A Dynamic Distributed Trust Model to Control Access to Resources Over the Internet

by

Hui Lei
B.Eng., ChengDu University of Science & Technology (now SiChuan University), 1991

A Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

in the Department of Computer Science

We accept this thesis as conforming
to the required standard

Supervisor: Dr. G.C. Shoja

Abstract

The access control mechanisms used in traditional security infrastructures, such as ACL and password applications, have been proven inadequate, inflexible, and difficult to apply in the Internet due to the incredible magnitude of today's Internet. Recently, research for expressing trust information in the digital world has been explored to be complementary to security mechanisms.

This thesis deals with the access control for the resources provided over the Internet. On line digital content service is exemplary of such an application. In this work, we have concentrated on the idea of a trust management system, which was first proposed by Blaze *et al* in 1996, and we have proposed a general-purpose, application-independent Dynamic Distributed Trust Model (DDTM).

In our DDTM, access rights are directly associated with a trust value. The trust values in this thesis are further classified into direct trust values, indirect trust values and trust authorization levels. We have calculated and expressed each type of the trust values as explicit numerical values.

The core of this model is the recommendation-based trust model, organized as a Trust Delegation Tree (TDT), and the authorization delegation realized by delegation certificate chains. Moreover, the DDTM provides a distributed key-oriented certificate-issuing mechanism with no centralized global authority.

A Dynamic Distributed Trust Protocol (DDTP) was developed as a general protocol for establishing and managing the trust relationship in a TDT structure. The protocol was verified by means of the verification tool, SPIN, and was prototyped to simulate communication and behaviors among the certificate issuer nodes on a TDT.

Examiners:

List of Figures

List of Tables

Glossary of Terms

ACL	Access Control List
CA	Certifying Authority
CDT	*Children Degree Threshold*
DDTM	Dynamic Distributed Trust Model
DDTP	Dynamic Distributed Trust Protocol
DT	*Delegation Threshold*
IDEA	International Data Encryption Algorithm
IETF	The Internet Engineering Task Force
IPsec	Internet Protocol security extensions to IPv4. This is a protocol for negotiating encryption and authentication at the IP (host-to-host) level.
Ipv4	Version 4 of the Internet Protocol (IP)
PGP	Pretty Good Privacy
PKI	Public Key Infrastructure
RSA	Public-key cryptography algorithm, known by the initials of the three inventors (Rivest, Shamir, Adleman).
SPIN	Simple Promela INterpreter. A generic verification system that supports the design and verification of asynchronous process systems.
SPKI	Simple Public Key Infrastructure
SSH	Secure Shell. A unix shell program for logging into and executing commands on a remote computer.
SSL	Secure Sockets Layer. This is a protocol designed for providing encrypted communications on the Internet.
TDT	Trust Delegation Tree
THT	*TDT Height Threshold*

Acknowledgements

I sincerely thank my supervisor Dr. G. C. Shoja who gave me valuable advice during my research, which helped me keep it in the right direction.

Next, I gratefully appreciate New Media Innovation Center (NewMIC) and the Computer Science Department of University of Victoria for providing me with the financial support for carrying out this research work.

I also appreciate all the members of PANDA research group for their cooperation and for the encouragement they have given me at various stages of this research work. I especially would like to thank Glenn Mahoney, Md. Humayun Kabir, Doug Johnson, Eric Gowland and Jeff Hornsberger for proof reading the draft of the thesis and for their valuable comments.

I would like to thank Dr. Gene Racicot for his editorial suggestions on writing my thesis report.

Finally, I like to give special thanks to my husband, Wei Fu, who has encouraged me all along with my research work and sacrificed a lot of his time to take care of the family.

Chapter 1

Introduction

1.1 Motivation

As the Internet is increasingly being used in people's daily lives, it has also changed the way people do business and communicate with each other. Since many of the resources are private and not public, protection mechanisms are necessary to control access to those resources.

Various security mechanisms have been deployed to protect private or commercial digital information accessible through the Internet, such as Secure Shell (SSH) [Ylö96], and passwords at the application layer, Secure Socket Layers (SSL) [FKK96] at the transport layer, and Internet Protocol security extensions to IPv4 (IPsec) and firewalls at the network layer. Each of these mechanisms deals with a certain threat model, but each one has its own shortcomings. For example, the firewall performs access control on packets and connections, but it can't protect traffic from eavesdropping or modification, and it is not intended to guard against misbehavior by insiders. Furthermore, with the increasing size and complexity of the Internet, the security mechanisms are not suitable for such an open and distributed environment with a great deal of heterogeneity in the hardware and software.

Even when a secure and confidential channel is established between two entities, concern about the trustworthiness of the participants in a transaction remains. This concern translates directly into the well-known problem of access control to resources available via the Internet, and the degree of trust that can be assigned to a participant. Current security technologies cannot manage the general concept of trustworthiness. For

example, "*Cryptographic algorithms cannot say if a piece of digitally signed code has been authored by competent programmers and a signed public-key certificate does not tell you if the owner is a industrial spy.*" [AH98] To address the above problem, the existing security mechanisms need to be supplemented with a satisfactory trust model that manages trust effectively to provide a flexible and pervasive means of access control in computing network environments.

In recent years, several methods for expressing trust information in the digital world have been proposed.

X.509 [Al01] and PGP [Abd97a] express trust relations in distributed networks by managing certification of identities, which is only a part of total trust management. In X.509, certificate authorities are maintained in a top-down hierarchy, and only the bottom level certificate authorities issue certificates to users. This pre-defined hierarchy is not scalable for expressing trust relations. Trust in PGP is achieved using a web of trust model that breaks the hierarchical trust architecture and has no centralized authority that everyone trusts. However, the lack of fixed or formal certification paths in PGP creates anarchy, because each user decides on its own which keys to trust.

SPKI [EFL98] and PolicyMaker [AFL96] aim for decentralization of authority and management operations. However, the processing of certificates in SPKI may be done by means of an application-dependent method, and PolicyMaker has a complicated definition about the policy.

Currently, most trust decisions are incorporated into applications, "*which adds to the complexity of the applications and the inability to adapt changes in trust and lack in flexibility when setting up new relationships*" [T01]. Therefore, in this thesis, we concentrate on trust management in a separated functionality, which can be easily integrated into various applications thus providing a more scalable and flexible access control solution for distributed environments.

1.2 Applications for Using Trust in Access Control

The scope of this thesis is the trust relationships that exist between networked entities where some of these entities provide some kind of service. The entity may indicate a human being or a machine. The major concern of providing Internet services is about the trustworthiness of entities that access these services. Wherever entities without sufficient pre-established direct trust want to engage in transactions with protected services, trust establishment becomes an issue.

Trust applications involve every aspect of e-business. In the example of e-Bay, sellers and buyers need to trust each other before the final payment is made.

Another example mentioned in [HMM00] is that of access control to large databases of anonymous medical data for research purposes. The database is only accessible to authorized people. Hospitals from different countries can cross-certify by trusting the certificates issued by each other to create a web of trust in order to enable doctors in the hospitals to share data.

Coalition environments [FPPKK01] are characterized by the presence of multiple organizations or entities that have no common trusted root authority. In such an environment, controlled actions are presented in terms of roles within the trust domain of one entity and can be delegated to other roles within a different trust domain. Therefore, the entities can cooperate through their trust relationship to share protected resources that are necessary to the coalition and to protect the resources that they don't want to share.

1.3 Objectives

The objectives of this work are to provide an application-independent trust management model of access control for resources over the Internet, and to develop a set of scalable, consistent and reliable protocols to support this model. This thesis addresses the trust that is established and managed in truly distributed systems in which no member has a global view of the whole system.

Based on the establishment of the trust value, a Dynamic Distributed Trust Model (DDTM) is proposed for access control for Internet applications. The core of the DDTM

is the recommendation-based trust model organized as a Trust Delegation Tree (TDT) and the authorization delegation realized by delegation certificate chains.

In this work, we aim first to provide a method of enabling the establishment of an entity's trust value either when the entity is recommended through trusted intermediaries, or when the entity has direct transactions with the resource owner. To express the abstract trust notion as a measurable number, several parameters need to be considered for making a trust decision. In our analysis, these parameters are divided into objective and subjective ones. The objective parameters play important roles in deciding a direct trust value while the subjective ones are considered for indirect trust value.

After the method for calculating the trust value is provided, a Trust Delegation Tree is proposed as a means of managing the dynamic hierarchical trust relationship for trusted intermediaries. With the development of the Dynamic Distributed Trust Protocol (DDTP) and a TDT structure, a resource owner can be assisted to establish trust relationships with clients through the delegation certificate chains on its TDT.

1.4 Organization of This Thesis

The remainder of this thesis is organized as follows. Chapter 2 introduces the existing access control mechanisms and the trust notions in these systems. Chapter 3 analyzes the trust relationships in recommendation-based trust and proposes the method for calculating trust values. Chapter 4 gives an overview of the DDTM and the detailed algorithms applied inside this model. Chapter 5 describes the data and protocol messages that are used for the DDTM. Chapter 6 provides state diagrams for the DDTP and applies SPIN to simulate and verify the abstract design. Chapter 7 simulates the DDTP to demonstrate the dynamic trust relationship managed on a TDT. Chapter 8 concludes with a summarization of the major contributions in this work and suggests future work.

Chapter 2

Background and Related Works

2.1 Access Control Systems

The function of an access control system is to let authorized users access resources, and to deny illegal users such access. These systems always need to answer the basic questions: *"Who signed this request?"* and *"Who is permitted to do what?"* There are two steps involved in access control - authentication and authorization, in which authentication confirms the identity of a user, and authorization is concerned with what rights a user is permitted. They answer two separate types of questions:

1. Is this actually Alice who is requesting access to resources (Authentication)?
2. Is Alice allowed to access the resources kept by the resource owner (Authorization)?

2.1.1 A Centralized Approach

In the paper by Lampson [Lam74], the *access control matrix* was introduced to model access control. Lampson's access control matrix puts objects in the matrix columns and subjects in the matrix rows. Access rights are the operations allowed by a subject on an object. An Access Control List (ACL) is the concept derived from the access control matrix, which is a table that specifies which access rights each subject has to a particular object.

A password application, such as the UNIX username and password, is an application of an ACL. If a person has a correct username and password on a server, the server will allow the person to log in.

