement Specification Tool

Typically, a trust agreement specification document goes through the following life cycle. At the beginning, the document instance is created and then edited. At some point, it is saved. The saved specification document may be transferred to another network node to be reviewed. If the specification is rejected, it goes back to the “edit” state. If it is accepted, then the specification is deployed. Eventually the document becomes invalid when its valid period has expired. The design of the Trust Agreement Specification Tool is based on the life cycle. It consists of the GUI, a communication interface, a persistence manager, an editing component, and a deployment interface, as shown in Figure 6-5.

The trust agreement specification GUI is used by trust policy makers or negotiators to input and edit specification documents. To make the ubiquitous use of the tool possible, the GUI is implemented as a Web application using the JSP technology [76],
interacting with the other internal components for persistence, editing, and deployment of the document.

![Diagram of Trust Agreement Specification Tool and Deployment Tool](image)

Figure 6-5. Trust agreement specification tool

The communication interface is used for receiving trust agreements in the form of XML documents, through the Internet using a message transfer protocol (for example, our protocol described in Section 6.3.4). Once received securely and checked, the specification document is passed to the persistent manager.

The persistent manager is responsible for storing and retrieving trust agreement specifications. It is responsible for constructing the specification document object from an XML file. It also translates a specification document object into an XML file for storage.

Another internal component of the tool is the deployment interface. Once trust policy makers decide to accept a trust agreement specification, they use this interface to invoke the deployment tool. The deployment tool then invokes the TSM mapping
generator and Event-Triggering-Rule (ETR) rule generator to translate the specification document into mapping data, events, rules, and some metadata.

The top menu of the specification GUI gives four initial choices for creating and editing trust agreement specifications: 1) add a new trust agreement; 2) browse trust agreements that are saved but have not been deployed yet; 3) browse trust agreements that are received but have not been deployed yet; and 4) list deployed trust agreements. The user chooses the first option to instantiate a trust agreement specification. It will lead to the input dialog to receive the unique identifier of the specification from the user. After that, the GUI leads to an editing mode.

Figure 6-6. Review of a trust agreement specification using the tool

If the user chooses either the second or third option, the GUI displays the list of trust agreement specifications, which are distinguishable by their identifiers. The GUI uses the JSP template to generate dynamic HTMLs for both received specifications and locally saved specifications. As mentioned before, a specification can be instantiated
locally, and is being edited and saved. It could also be edited and received from another collaboration node. Regardless of its origin (either received or locally saved), every specification document is in XML and is managed by the persistent manager. The GUI retrieves a trust agreement specification document through the manager, generate a dynamic html, and display it as shown in Figure 6-6.

Notice that at the bottom of the screenshot of Figure 6-6, it shows two hyper links, each of which represents an operation (either edit or deploy) on the currently chosen specification. If the trust agreement displayed is received from another node, the user may get into the editing mode by clicking the ‘edit this trust agreement’ hyperlink, which will retrieve the specification document from the persistent manager and display it as shown in Figure 6-7. At the end of the editing process, the user will get a specification document in XML and return it to the sending TC node as a reply. If the user decides to deploy the trust agreement just reviewed, he/she can click the ‘deploy this trust agreement’ hyperlink. The same interface is used to access locally created specifications. When the tool is used in an editing mode, the left side of the UI shows a tree structure of a trust agreement specification document. The top tree structure of UI menu is organized into ‘Parties’ and ‘Trust policies’. It is equivalent to the specification language structure we have described in Chapter 4. When the user expands the second level of the tree, it shows the sub-category that contains the list of either the parties or the trust polices within the specification. Figure 6-7 shows a role authorization policy defined in the trust agreement specification ‘tsm01’. The user can add additional role authorization policies by choosing the pull-down menu located at the upper right corner of the UI.
Figure 6-7. Editing screen shot of the trust agreement specification tool

Figure 6-8. Editing a role authorization policy using the specification GUI
To view the details of a party or a policy, the user can choose the “Edit” hyperlink. This returns the detailed information of the corresponding entity. For example, in Figure 6-7, when the user clicks the ‘Edit’ hyperlink in the list, a HTML page shown in Figure 6-8 is generated and displayed. The page contains the detailed policy information of ‘authorization1’ in a tabular format. The conditional statement defines a constraint, which states that the requestor who holds the membership ‘manager’ must have a trust level greater than 0.7. The syntax used for specifying constraints follows the syntax of the condition statement used in the ETR rule specification [46]. The condition statement is a Java statement whose logical expression contains logical AND and OR operators instead of ‘&& and ‘||’ used in Java.