However, as pointed out by both [BFIK99] and [Kan01], an ACL style access control mechanism could be easily implemented in a single organizational domain with a centralized network, but this became difficult when the computer systems had an increase in the number of users and extended over a large physical area. The problems mainly appear in the following four aspects:

1. In distributed systems, the user's identity is not well known in advance.
2. An ACL on a central server could not be able to handle a large number of requests and a large number of users over the Internet.
3. An ACL also does not have enough flexibility to describe access properties and security policies for a distributed system. When new and diverse security policies are created, an ACL, which is the pre-defined authorization list, cannot accommodate them.
4. An ACL enforces a uniform access control policy that may be not suitable for the large number of entities in a distributed system. Different parts of a distributed system should be able to have different access control policies.

In response to these difficulties, the concept of delegation was introduced to enable access control to be implemented in distributed networks, and to improve the scalability of access control for handling a large number of users. Delegation is defined in [MWTO] as " *1. the act of empowering to act for another; 2. a group of persons chosen to represent other*". It enables one entity to delegate some or all of its rights to another entity.

2.1.2 A Distributed Access Control Management System

Distributed access control is enabled through the use of public key certificates containing statements that one principal makes about another. Authentication and

authorization protocols utilize the public key certificates to delegate access rights to users. For example, assume a resource owner issues a certificate to an access control manager (top of the access control management hierarchy), and the access control manager is allowed to further delegate certificates to other users and so on. Later, when a user presents a certificate, the resource owner will only need to check whether the certificate chain of distributed access managers is valid and need not keep a long list of those with access rights. In the above scenario, the access control system is truly distributed, without the need for a centralized database.

A distributed access control using credential-based system was initially described by Bull *et al.* [BGS92]. In this system only shared-secret mechanisms are used. A server itself, or its trusted agent, is able to issue access certificates to authorize users to use the services provided by the server. The certificate is signed by using a secret-key algorithm, and only the server and its trusted agent know the key.

A user can delegate some or all of its rights to other users as long as the user has an ability to delegate. As a result, a chain of delegation is formed, and the server can verify the chain because the chain originates from the server. Finally, the server is able to control the access of the users as long as they can provide the secret key.

However, key management is a main challenge in credential-based distributed access control systems based on symmetric cryptography. Secret keys must be distributed with each identity and must also be protected from theft, and multiple secret keys are provided for different authority-client pairs. This problem is simplified by public-key cryptography because only one public key for each identity is needed for distribution to the entire world.

An identity certificate was originally introduced by Kohnfelder [Kon78] to bind a public key to the identity of the key owner. The identity certificate is signed by a trusted entity. Various technologies based on identity certificates are used in public-key cryptography, such as the anarchic PGP web of trust and X.509 Public Key Infrastructure (PKI) with a hierarchy of Certification Authorities (CA). Since PKI is a hierarchical authority structure, it is mainly useful only inside a single organization. Moreover, an

identity certificate only maps public keys to names, and the names must be mapped again to the access rights. Therefore, the identity certificate based techniques, which have a trend towards centralized management of the identity, are not suitable for distributed systems.

2.1.3 A Trust Management Approach

Blaze *et al* first proposed a trust management system and implemented its prototype, PolicyMaker, in 1996 [BFL96]. In this paper, the authors explicitly specified that the problems about trust management are concerned with security policies, security credentials and trust relationships. A trust management system is a general-purpose, application-independent mechanism for checking credentials, and it offers a unified approach to specifying and interpreting security policies, credentials, and relationships that allows direct authorization of security-critical actions [BFIK99]. Credentials in the trust management directly authorize actions with public keys and specify delegations of trust among the public keys.

Trust management answers the question "*Does the set C of credentials prove that the request r complies with the local security policy P?*" The "*trust management engine*" is a separate system component that works as *f(r, C, P)->{true, false}*, where the inputs are *r*, *C* and *P*, and the output is the decision whether the compliance exists or not.

Figure 2.1 shows the trust management engine.

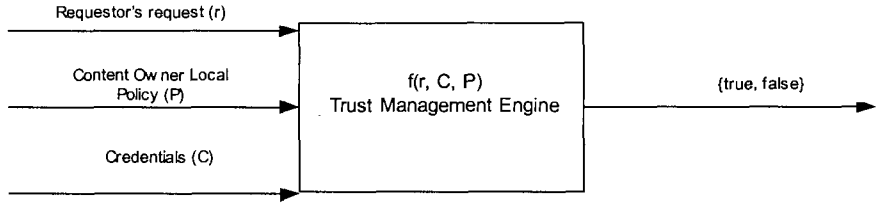


Figure 2.1:The trust management engine

X.509 and PGP can support security in network applications by managing certification of identities, which is only a partial role of total trust management. In X.509 and PGP, applications decide how the certified identity is acted upon. The Simple Public Key Infrastructure (SPKI) uses the authorization certificate to represent an authorized user,

but the specification of the way to express the authorization rules and processing of the certificate may be done in application software. PolicyMaker is the first embodiment of a *trust management engine* and is a general-purpose, application-independent algorithm for processing certificates.

2.2 Notion of Trust

Since trust is a difficult concept to define exactly, several definitions are worth noting. In the Merriam-Webster Thesaurus [MWTO], trust is defined as "*complete assurance and certitude regarding the character, ability, strength, or truth of someone or something*". In computer security literature, Trusted Computing Base (TCB), defined as a set of all hardware, software and procedural components that enforce the security policy, is the embodiment of the above definition of trust. It is very obvious that trust in security mechanisms is generally implemented with the goal of establishing complete certainty, which is a "*Yes*" or "*No*" binary control.

As the Internet has become major arena of commercial transactions during recent years, the concept of trust has been defined more explicitly in [Hmd] with regard to relationships in business activities through the Internet:

"Trust is the willingness of one party to adopt a vulnerable position in relation to systems or another party, based on the assumption that the other party will perform a certain action or comply with an obligation, and without there being a simple way to monitor this action."

Based on this definition, trust is not absolute assurance about the other party, but rather is a prediction based on knowledge of the other party. When two parties are not known to each other, one can predict the trustworthiness of the other by any possible reference from other third parties. Complete trust is formed through long-term experience, and it is difficult to form but easy to lose.

The concept of trust is also defined in [Gam90] as:

"trust (or, symmetrically, distrust) is a particular level of the subjective probability with which an agent will perform a particular action, both before [we] can monitor such

action (or independently of [our] capacity of ever to be able to monitor it) and in a context in which it affects [our] own action".

Abdul-Raduman et al pointed out in [AH98] that this definition emphasizes three important features about trust: trust is subjective; trust is affected by those actions that we cannot monitor; the level of trust depends on how our own actions are in turn affected by the agent's action. According to the above definition, trust is not a simple "*Yes*" or "*No*" concept but can be expressed in fine-grained levels. Based on the description in [Hmd], Figure 2.2 shows the most common factors used in making decisions about trust levels in real life. These factors enable and complicate the trust decision-making, and action is based on the result of the level of trust.

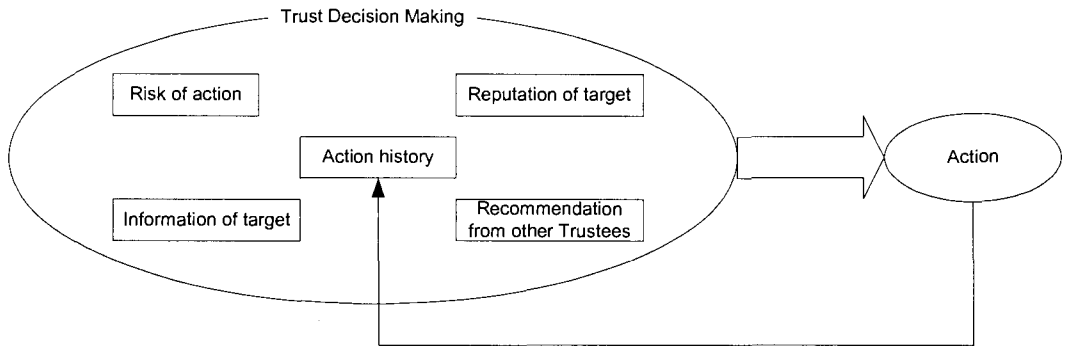


Figure 2.2: Factors in trust decision-making

2.3 Trust Models In Related Work

As online business activities have grown, the demand for security services has increased. Authentication and authorization are two objectives of security for achieving access control, and cryptography is the main method used to implement these objectives.

Public key cryptography is the most commonly used mechanism to authenticate and authorize entities in the Internet. Different PKI trust models have been proposed, such as the Pretty Good Privacy public-key cryptographic system (PGP), X.509 standard Public Key Infrastructure (PKI) and Simple Public Key Infrastructure (SPKI) [Wan98]. However, the question here is: *How much can a user trust cryptographic keys?* The notion of trust is a way to make people feel confident about the public key obtained in the

public key cryptography to enhance security. Trust is established by interpreting policies to validate credentials.

2.3.1 PGP

Pretty Good Privacy (PGP) was created by Phil Zimmermann in 1991 to make a decentralized public-key cryptographic system available for use in exchanging secure e-mail over the Internet. PGP utilizes an encryption algorithm, such as RSA or IDEA, to encrypt/decrypt and/or sign messages. Trust in PGP is achieved using a web of trust model that breaks the hierarchical trust architecture and has no centralized authority that everyone trusts. The basic idea is that each PGP user maintains a list of public keys, and the trusted PGP users can be used to introduce others, i.e., trust is a transitive relation.

There are two explicit trusts in PGP [Abd97a], of which one is the trustworthiness of public-key certificates, and the other is the trustworthiness of an introducer.

The key certificate is central to PGP, which makes the trust relationship spread. When a user cannot verify the authentication of a public key by himself or herself, the user can rely on the judgment of other users who have signed this key. For example, you can send your public key to your trustees to request them to sign your public key. The returned signatures you will post make you more trustable than your public key alone. The key certificate contains the public key itself, the owner ID (such as Name and Email Address) and one or more digital signatures. When a user receives a key certificate, the user will verify the public key by noting the people who have signed the key and the trust level the user has given to the signer. For example, a user's public key could be treated as valid if it had one fully trusted signature or two marginally trusted ones.