### 6.3.2 RBAC Specification Tool

The RBAC specification tool is another GUI tool we developed for defining local security rules in terms of role policy. It is used to define role objects and populate them. The tool supports basic RBAC specification activities, such as defining the managed resources and their exposed operations, a set of permissions based on those resource definitions, and a set of roles as a collection of privileges, and specifying a role hierarchy to represent a parent-child relationship among roles. The RBAC specification tool has an editing component like the trust agreement specification tool. However, since the tool will be used locally to define local security rules in terms of roles, we do not need to generate an XML document for persistence and exchange purpose. Unlike the trust agreement specification document, which is converted into an XML document, role objects are stored in a database management system, which is a Persistent Object Manager library (POM) that we developed in our previous project.
Figure 6-9. Role based access control specification GUI.

6.3.3 Run-time Enforcement Engine

The run-time enforcement engine (that is, a set of components in the enforcement-time architecture shown in Figure 6-3) was implemented as an extension of a Web server, shown as a security server in Figure 6-10. It is a plug-in component to a Web server. For our prototype implementation, we integrate the engine with the Apache Tomcat Web server. Basically, the server takes security mapping data, events, rules, and metadata that are generated by the specification-time components and enforce agreed security policies accordingly. To simplify the figure, we represent this relationship as an arrow between ‘Deployment interface’ and the generated mapping data and event-triggering rules.
When a secured connection is established at the transport layer using SSL/TLS between a Web service requestor and the enhanced Web server we developed, the server creates a pair of request and response flows. The components of the server (authenticator, authorizer, and security constraint enforcer) use the pair to check certificates, constraints, and perform mappings between trust agreements and apply local security rules. We will describe each component in turn. But, before we do that, we shall first explain the ETR technology and its relationship with the security server.

The Event server and the ETR server have been developed in our previous project [46] to implement rule processing in the Event-Trigger-Rule (ETR) paradigm. The ETR paradigm is a generalization of the Event-Condition-Action (ECA) paradigm. Unlike the ECA paradigm, the ETR paradigm separates event and rule specifications and uses trigger specifications to relate events with rule structures. Events can be “triggering
events” or events that participate in a composite event expression. Triggers are specifications that relate events with rule structures, making it possible to fire structured rules upon the occurrences of events. When a triggering event occurs, the corresponding triggers are activated for processing. During the processing of a trigger, the event history (or a composite event) is evaluated. If it evaluates to be “true,” then the corresponding rules are fired. Each rule represents a small granule of logic. A structure of rules explicitly specifies a large granule of control and logic that can be used to enforce some security constraints. Also, a single rule can participate in multiple rule structures, thus making each rule reusable in building a larger granule of control and logic that specifies a security policy.

We integrate the security server and the ETR server in a loosely coupled manner. They communicate through events, depicted as an arrow between them in Figure 6-10. In other words, the authorization component and the authentication component in Figure 6-10 can generate and post events to the ETR server to check conditional statements. For example, let us assume that a trust agreement specification has a policy, which allows several memberships (including membership “m”) to acquire a role “r” on a resource “rs”. However, as an exception for this month, requestors who hold “m” can do so only during the working hours on weekdays. In this case, the acquiring of role “r” on resource “rs” is defined as an event. When that event occurs in the authorization component, a rule is triggered to check the stated security constraint on role “r”. We implemented this functionality and tested with the Tomcat Web server. At run-time, the engine makes a reference to the generated metadata to find out what type of event (in this case, acquiring the role “r”) should be posted to trigger the constraint evaluation. Then it creates an event
object from the current Web session with the requestor and posts it to the ETR server. The ETR server then triggers the rule that was translated and generated by the deployment tool from the conditional statement of the agreement.