Each user could also sign any individual's public key that the user trusts as an introducer, by using his or her private key. Moreover, PGP allows four levels of trust to be specified on each introducer: full, marginal, untrustworthy and don't know. This level of trust indicates how much the user trusts the owner of the public key vouching for the authenticity of someone else's public key.

- PGP has some advantages, such as:

- There is no preordained core set of Certifying Authorities. The CA is a trusted third party that issues a digital certificate.
- Any user may sign public-key records and act as an introducer.
- A user can decide whether to accept others' public keys, and the user does not need to wait until a CA gives out a certificate.

However, the lack of fixed or formal certification paths not only makes searching for an appropriate certificate complicated, but also the authenticity uncertainty of any PGP key certificate becomes a critical problem.

- The followings are the disadvantages of PGP:
  - PGP does not enforce any structured trust hierarchy
  - PGP assumes that a trusted introducer will never certify someone who is not trustworthy. This is the reason that PGP is simple and is not appropriate for use beyond secure personal communication.
  - PGP is not scalable beyond a relatively small community of trusted individuals.

2.3.2 X.509 Public Key Infrastructure

ITU-T X.509 was published in 1988. It is a part of the X.500 directory recommendations to define a standard Public Key Certificate (PKC); X.500 provides a global and distributed directory standard for Internet users. X.509 includes data formats and procedures related to the distribution of public keys via PKCs that are digitally signed by CAs, and it is widely used as a basis for a PKI. Therefore, PKIX, one the IETF working group has been working on since 1995, is a Public Key Infrastructure based on X.509 [Pkix02]. Compared to PGP, X.509 certificates contain more information, but the basic purpose is the same as simply linking users to keys.

X.509 Public Key Infrastructure (PKIX) is based on two basic components: digital certificates and Certifying Authorities (CAs). Individuals or organizations apply to CAs for digital certification. CA will verify the identity of these individuals or organizations, get their public keys, and issue certificates signed by the CA's private key. PKIX trust has the following aspects [Al01]:

- The CA system that is used to issue and to maintain the certificate is secure.
- The CA's keys are secured and have not been compromised.
- The process of verifying the identity of the certificate applicant is robust.
- The process of maintaining the certificates and making them valid is also robust.

- The CAs are trustworthy.

Since a single CA is not enough, multiple CAs are maintained within X.509 and arranged in a top-down hierarchy. A root CA, being the most trustworthy CA in the hierarchy issues and signs certificates to other CAs below it (called subordinate CAs), which can further sign other CAs in the next level, or users. Only at the bottom level do CAs issue certificates to users. An individual signed by one of the subordinate CAs must present the certificates of all CAs along its certificate chain. In order to achieve trust between two parties, each should verify all certificates along the chain of certificates supplied by the other party, until each of them reaches the certificate of a CA that both trust.

There are three general hierarchical structures described in [Kan01]. In the *separate secure domains* structure, see Figure 2.3, if users within a domain want to communicate securely, they obtain information about each other through their common trusted third party. The problem rises when users from different domains want to communicate. The solution is to group these trusted third parties to form a large domain and control them by means of a higher-level CA. This leads to the *strictly hierarchical* structure shown in Figure 2.4. If A wants to communicate with B and they are not under the same CA, A must find the certificate path from A to B. The certificate path looks like "CA_1, $cert_1$, CA_2, $cert_2$, ..., CA_n, $cert_n$, *where* $cert_n$, $1 \leq i < n$, *is a certificate of* CA_{i+1} *that has been signed by* CA_i *and* $cert_n$ *is a certificate of B*" [BFL96]. In the *multiple root trees* structure, shown in Figure 2.5, there is no single highest CA so that a group of peer CAs should trust each other.

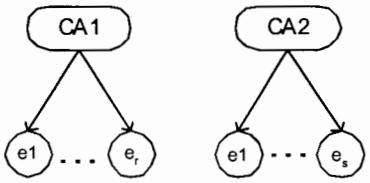


Figure 2.3: Separate security domains

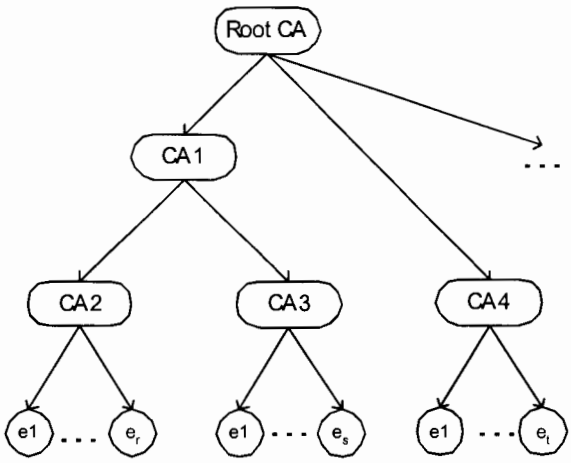


Figure 2.4: Strict hierarchy

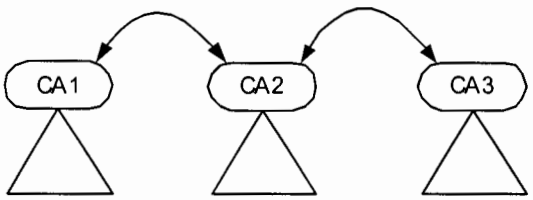


Figure 2.5: Multiple rooted trees

- X.509 Public Key Infrastructure has advantages for providing secure service for some applications that desire a centralized hierarchy.

- The followings are the disadvantages of X.509 Public Key Infrastructure [Al01]:

  - If a CA is compromised, all certificates signed under this CA will be nullified.
  - A CA exhibits a congestion point with the increasing number of certificates.

- The assigned single entity that runs this central CA should be trusted by all organizations and individuals in all countries of the whole world. This might not be achievable in practice.
- Everyone who wants to use name certificates should belong to the same hierarchy.
- X.509 assumes that every user has a unique name.

2.3.3 Simple Public Key Infrastructure

Simple Public Key Infrastructure (SPKI), which is an Internet draft that was started by the IETF SPKI working group in 1996 [EFL98], is a proposed standard for public-key certificates. It is based on public key cryptography and is mainly used in access control. SPKI emphasizes decentralization and the use of keys rather than names. Access rights can be assigned directly to the public keys instead of to names, so that SPKI can avoid the problem of mapping public keys to names, and the trusted third party is no longer necessary to certify the mapping of a name and a public key. Furthermore, different permissions can be defined in SPKI certificates.

PGP and X.509 are identity-certifying systems used for access control. The certificate does the binding of a key to a name (Figure 2.6), and the name is authorized to perform certain operations according to an access control list (ACL). In contrast to identity certification, authorization certifications such as SPKI and policymaker, which will be discussed later, bind a key to an operation directly (Figure 2.7).

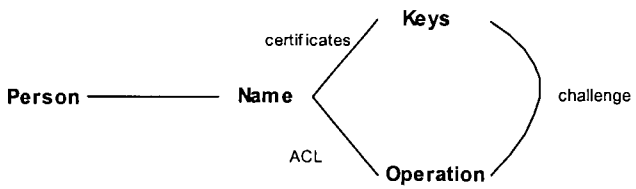


Figure 2.6: Identification certificate binding

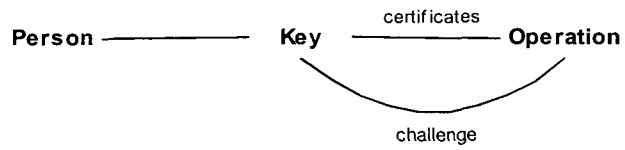


Figure 2.7: Authorization certificate binding

SPKI supports the following functions:

- Everyone can freely issue certificates and delegate access rights to others.
- Rights can be transferred.
- Authorizations can be freely defined and distributed.
- Validity dates are clearly written in certificates.

The five components that are shown in Figure 2.8 form the body of a certificate, and a SPKI certificate is signed by an issuer's private key. It is possible for a subject to hold several chains of certificates that give the same rights. Moreover, the issuer can freely give rights to a subject as long as the rights are not greater than the issuer's.

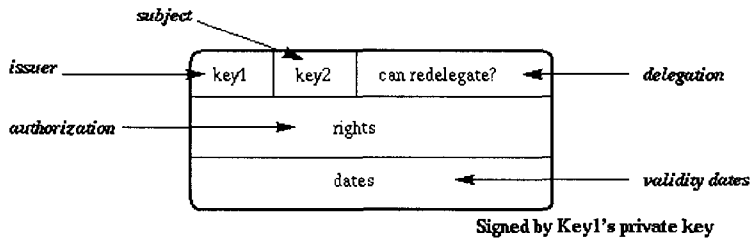


Figure 2.8: SPKI structure [Wan98]

- Advantages of SPKI
  - SPKI has no pre-defined trust hierarchy, but any user can define whom to trust.
  - SPKI binds permissions to keys directly.
  - Certificates are not stored in a global repository. Certificates are brought by a keyholder to the verifier.

- Disadvantages of SPKI
  - SPKI has difficulty on controlling redelegation rights.
  - An SPKI certificate is not fully programmable to provide a flexible and extensible trust management.

2.3.4 PolicyMaker

PolicyMaker was the first tool for processing signed requests in a trust management engine proposed in [BFL96]. It accepts a set of local policy statements, a collection of credentials, and a string describing a proposed action as input, and evaluates the proposed action by interpreting the policy statements and credentials. The output of PolicyMaker is either a simple yes/no answer or additional restrictions that would make the proposed action acceptable.

PolicyMaker resembles SPKI in that it addresses the authorization problem directly rather than handling the problem indirectly via authentication and access control.

Besides this feature, there are several general principles in PolicyMaker. First, the policies, credentials, and trust relationships are programmable; second, the system can support trust relationships that can be in whatever form naturally occurs in the application and can be changed without altering the trust management system; finally the mechanism for verifying credentials does not depend on the application.

The PolicyMaker mechanism is focused on the implementation of the function $f(r,C,P) \rightarrow \{true, false\}$, but the collection of credentials and all cryptographic verification of signature are left to be the calling applications' responsibilities. One of the two core operations in PolicyMaker is to process a query, which is a request to determine whether a particular public key is permitted to perform a particular action according to local policy. The other one is assertion, which confers authority on keys. PolicyMaker processes a query according to the trust information in the assertion. Two types of assertions are certificates and policy. The first one is a signed message that binds a particular authority structure to a filter, and the second one also binds a particular authority structure to a filter. The filters are security policies and credentials defined in terms of predicates. The format of the assertion and the query are as follows:

Source ***ASSERT*** *AuthorityStruct* ***WHERE*** *Filter*

Key1, key2, ..., keyn ***REQUEST*** *ActionString*

The *Source* value indicates the authority, which can be either local policy or a public key of a third party. *AuthorityStruct* specifies the public key or keys to which the assertion applies. *ActionString* is an application-specific message that describes a trusted action requested by a (sequence of) key(s) [BFL96].

PolicyMaker represents the notion of trust by binding public keys to predicates that describe the actions that they are trusted to sign for. In an expressive PolicyMaker language, a public key can be directly authorized to perform certain actions so that the trust relationship is more flexible in PolicyMaker than in other trust models.

- Advantages of PolicyMaker
  - PolicyMaker is a general and flexible system that is separated from the application-specific policy.
  - PolicyMaker credentials and policies are fully programmable, and PolicyMaker assertions can be written in any programming language.
  - PolicyMaker is more secure because risks are reduced by eliminating the binding of the identity to the public key.

- Disadvantages of PolicyMaker
  - PolicyMaker is guaranteed to be correct only when all assertions are monotonic, which certain types of policies in practice do not obey.
  - Description languages for action must be carefully selected, and the predicate in policy and certificate assertions must be carefully written to reflect the intentions of the policy [BFL96].

2.3.5 Comparison of Trust Models

Table 2.1 lists the comparison among the above trust models.

Table 2.1: The comparison among the trust models

	X.509	**PGP**	**SPKI**	**PolicyMaker**
Certification Type	Identity certification	Identity certification	Authorization certification	Authorization certification
Trust Model	Top-down oriented hierarchical structure	Web of trust structure	Freely issued	Fully expressive and programmable policies and credentials
Explicative Delegation Mechanism	No	No	Yes	Yes

Chapter 3

A Recommendation-Based Trust Concept

Trust is formed through experience and knowledge. However, in a large distributed system, obtaining knowledge about every entity in the system is a nearly impossible task. A recommendation is a means for establishing a trust relationship between entities in a network, and thus for coping with uncertainty. As human beings use word of mouth to learn about a stranger, an entity in a network can make a decision about whether to trust an unknown entity based on the recommendations of its trusted intermediaries. Figure 3.1 shows the recommendation-based trust scenario proposed in my thesis.

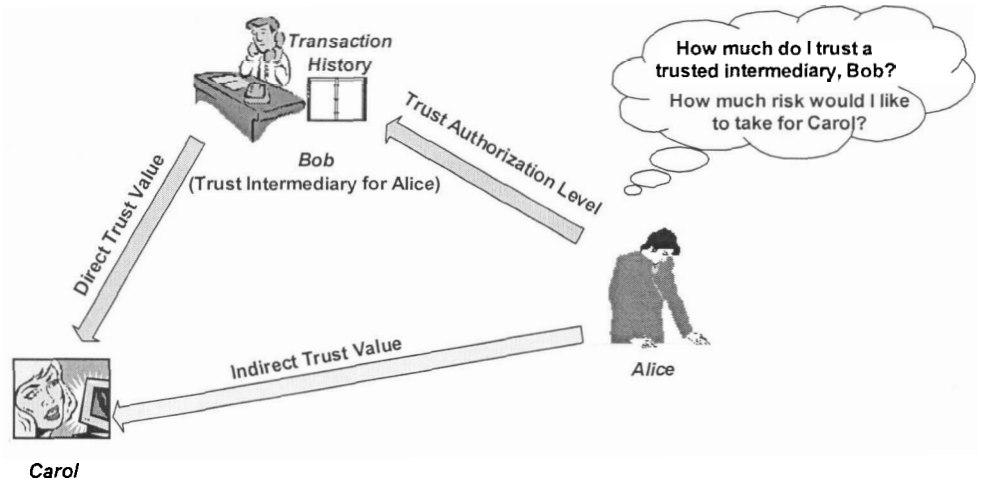


Figure 3.1: Recommendation-based trust scenario

3.1 Trust Relationships

There are two distinguishable trust relationships: *direct* and *indirect*. For example, if Alice trusts Bob as a result of a long friendship experienced over time, then there is a direct trust relationship between Alice and Bob. However, if Alice trusts Bob enough to accept his recommendations about Carol's trustworthiness, then there is an indirect trust relationship between Alice and Carol through an intermediary, Bob.

In our model, the degree of trust value should be measurable in order to evaluate the degree of trust that one entity has in another. There are three types of explicit trust values, each of which is represented by real numbers in interval [0, 1], where 0 indicates complete distrust, and 1 stands for the highest trust value.

Direct trust value: This represents the quantified trustworthiness in a direct trust relationship. The direct trust value that entity u trusts entity v is denoted as $dT(u, v)$.

Indirect trust value: This represents the quantified trustworthiness in an indirect trust relationship. The indirect trust value that entity u trusts entity v is denoted as $iT(u, v)$.

Trust authorization level: This represents the quantified authorization level of an entity for recommending other entities. The trust authorization level that entity u grants to entity v is denoted as $aL(u, v)$.

3.1.1 Features of A Trust Relationship

A trust relationship can be treated as a relation on a set of entities, and it is always between exactly two entities. We can consider the mathematical properties of relations presented in [SR86] as applied to our trust relationships.

Reflexive relations: *"R is reflexive if for all $x \in A, xRx$."*

Since an entity would never be malicious to itself, it would always trust itself. However, a trust value is evaluated by other entities, not by the entity itself.

Symmetric relations: *"R is symmetric if for all* $x, y \in A, xRy = yRx.$ *"*

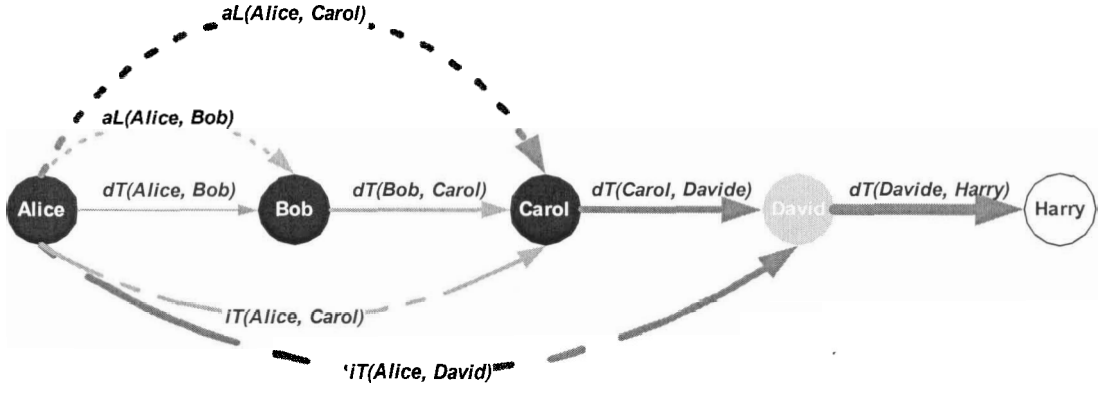
In general, a trust relationship is asymmetric because if Alice trusts Bob it does not always follow that Bob also trusts Alice.

Transitive Relations: *"R is transitive if for all* $x, y, z \in A, xRy$ *and* $yRz \Rightarrow xRz.$ *"*

If delegation is permitted, it is basically true to say that a trust relationship is transitive. For example, if Alice trusts Bob's recommendation and Bob trusts Carol, this implies that Alice trusts Carol depending on how much Alice trusts Bob and Bob trusts Carol.

3.1.2 Trust Relationship Graph

A weighted digraph is used to represent the trust relationship. The set of vertices includes all the entities, and an edge of $E(u,v)$ represents a trust relationship and is weighted by the value of trust that entity u has in entity v. In Figure 3.2, the arrow lines with the same line width indicate the transitive closure. The black-filled circles are the entities authorized for recommending other entities to Alice, but the gray-filled circle is not able to recommend other entities, such as Harry, to Alice, so that Alice does not grant a trust authorization level to David.



Legend:

—·—·—▶	Indirect trust value
- - - - - -▶	Trust authorization level
——▶	Direct trust value

Figure 3.2: Trust relationship graph

Methods for deciding and quantifying the trust values shown in Figure 3.2 are described in next.

3.2 Trust Metrics

Before giving the detailed description of the algorithm to calculate trust values, the elementary concept about which variables influence quantified trust and distrust must be addressed.

Although existing trust management systems are seldom concerned about how to quantify trust into real numbers, it is very important to have a measurable trust value to evaluate the trustworthiness between two entities. There could be a set of metrics contributing to the value of trust, and the multiple metrics could be combined into a single trust value.

3.2.1 Parameters of Trust Metrics

Although there is no single universal value system for the quantitative measurement of trust, a set of metrics critical to a trust value could be analyzed in order to decide the degree to which one entity has trust in another entity. Several parameters that influence the trust value are mentioned in [Man00] and [TPC], and some of them are rephrased and incorporated into the trust metrics in this thesis.

The following parameters provide evidence for trust metrics, and they are classified into objective parameters and subjective parameters. The objective parameters are measured as a probability of likelihood of trustworthiness while the subjective parameters are regarded as a subjective measurement of confidence. All of these parameters are real numbers in the interval [0, 1].

- **Objective Parameters**

Transaction history: The transaction history of an entity consists of the quantity and the monetary value of the previous transactions of the entity. With respect to the trust, the quantity of transactions and their monetary values are used to quantify a belief in an entity's performance and loyalty. A successful transaction means that the service provider is paid by the consumer of the service. $Count_{total}(v \rightarrow u)$ and $Cost_{total}(v \rightarrow u)$ denote the total quantity and monetary value of the transactions that entity v did with entity u; $Count_{success}(v \rightarrow u)$ and $Cost_{success}(v \rightarrow u)$ indicate the quantity and the accumulated monetary value of the successful transactions that entity v did with entity u; $Count_{threshold}(u)$ and $Cost_{threshold}(u)$ represent entity u's thresholds for the expected quantity and accumulated monetary value of transactions.