Events can also be received from outside of a Trusted Collaboration (TC) node to trigger local rules. For example, suppose a user in organization A (who has been working on a collaborative project using the resources provided by organization B) is transferred. His/her privilege of access to the resources must be invalidated in a timely manner. With the revocation of his/her certificate captured as an event, the system will be able to notify the relevant TC nodes of the change and trigger the rules to revoke the access rights.

Generally speaking, anything that is of interest in a collaborative environment can be captured as an event and used to trigger rules to enforce security policies and constraints, regardless whether they are locally or globally defined. Also, not shown in Figure 6-10, the posting of an event may trigger the processing of distributed rules if multiple rules are tied to the same event as specified in multiple trigger specifications. An event notification mechanism provided by the Event server would send a notification to its replicas at other sites, which would activate their corresponding ETR servers to process the triggers and rules installed at these sites. Trust and security management can then be carried out in a distributed manner by replicas of peer-to-peer servers in the proposed network architecture.

Let us continue our discussion on each component of the server. The authenticator is responsible for authenticating service requestors. The authenticator exchanges public key certificates (and some additional attribute certificates) with requestors, verifies the attributes of the certificates and determines the requestors’ membership from the verified
attributes. In addition, it posts events to the ETR server to trigger constraint evaluation using the security constraint enforcer. The authenticator makes use of X509 v3 technology and SSL/TLS to exchange public key certificates at the transport layer. For the prototype implementation, several certificate authorities were set up and they were used to create several public key certificates for both the organizations and their employees in our scenario. Some users’ certificates may have the value for the “Alternative SubjectName” field in certificates for access control.

We also made use of an HTTP’s header and SPKI to include requestors’ additional certificates [14, 77]. The “Authorization” requester header field is reserved for an Web user agent (typically Web browsers) to authenticate itself with an HTTP server [77, 78]. The field value consists of credentials containing HTTP requestors’ authentication information. For example, HTTP Basic Authentication is in the following format:

**Authorization: BASIC** ‘user:password’ in bas64

The credential part is encoded in base64, a text representation of an arbitrary object to be exchanged in the Internet. The bold letters are reserved keywords. In our prototype implementation, the attribute certificates in SPKI [14] can also be employed for authentication. The enhanced Web server can recognize the following format of HTTP authorization headers containing SPKI certificates encoded in base64:

**Authorization: SPKI** ‘SPKI certificate’ in bas64

The authenticator decodes the base64, reconstructs SPKI certificates, verifies certificates, and retrieves the certified attributes of requestors. Both X509-based and SPKI-based certificates can be used for authentication simultaneously or separately. The
enhanced Web server is able to recognize both types of certificates. To support this functionality, we extend the Axis SOAP toolkit so that the client side SOAP library checks environment variables, such as “KeyStore” for X509 certificates and “SPKI” for SPKI attribute certificates when constructing request connections, and includes attribute certificates in the headers, if necessary. This simple API allows a Web client program to select certificates at run-time and attach to its Web service requests. The server parses certificates to retrieve security attributes stored in the certificates. We have developed the certificate parser component using JavaCC, a Java version of a parser generator.

Once the authenticator identifies the requestor’s membership and his/her associated attributes, the authorizer determines a role that requestors will play based on the membership-role mapping data. It checks if the role has a permission to access and perform a requested operation on a requested resource (or a service in the context of Web services). If the authorizer receives a SOAP request message, it looks for the value of the “soapaction” HTTP header to determine what permission the requestor needs. The “soapaction” HTTP header is a required attribute of the binding elements in SOAP, if SOAP is bound using HTTP [5]. We make the corresponding extension to the Apache Axis SOAP toolkit. As with the authenticator, the authorizer may also post an event to evaluate constraints associated with the mapping.