- Performance: This is the measure of how reliable, with respect to appropriate behavior, entity v is as viewed by entity u. The probability of successful performance is shown in Equation 3-1.

$$P_{performance}(u,v) = \frac{Count_{success}(v \rightarrow u)}{Count_{total}(v \rightarrow u)} \times \frac{Cost_{success}(v \rightarrow u)}{Cost_{total}(v \rightarrow u)} \qquad \textit{Equation 3-1}$$

- Loyalty: This relates to the quantity and accumulated monetary value of the transactions expected from an entity. Transaction quantity and monetary value thresholds indicate the levels below which the entity's trustworthiness will be reduced. Those two thresholds prevent assigning a high trust value to an entity before a history of trust is built. Therefore, the loyalty of entity v as judged by entity u is given in Equation 3-2.

$$P_{loyalty}(u,v) = \frac{Count_{success}(v \rightarrow u)}{Count_{threshold}(u)} \times \frac{Cost_{success}(v \rightarrow u)}{Cost_{threshold}(u)} \qquad \textit{Equation 3-2}$$

$$\textit{and} \quad \begin{cases} \dfrac{Count_{success}(v \rightarrow u)}{Count_{threshold}(u)} = 1 & \textit{if } Count_{success}(v \rightarrow u) \geq Count_{threshold}(u) \\ \dfrac{Cost_{success}(v \rightarrow u)}{Cost_{threshold}(u)} = 1 & \textit{if } Cost_{success}(v \rightarrow u) \geq Cost_{threshold}(u) \end{cases}$$

The rationale for using $P_{performance}$ and $P_{loyalty}$ in direct trust value calculation is shown in the example of section 3.3.1.1.

- **Subjective Parameters**
  - *Indemnity of trusted intermediary* (*denoted as* $aL(u,v)$) : If an entity u has a trusted intermediary v standing as a guarantee for the trustworthiness of another entity e, then there is an increase in the trust level between entity u and e.
  - *Suspicious trusted intermediary* (*denoted as* $STI(v)$) : If an entity is recommended through an untrustworthy intermediary v , the entity's trust value is decreased.
  - *Risk of transaction* (*denoted as* $R(u)$) : A lower risk number associated with an entity u means the entity is cautious in transacting with another entity. A value of 0 means not allowing any risk, while a value of 1 means accepting a lot of risk.
  - *Suspicious transaction pattern* (*denoted as* $STP(v)$) : If an entity v conducts transactions in a certain period of time or with a certain frequency, this suspicious activity will trigger a warning to its transaction partners. Its partners may then decrease the entity's trust value.

3.3 Trust Quantification

In this section, we explained a set of metrics that may affect trust. We now present a way to calculate the trust values based on those metrics.

3.3.1 Computation of Direct Trust Value

The direct trust value is decided by the objective parameters of transaction history. It is straightforward to measure the direct trust value by the number and the monetary value of the successful transactions over the transaction history. However, this measure should

be weighted along side the loyalty value $P_{loyalty}(u,v)$. As $P_{loyalty}(u,v)$ and $P_{performance}(u,v)$ were defined in section 3.2.1, the direct trust value $dT(u,v)$ assigned by entity u to another entity v is defined in Equation 3-3.

$$dT(u,v) = P_{loyalty}(u, v) \times P_{performance}(u,v) \qquad \textit{Equation 3-3}$$

Suppose Bob has 50 previous transactions with Alice, but the number of successful transactions is only 40. At the same time, assume the total cost of the successful transactions is \$1000 and amounts to 50% of the total cost of 50 transactions. Thus, we get $P_{performance}(Alice, Bob) = 0.8 \times 0.5 = 0.4$. We also assume that the thresholds for Alice considering Bob's loyalty are 100 successful transactions and \$2000 successful transaction value. By applying these thresholds, $P_{loyalty}(Alice, Bob) = \frac{40}{100} \times \frac{1000}{2000} = 0.2$. In this case, the direct trust value that Alice assigns to Bob is:

$$dT(Alice, Bob) = (\frac{40}{100} \times \frac{1000}{2000}) \times (0.8 \times 0.5) = (0.4 \times 0.5) \times 0.4 = 0.08$$

Following the example above, assume Cathy is doing transactions with Alice, and all of the transactions Cathy does with Alice are successful and no fraud is committed.

Figure 3.3 shows the calculated direct trust value from Alice to Cathy when the monetary value of each transaction is randomly selected in the range of \$10 to \$100.

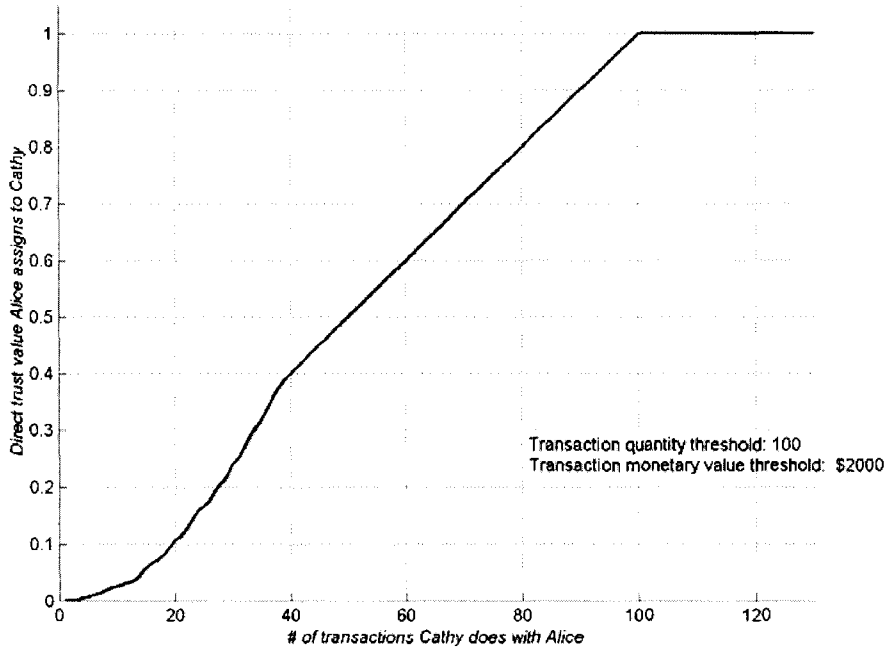


Figure 3.3: The direct trust value Alice assigns to Cathy

Note that after 40 transactions, the monetary value of $P_{loyalty}(Alice, Cathy)$ reaches its threshold, and the direct trust value grows linearly from then on. When both the thresholds are reached, the direct trust value from Alice to Cathy becomes 1.

3.3.2 Propagation of Trust

If an entity u would trust what entity w recommends, the authorization level from entity u to entity w is represented by $aL(u, w)$. Furthermore, if entity w directly trusts an entity v and also trusts what entity v recommends, the authorization level from entity u to entity v is $aL(u,v)$. In this case, delegation is explicitly declared. As shown below, trust is propagated by the production rule at the time that the chain of trust relationship is formed.

$$aL(u,w) = dT(u, w) \qquad \textit{Equation 3-4}$$

$$aL(u,v) = aL(u, w) \times dT(w,v) \qquad \textit{Equation 3-5}$$

3.3.3 Computation of Indirect Trust Value

Although two unknown entities do not have much knowledge with which to evaluate the trust relationship between them, they can be channeled through trusted intermediaries. The recommendation of a trusted intermediary is the critical factor for deciding indirect trust value from one entity to another. However, other subjective parameters listed in section 3.2.1 could affect the entity's indirect trust value as well.

Although these parameters provide evidence useful for analyzing a trust relationship, none of them is certain when it comes to deciding the trust value. For example, assume Alice believes in Bob and would like to believe in what Bob says about Carol. However, if Alice learns in a different way that Carol has a very suspicious activity, then, whenever Alice evaluates the trust value for Carol, she will consider Bob's recommendation as well as her suspicion about Carol.

We use the Dempster-Shafer function to combine several parameters to get the indirect trust value between the two unknown or less well-known entities.

The Dempster-Shafer theory originated with the work by A.P. Dempster in 1968 [Dem68], and Glenn Shafer brought the material to a wider application in his doctoral dissertation in 1976 [Sha76]. Dempster-Shafer theory is well known as the theory of belief function. It provides a numerical method for evidential reasoning and a powerful method of combining accumulative evidences. It aims to model and quantify uncertainty by degree of belief. Dempster's rule expressed in [Lug02] states that combined belief of trust for considering n evidences is expressed by the indirect trust value in Equation 3-6.

$$iT(u,v) = m_n(Z) = \frac{\sum_{X \cap Y = Z} m_{n-2}(X) m_{n-1}(Y)}{1 - \sum_{X \cap Y = \Phi} m_{n-2}(X) m_{n-1}(Y)} \qquad \textit{Equation 3-6}$$

In a Dempster-Shafer reasoning system, all the mutually exclusive possibilities are enumerated in a "*frame-of-discernment*", denoted as Θ.

m: A mass probability. This is defined for each member of the set 2^{Θ} and takes values in the range [0,1].

n: This represents the number of sources of independent evidence.

X: This is the set of subsets of Θ to which m_{n-2} assigns a nonzero value.

Y: This is the set of subsets of Θ to which m_{n-1} assigns a nonzero value.

Z: This is a subset of Θ.

The quantity $m_n(Z)$ measures the amount of belief that is assigned to the subset Z of all the possibility, Θ. Evidence for sets X and Y that support Z (i.e. $X \cap Y = Z$) is summed and is normalized by the evidence that X contradicts Y (i.e. $X \cap Y = \Phi$).

Suppose Θ contains two possibilities: Trust (U) and distrust (U'). Continuing with the previous example, assume Alice believes in Bob's recommendation with 0.5, and Bob trusts Carol with a direct trust value of 0.8. Therefore, by Bob's recommendation, Alice trusts Carol at $0.5 \times 0.8 = 0.4$ $\{U\}$. In the mean time, Alice presents a distrust factor 0.2 $\{U'\}$ for Carol's suspicious activity pattern. In the following table, Bob's recommendation is the evidence source 1, and Alice's suspicion for Carol is the evidence source 2.