It is the security constraint enforcer that is responsible for creating an event object, posting it, and returning a Boolean value. We use the Java reflection API to create an event object dynamically at run-time, instead of hard-coding the posting of pre-determined number of events. The data value of events comes from the requestors’
certificate, some pre-defined request object’s attributes (like IP of user agents, request time), and HTTP request header values.

Depending on the requestor’s interface (either Web browsers or client-side Web service applications), the requestor’s certificates may be loaded to request messages differently. If a requestor’s interface is a Web browser, then the browser will be prompted for X509 public key certificates. If the interface is a SOAP client program, the SOAP library we have extended will load certificates from a local file systems based on environment variables and add them to the request messages.

6.3.4 Protocol Implementation

We have implemented the protocol proposed in Chapter 5, using the Apache Axis SOAP toolkit. The implementation is in between the application and the SOAP message layers, as shown in Figure 6-11. The protocol implementation consists of the sender-side protocol handler, a receiver-side protocol handler, and a TPA-side protocol handler. The sender-side handler takes care of generating encrypted/signed SOAP messages and sends them out on behalf of sender applications. The receiver-side handler receives SOAP messages, generates the recipient-side signature, interacts with the TPA-side handler to get a secret key, and reconstructs the original document for receiver-side applications. Finally, the TPA-side protocol handler collects the necessary signatures for sender applications and authorizes the release of secret keys.

The process of the protocol is as follows. At first, a sender application invokes a sender-side protocol handler with a document as a parameter. The handler then applies the cryptographic operations on it, packages the result into a SOAP message, and sends the message to the receiver-side Web server. The receiver-side Web server employs the receiver-side handler to process the incoming SOAP message. The receiver-side handler
interacts with the TPA-side protocol handler, which is installed in a Third Party Authority (TPA) as a Web service. Once the receiver-side handler retrieves the secret key for decryption as a result of step 2, 3, 4, and 5 of our protocol, the receiver-side Web server is able to decrypt the SOAP message and forwards the original document to internal applications at the receiver’s network.

![Diagram showing the implementation of three protocol components](image)

Figure 6-11. Implementation of three protocol components

### 6.4 Experiment

We have experimented with our implementation (both security servers and the protocol components) with the following system configuration. We deployed a couple of demo Web service applications in the extended server (that is, the web server with our security components plugged-in). Then we used the organization’s RBAC specification tool and define roles that have permissions on the applications. Using the specification tool (i.e. TAST in Figure 6-12), a policy maker at a resource requestor organization is assumed to specify a trust agreement (step 1 shown in Figure 6-12). The tool generates an XML document. We employed our protocol to transfer the specification to the resource provider organization (step 2, 3, and 4). The protocol takes care of packaging the document into a signed/encrypted SOAP message (that is, between step 2 and step 3). The protocol makes use of the TPA’s protocol service to decrypt the secret key in
exchange (step 4). Once the document is received securely and document integrity is verified, it is passed to TAST (step 5). A security expert working for the provider organization will use the tool to review the document. He/she will invoke the deployment tool to generate mapping metadata, events, rules, and triggers (step 6, 7, and 8), if the specification is accepted.

Figure 6-12. Typical use of trust agreement specification tool

For run-time testing, we generated public key certificates and attribute certificates for collaborating organizations and users in our demo scenario. We used the OpenSSL toolkit and the Sun’s keytool utility (included in JDK 1.4) to generate X.509 public key certificates. We also used SPKI toolkit to generate attribute certificates for Web service client applications. To convert the X509 public key inside certificates to compatible formats so that they can be imported into the SPKI toolkit, we developed a conversion program as well.
Emerging technologies, such as Web service and grid service technologies, have enabled the development of Internet-based application areas such as e-business, e-government and virtual enterprise management. These application areas all involve a number of collaborating organizations sharing distributed and heterogeneous data, software and other resources over the Internet. As in all other distributed systems, security is a key requirement. However, Internet-based collaborating computing presents new challenges in terms of security and trust management. This is mainly because conventional security is intended for the centralized protection of resources in a client/server environment from malicious attacks, unauthorized access, and denial of services, while security in collaborative computing requires the additional establishment of trust relationship between collaborating parties. Research is needed to investigate how to establish trust policies governing message exchanges and resource sharing between collaborating organizations, and how to enforce them by making use of the existing software components.