Table 3.1: Probability distribution on the subsets of {U,U'} by combining evidence m_1 and m_2.

$m_2 \backslash m_1$	**0.4{*U*}**	**0{*U'*}**	**0.6 {*U,U'*}**
0{*U*}	0{ *U* }	0{Φ}	0{ *U*}
0.2{*U'*}	0.08{Φ}	0{ *U'* }	0.12{ *U'*}
0.8{*U,U'*}	0.32{ *U* }	0{ *U'*}	0.48{*U,U'*}

The indirect trust value from Alice to Carol is obtained by combining the two pieces of evidence, and the result is as follow.

$$iT(Alice, carol) = m_3(\{u\}) = \frac{0.32}{1-0.08} = 0.348$$

Chapter 4

Architecture of The Dynamic Distributed Trust Model

4.1 Overview of The DDTM

As the possibility of doing business or sharing resources over the Internet has dramatically increased, the notion of trust management in electronic communities has become an important research issue.

Since Internet services are not limited to specific range of domains or organizations, a distributed, flexible and general-purpose trust management scheme is necessary. The proposed dynamic distributed trust model (DDTM) is a model for establishing a trust relationship between entities that may never meet each other, and aims to provide a scalable, decentralized access control mechanism over the Internet.

The objectives of the DDTM are to:

- Provide a way to evaluate the trust between a resource authority and unknown entities by the recommendation of trusted intermediaries.
- Decentralize the trust management from a central resource authority, and establish a Dynamic Distributed Trust Protocol (DDTP) that is reliable for issuing recommendation certificates, scalable for handling a large number of requestors, and flexible for providing general trust relationship evaluation.

The DDTM provides a distributed key-oriented certificate-issuing mechanism with no centralized server granting authorization. The certificate-issuing mechanism is decentralized, but it has a structure that can tightly control the certificate issuers to serve the interests of a content owner.

The DDTM builds a hierarchical tree structure from an authorized root, and delegates the certificate-issuing authority to the nodes of the tree. The tree is dynamically updated to represent the trust relationship between nodes. Any node on the tree may delegate certificate-issuing authority to any others it trusts and add them as children on its subtree. The trust delegation certificate uses the format of SPKI (Simple Public Key Infrastructure) as described in [EFL98], and it is used in the DDTM to delegate certificate-issuing authority.

In this thesis, the DDTM is presented in the context of digital content distribution. Before the DDTM is described in detail, its specific components are explained here to provide an overview of its trust model.

- **Content Owner:**

A content owner holds the master copy of the content. It can delegate certificate-issuing authority to a root it directly trusts, and the root takes responsibility for managing the certificate issuers for this content owner. Each content owner has only one root.

- **Requestor:**

This is a user who wants to utilize certain content. It can access the content through obtaining a signed recommendation certificate, which provides evidence for the content owner to trust the user.

- **The Certificate Issuers:**

These are trusted intermediaries for a content owner. A Trust Delegation Tree is a virtual relationship for a specified content owner. Every node on this tree is able to sign delegation certificates for its children and recommendation certificates for requestors so that every node on the TDT is a certificate issuer. Each TDT corresponds to one content

owner, but some nodes on a TDT may certify requests for multiple content owners. Any node on a TDT can also be a requestor.

Figure 4.1 depicts the Dynamic Distributed Trust Model.

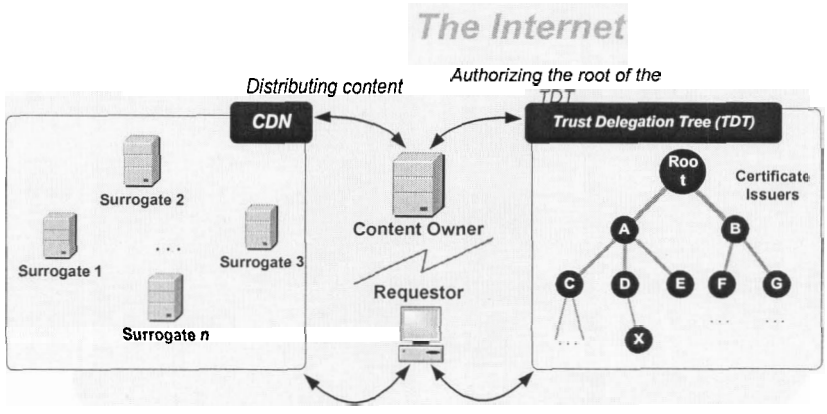


Figure 4.1: Dynamic Distributed Trust Model

The DDTM is not concerned with how a content owner delivers the content. Rather, it mainly focuses on providing a general-purpose, application-independent model that is suitable for access control for Internet applications. In this model, a requestor is granted a general and flexible trust value instead of specific access rights in a certificate. It is up to the content owner to decide what kind of access rights the requestor deserves based on the evaluation of the trust value presented. The above diagram is further simplified in Figure 4.2 by assuming that a requestor can communicate with a content owner directly instead of contacting the content owner's intermediate surrogates.

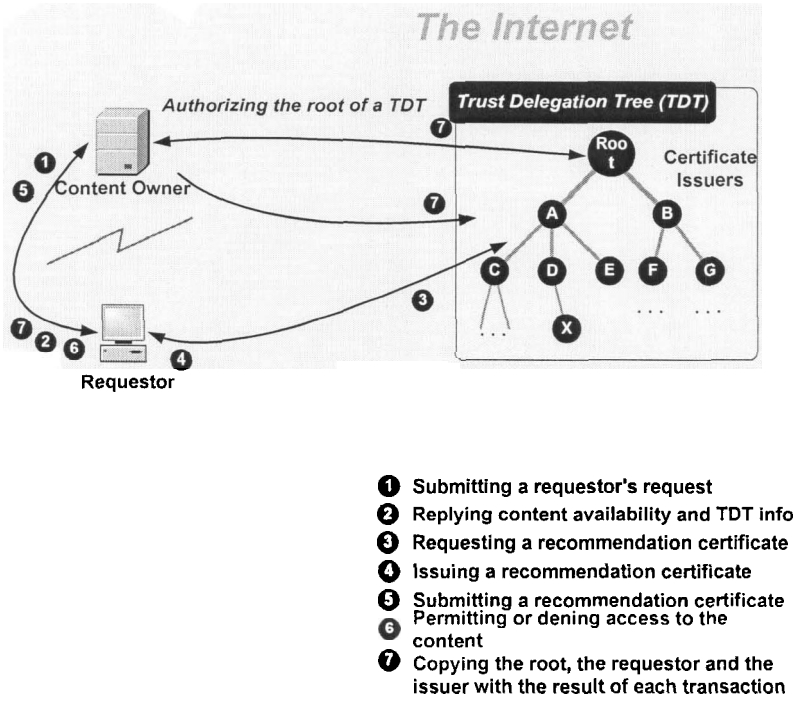


Figure 4.2: Simplified DDTM

4.2 Structure of A Trust Delegation Tree

4.2.1 Components of A Trust Delegation Tree

A Trust Delegation Tree (depicted in Figure 4.3), as proposed in this thesis, represents the levels of authorized certificate-issuing authority on behalf of a given content owner. There is one TDT representative of one content owner. Central to the TDT concept is the ability to distribute certificate-issuing authority, to manage the certificate-issuing nodes in a tree structure to achieve scalable authorization capacity, and to avoid random distribution of the certificate issuers.

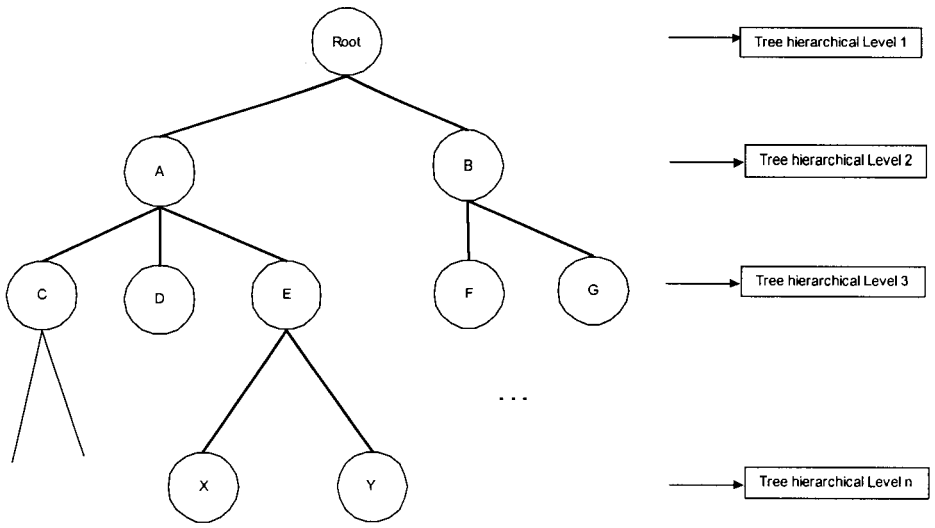


Figure 4. 3: The hierarchical structure of a Trust Delegation Tree

Before going into a detailed description of the TDT structure, several definitions need to be given here.

- **Authorized Root:**

The root is identified and authorized by a content owner, and functions on its behalf. It is the entity that the content owner knows about and trusts to take care of certificate-issuing matters. This root is the ultimate authority with respect to transaction history of all the nodes on the TDT.

- **Certificate-Issuing Nodes:**

These are the nodes on the Trust Delegation Tree to be used as the trusted intermediaries by a content owner. There is no centralized management for all the nodes on a TDT. Every node manages its own children and is managed by its parent.

- **Trust Authorization Level:**

This is the value that represents a content owner's belief in a node's recommendation. The highest authorization level is at the root. As a node nears the leaves on a TDT, its trust authorization level is decreased.

The TDT structure is the topology for representing a distributed and dynamic authorization capacity. Since the trust relationship between a content owner and its less well-known requestors is established through the root or through other nodes on the tree, the TDT provides a strong delegation certificate chain to introduce the less well-known requestors to the content owner. The root gains the highest level of trust from the content owner, and each level on the TDT under the root has a gradually decreasing authorization level from the content owner. The decreased authorization level along the TDT from top to bottom is expressed as:

$$\textit{Trust Authorization Level}: \quad aL(\textit{content owner}, \textit{node } i) = \prod_{i=1}^{n} (dT_i)$$

where $0 \le dT_i \le 1$, n is the hierarchical level of the nodes on the TDT.