This work has investigated the following four research issues. First, it has investigated the unique characteristics of collaborating computing that can be exploited for the potential security threats. Second, it has introduced the concept of trust agreement and developed a trust agreement specification language for establishing inter-organizational security policies and constraints. Trust agreement represents an agreement about the inter-relationships between trust concepts in the Internet environment (e.g.,
certificates, and certificate authorities) and the conventional security concepts (e.g., roles, permissions and etc.) in an organization’s security setting. It governs the message exchanges and resource sharing. Third, this work has presented the design and the implementation of the trust-based security server. We have demonstrated the “specification-driven” approach to trust and security management by developing an automatic deployment technique, which generates security mapping data as well as executable security constraints from a high-level trust agreement specification. Fourth, this work has identified additional security requirements for non-repudiation in collaborative computing, analyzed existing protocols, and developed a new non-repudiation messaging protocol. We have implemented the proposed protocol using a Web service toolkit and used it to transfer trust agreement specifications from one party to another.

For future work, we suggest the following research issues. We strongly believe that the research outcome in this work is closely related to other collaboration techniques such as e-contract, Workflow, and Service Level Agreement (SLA). Thus, future research will investigate the possibility of an automated collaboration design and code generation that integrate all these technologies. Another research issue arises from the fact that inter-organizational trust agreements may conflict with the existing organizational security policies and constraints. A formal study is needed to investigate what conflicting or inconsistent factors exist between inter-organizational trust policies and local security policies; also we will look into how to systemize the verification process.
APPENDIX A
TRUST AGREEMENT SPECIFICATION

The XML Document Type Definition (DTD) for trust agreement specification documents is as follows:

```
<!ELEMENT trustagrement (organizations, policies)>
<!ATTLIST trustagrement id CDATA #REQUIRED>
<!ELEMENT organizations (collaboratingParty+, ca+, tpa+)>
<!ELEMENT collaboratingParty (contact, wsdl, exportedRoles)>
<!ELEMENT contact (#PCDATA)>
<!ELEMENT wsdl (#PCDATA)>
<!ELEMENT exportedRoles (#PCDATA)>
<!ATTLIST collaboratingParty id CDATA #REQUIRED>
<!ELEMENT ca (publickey, revocationRepository)>
<!ELEMENT publickey (#PCDATA)>
<!ELEMENT revocationRepository (#PCDATA)>
<!ATTLIST ca id CDATA #REQUIRED>
<!ELEMENT tpa (wsdl, contact)>
<!ATTLIST tpa id CDATA #REQUIRED>
<!ELEMENT policies (membership+, delegation+, msderivation+, rolegrant+, non-repudiation?)>
<!ELEMENT membership (ca-list, attrs, condition)>
<!ELEMENT ca-list (#PCDATA)>
<!ELEMENT attrs (#PCDATA)>
<!ELEMENT condition (#PCDATA)>
<!ATTLIST membership pid CDATA #REQUIRED>
<!ELEMENT delegation (delegator, delegatee, authorities, furtherdelegationflag)>
<!ELEMENT delegator (#PCDATA)>
<!ELEMENT delegatee (#PCDATA)>
<!ELEMENT authorities (#PCDATA)>
<!ELEMENT furtherdelegationflag (number | "*")>
<!ELEMENT number (#PCDATA)>
<!ATTLIST delegation pid CDATA #REQUIRED>
<!ELEMENT msderivation (have, willhave)>
<!ELEMENT have (#PCDATA)>
<!ELEMENT willhave (#PCDATA)>
<!ATTLIST msderivation pid CDATA #REQUIRED>
<!ELEMENT rolegrant (ms-id, role-id, condition)>
<!ELEMENT ms-id (#PCDATA)>
<!ELEMENT role-id (#PCDATA)>
<!ATTLIST rolegrant pid CDATA #REQUIRED>
<!ELEMENT nonrepudiatiion (protocol, tpa-id)>
<!ELEMENT protocol (#PCDATA)>
<!ELEMENT tpa-id (#PCDATA)>
```
APPENDIX B
AN EXEMPLARY SPECIFICATION OF TRUST AGREEMENT

The following is the exemplary trust agreement document for our scenario.