A content owner gives the direct trust value dT_1 to the root. In the same way, each tree level (i) node assigns direct trust value dT_{i+1} to its trusted children. An example is shown in Figure 4.4.

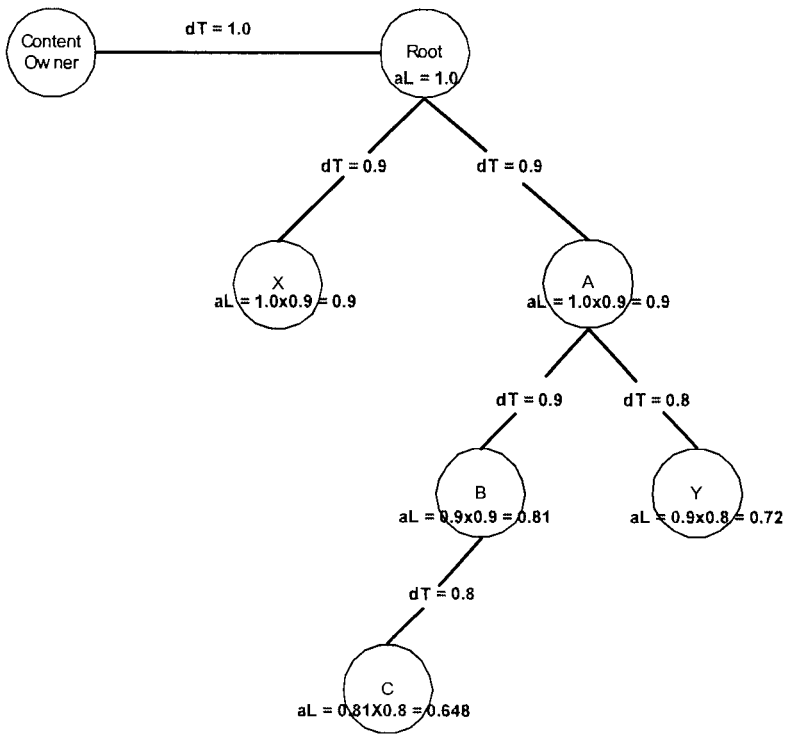


Figure 4.4 Example of the trust authorization level calculation

The TDT resembles the trust relationship presented in Figure 3.2 in a tree structure. All the nodes on the tree have a trust authorization level value associated with them. Therefore, the nodes can recommend other users that they directly trust to the content owner.

The TDT structure is not a fixed topology. A node can dynamically increase its authorization level by having more successful transactions directly with a content owner. On the other hand, a node can be dynamically degraded or removed if it engages in unsuccessful transactions, or by introducing malicious requestors.

4.3 Structure of Certificates

There are two types of trust expressing certificates in the DDTM: a *delegation certificate*, and a *recommendation certificate*. In both certificate types, an issuer binds the direct trust value with the subject's public key.

4.3.1 Delegation Certificate

The *delegation certificate* is a signed message to delegate authority from one entity to another and makes the distribution of certificate-issuing authority possible. The fields of the delegation certificate are: subject's public key, direct trust value, delegation, valid time and digital signature by the issuer's private key. The delegation field can be "*Yes*" or "*No*". If the delegation field is "*No*" for a subject node, the node cannot delegate authority any more. The structure of the delegation certificate is shown in Figure 4.5.

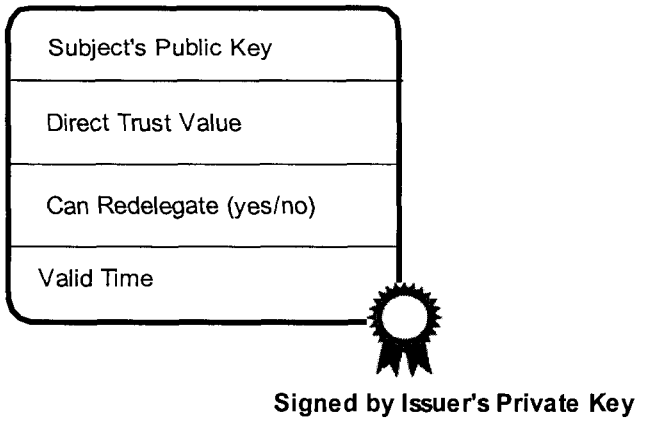


Figure 4.5: Structure of trust delegation certificate

The delegation certificate is used in a chain to convey trust values in a flow. There is no centralized trusted authority that can control the flow of the trust values, but each node of a TDT has the freedom to issue a delegation certificate to the children under its own control. The *Valid Time* field in the certificate defines the period for which a certificate is valid, and it cannot be longer than that of its parent node in the certificate chain. Figure 4.6 shows the trust delegation from a root to node C through nodes A and B.

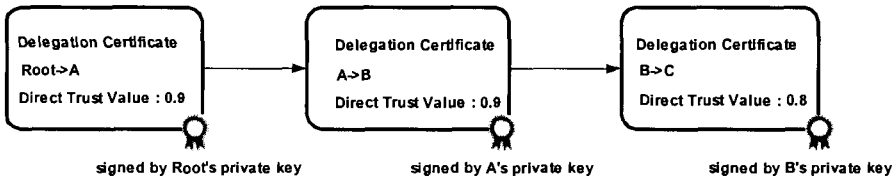


Figure 4.6: Structure of delegation certificate chain

Each node should record the delegation certificate chain from the root to itself in order to prove its trustworthiness. The authorization level of each node for a content owner is calculated through the delegation certificate chain by the following formula:

$$aL(content\ owner, node\ i) = \prod_{i=1}^{i} direct\ Trust\ Value\ in\ Delegation\ Certificate\ i$$

Therefore, by taking the example of Figure 4.6, assuming the content owner has the direct trust value of 1.0 on the root, the trust authorization level for node C is:

(0.9x0.9x0.8 = 0.648)

4.3.2 Recommendation Certificate

The recommendation certificate is issued only to authorize a requestor. Inside a recommendation certificate, the direct trust value and the delegation certificate chain should be provided from an issuer to a requestor. The certificate chain inside the recommendation certificate proves the certificate-issuing authority and the trust authorization level of the issuer node. The structure of the recommendation certificate is shown in Figure 4.7.

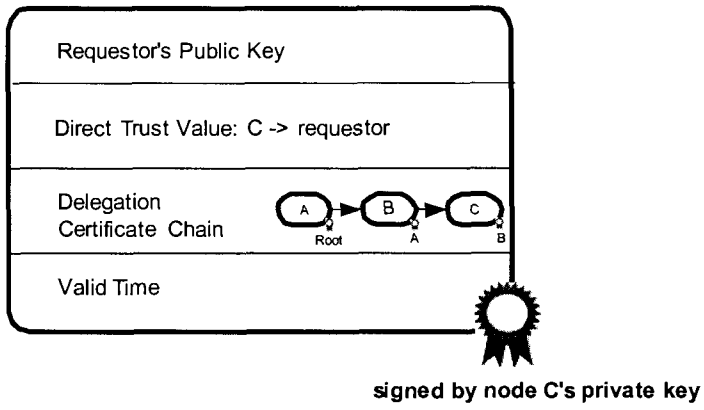


Figure 4.7: Structure of recommendation certificate

When a requestor submits a recommendation certificate to a content owner, the content owner will verify whether the delegation chain originates from the content owner's authorized root and ends at the issuer and whether the V*alid Time* in each certificate is still valid. After the recommendation certificate has passed all these checks, the content owner can do a trust evaluation of the requestor based on the issuer's authorization level and the requestor's direct trust value from the issuer.

4.4 The Algorithm for Obtaining Recommendation Certificates

In order for a requestor to access the resources of a content owner, the requestor needs to obtain a recommendation certificate to prove that it has a trust value greater or equal to the one required by the content owner.

Each entity is supposed to maintain the following information about its previous recommendation certificate issuer in a local "Issuer History Database":

- ID of content owner that issuer represents
- Issuer's ID and its authorization level (aL)
- Direct trust value (dT) given to the entity
- Recommendation certificate's valid time
- The recommendation certificate issued to the entity.

By using aL and dT, the trust value of the entity can be calculated.

The following example shows how to use the information in the "Issuer History Database" to obtain a recommendation certificate. Assume Alice is an issuer node of the TDT for content owner A, and it issues a recommendation certificate to Bob. Bob records Alice as its previous issuer for content owner A. Whenever Bob requests service from content owner A again, Alice is a possible recommendation certificate issuer that Bob can contact. Later, Alice may become an issuer node for other content owners, and Bob has Alice's information as its issuer in the "Issuer History Database". Bob also has the potential of getting a recommendation certificate from Alice for accessing other content owners. The pseudo code for a requestor obtaining a recommendation certificate is shown in Figure 4.8. Figure 4.9 shows the flow chart of the algorithm.

```
/* This algorithm is used for a requestor to obtain a recommendation
certificate. */
Obtaining_Recommendation_Certificate_Algorithm(){
  /* A requestor firstly finds its previous issuers.*/
  issuerSet = select all from "Issuer History Database" sort by trust
  value;

  /* If the requestor does not have any previous issuer that signed a
  recommendation certificate, the requestor will contact the root and the
  root may redirect the request to other nodes. */
  If (issuerSet == null){
    request to the root;
    return;
  }
  else{
    /*Check whether there is a still valid recommendation certificate that
    meets the desired trust value.*/
    if (content_owner_ID = desired_content_owner_ID and certificate is
    valid and trust_value >= desired_trust_Value)
       return the recommendation certificate;
    else
      /*Remove the valid cert. Of the required content owner ID from
      issuerSet to a new set*/
      VIS = getValidIssuerSet();

      /*Contact the previous issuers from the one with the highest trust
      value until the requestor obtains a recommendation certificate and
      satisfies with it. */
      while(issuerSet.cursor!= null){
        /*Contact the issuer that the cursor pointed and wait for reply*/
        cert = requestToIssuer();

        if (cert != null){
          if (getTrustValue(cert) > desired_trust_value)
            return cert;
          else
            /*Append the issuer with its recommendation certificate into
            VIS*/
            appendVIS(cert);
        }

        issuerSet.cursor = issuerSet.cursor→next;
      }
    }
  }


  if (VIS != null) {
    return the cert with the highest trust value from the VIS;
  }
  else{
    /*Contact the root when the requestor cannot obtain a recommendation
    certificate from all previous issuers.*/
    request to the root;
    return;
  }
}
```