<trustagreement id="example1">
<organizations>
  <collaborating Party id="ORG-S">
    <contact>http://www.org-s.com</contact>
    <wsdl>https://www.org-s.com/orderProcessing.wsdl</wsdl>
    <exportedRoles>Order-Requestor</exportedRoles>
  </collaborating Party>
  <ca id="Primary_CA">
    <public key>http://ca.virtual.com/pk</public key>
    <revocation_repository>http://ca.virtual.com/revoke/list</revocation_repository>
  </ca>
  <tpa id="ReceiptDistributor">
    <serviceDefinition>http://www.receipt.com/axis/non-repudiation.wsdl</serviceDefinition>
    <contact>http://www.receipt.com/axis/ReceiptDistributor</contact>
  </tpa>
</organizations>
<policies>
  <membership id="manager">
    <ca>Primary_CA</ca>
    <attributes>text:job_title, text:company_name, double:trust_level</attributes>
    <condition>this.organizations.contains(company_name) AND (job_title == 'manager')</condition>
  </membership>
  <delegation id="delegation_relationship1">
    <delegator>Federal_CA</delegator>
    <delegatee>Florida_CA</delegatee>
    <authorityTobeDelegated>manager</authorityTobeDelegated>
    <furtherDelegate>1</furtherDelegate>
  </delegation>
  <roleGrant id="Authorization1">
    <membershipID>manager</membershipID>
    <roleID>OrderRequestor</roleID>
    <condition>(manager. trust_level > 0.7)</condition>
  </roleGrant>
  <non-Repudiation id="Comm_Protocol">
    <protocol>UF_Non_repudiation</protocol>
    <tpa>ReceiptDistributor</tpa>
  </non-Repudiation>
</policies>
</trustagreement>
LIST OF REFERENCES


50. W3C XML Encryption WG, “XML Encryption,”
http://www.w3.org/Encryption/2001/, 2001, Accessed 03/05/01.


Distributed Environment,” IEEE Symposium on Security and Privacy, pp. 3-14,
May 1998.

Proceedings of the 33rd Annual Hawaii International Conference on System


Certificates for Widely Distributed Access Control,” In IEEE 7th International
Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises -

International Computer Software & Applications Conference (COMPSAC), Vienna

60. Yang, S., Lam, H., and Su, S. Y. W., “Trust-based Security Model and
Enforcement Mechanism for Web Service Technology,” The 3rd VLDB Workshop
on Technologies for E-Services (TES'02), Hong Kong, Aug. 23-24, 2002, pp151-
160.

61. Yang, S., Su, S. Y. W., and Lam, H., “A Trust-Based Security Architecture and
Model for Enabling Collaborative E-Business,” The 5th International Conference

62. Hildmann, T. and Barholdt, J., “Managing Trust between Collaborating Companies
using Outsourced Role Based Access Control,” Proceedings of the Fourth ACM
Workshop on Role-Based Access Control, October 28 - 29, 1999, Fairfax, VA,
USA, pp. 105-111.


BIOGRAPHICAL SKETCH

Seokwon Yang was born in Chunchon, Kangwon Province, Korea. He earned a bachelor degree in computer engineering in Computer Science and Engineering department from Hanyang University, Korea, in February 1997. He earned his master degree in computer science from the University of Florida, Gainesville, in August 1999 and will receive Ph.D in Computer Engineering in December 2003. His research interests include active, object-oriented databases, distributed systems, the Web-service technology, and Internet security.