Figure 4.8 Pseudo code for a requestor obtaining a recommendation certificate

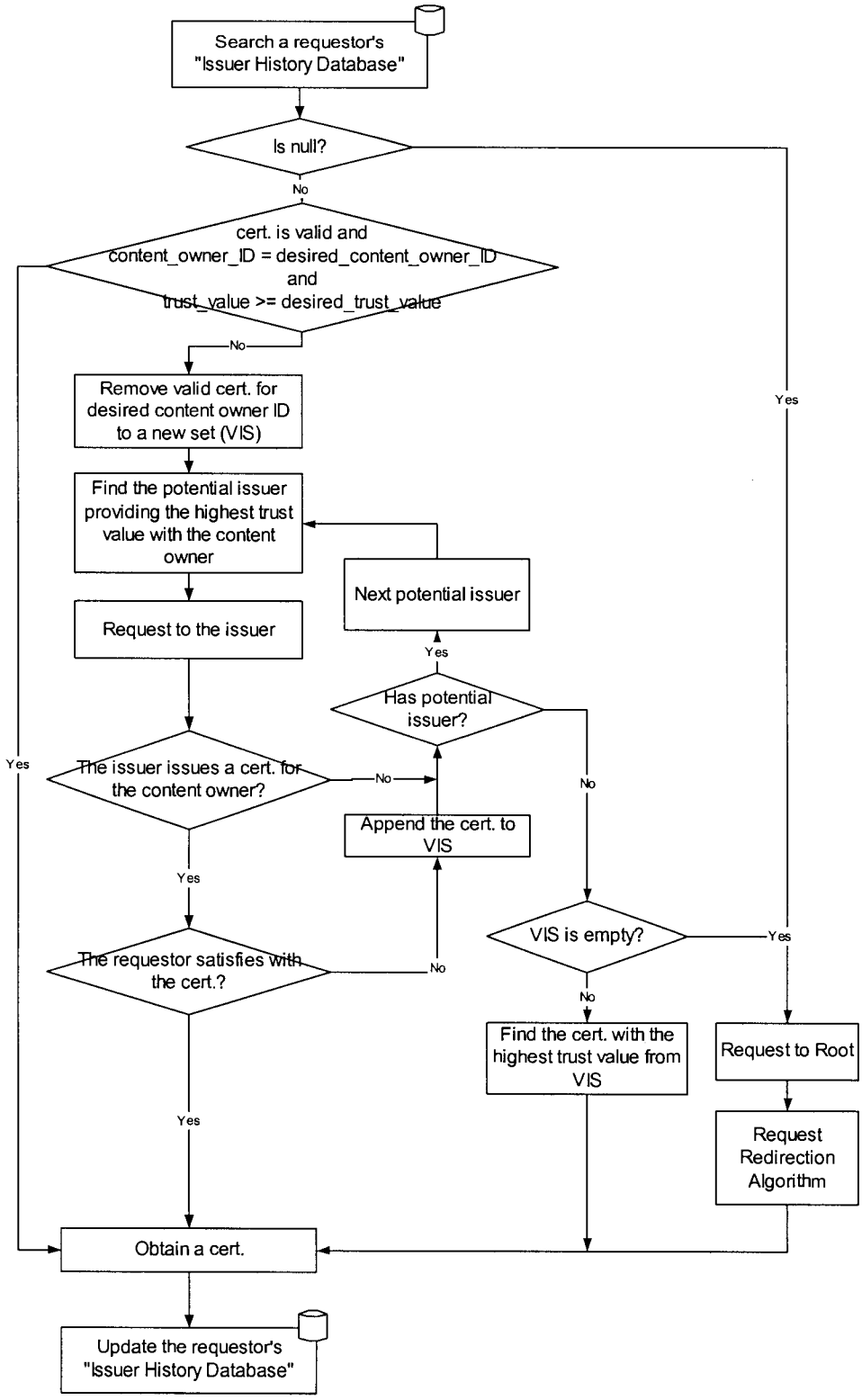


Figure 4.9: Obtaining a recommendation certificate algorithm flow chart

Since the previous issuers' information is stored in the "Issuer History Database" of each requestor, it can be searched in advance, and the requestor is able to contact the best of the previous issuers each time. Suppose there are n previous recommendation certificate issuers recorded at the requestor's side. In the worst case, it will cost the requestor $O(n)$ network messages for contacting the n previous issuers. In the best case, the requestor can directly use a valid recommendation certificate stored locally without any message traffic.

4.5 Operations of The Trust Delegation Tree

4.5.1 Initialization Operations

The initialization operations are aimed at generating a Trust Delegation Tree. The TDT is started from the root authorized by the content owner. After the root is established, the TDT grows by each node delegating trust authorization to other entities.

- Request Redirection Algorithm

This algorithm is for requestors that cannot directly get a recommendation certificate from any node of the TDT. For this kind of new requestor, their requests are redirected through the root to any node that has the capacity to accept more children for delegating authorization. Since the requestor does not have any trust relationship with the node that accepts the redirected request, the direct trust value inside the recommendation certificate that the node may give to the requestor could be zero. However, when the requestor requests a service later on, it can directly request a recommendation certificate from its previous issuer node and gradually build up a trust relationship with that node.

There are two parameters considered for each node in this algorithm, *tree_level* and *children_degree*. *Tree_level* cannot be greater than the global variable *THT (TDT Height Threshold)*, which is specified by a content owner. *Children_degree* is a variable for each node and is constrained by a local node's *CDT (Children Degree Threshold)*. The request redirection algorithm shown in Figure 4.10 is able to let the TDT grow evenly: the extreme situation in which all nodes are added along a long

single path is avoided by letting a non-leaf node under the *CDT* accept a redirected request; a situation in which a single node is overloaded by recommendation certificate requests is avoided by not letting nodes with the maximum *children_degree* value accept redirected requests.

Figure 4.10 shows the pseudo code of the request redirection algorithm, and the flow chart for this algorithm is shown in Figure 4.11.

```
/* This algorithm is used when a root or a node receives a request for issuing
a recommendation certificate. */
Request_Redirection_Algorithm(){
  If (children_degree < CDT){
      return a recommendation certificate;
  }
  else{
    if( tree_level < THT ){
      /* The request is redirected to a child of the current root or node.*/
       redirectToChild();
       return;
    }
    else{
      /* When the node's tree level is equal to the THT, the node must issue a
      recommendation certificate to the requestor.*/
      return a recommendation certificate;
    }
  }
}
```

Figure 4.10 Pseudo code for a root or node running the request redirection algorithm

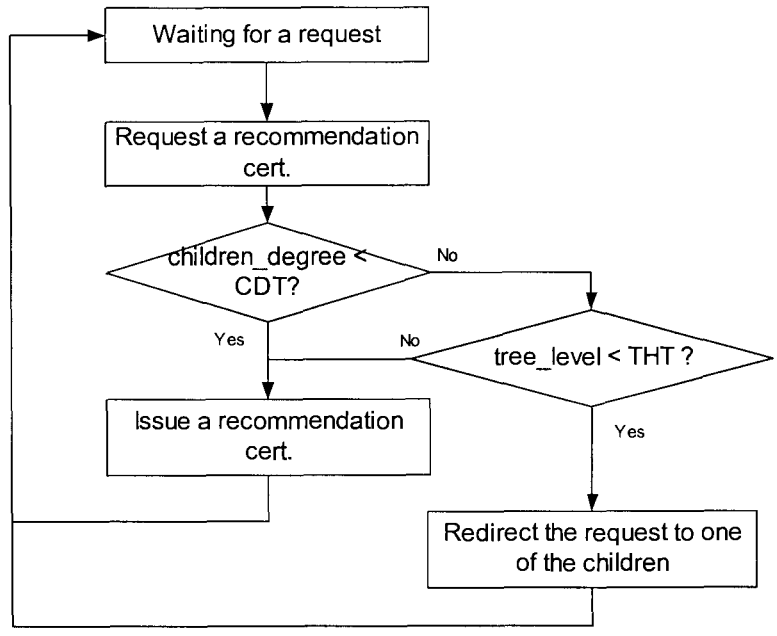


Figure 4.11: Request redirection algorithm flow chart

- Delegation Algorithm

The delegation algorithm enables an entity to be a node on a TDT by issuing a delegation certificate to the entity. Thereafter, the entity can issue recommendation certificates on behalf of a content owner. The *Delegation Threshold (DT)* is compared with an entity's direct trust value to decide whether the entity can be delegated certificate-issuing authority.

The pseudo code of how a node delegates certificate-issuing authority to another entity is shown in Figure 4.12, and the corresponding flow chart is shown in Figure 4.13.

```
/* This algorithm is used for a node to delegate authority to another
entity. */
Delegation_Algorithm(){

  /* Check the entity's direct_trust_value with the node's delegation
  threshold. */
  if (direct_trust_value >= DT){
    /* Check the node's number of children with and the node's Children
    Limit Threshold. */
    if (children_degree < CDT){
      if (the entity would like to be the node){
        /* The node issues a delegation certificate to the entity.*/
        issue a delegation certificate to the entity;
      }
      return;
    }

    else{
      /*Find the smallest direct trust value of the node's existing
      children.*/
      direct_trust_value_smallest  = findSmallestDirectTrustValue();
      /* Compare the node's direct_trust_value of the entity with the
      smallest direct trust value of the existing child. */
      if (direct_trust_value > direct_trust_value_smallest ){
        if (the entity would like to be the node){
          issue a delegation certificate to the entity;
          /*Remove the child with direct_trust_value_smallest*/
          removeChild(child_direct_trust_value_smallest);
          /*Recommend the child with direct_trust_value_smallest to one of
          the node's other children to accept it as a child*/
          recommendChild(child_direct_trust_value_smallest);
        }
      }
      else {
        /*Recommend the entity to one of the node's children to accept it as
        a child*/
        recommendChild(entity);
      }
      return;
    }
  }
```

Figure 4.12: Pseudo code for a root or node running the delegation algorithm

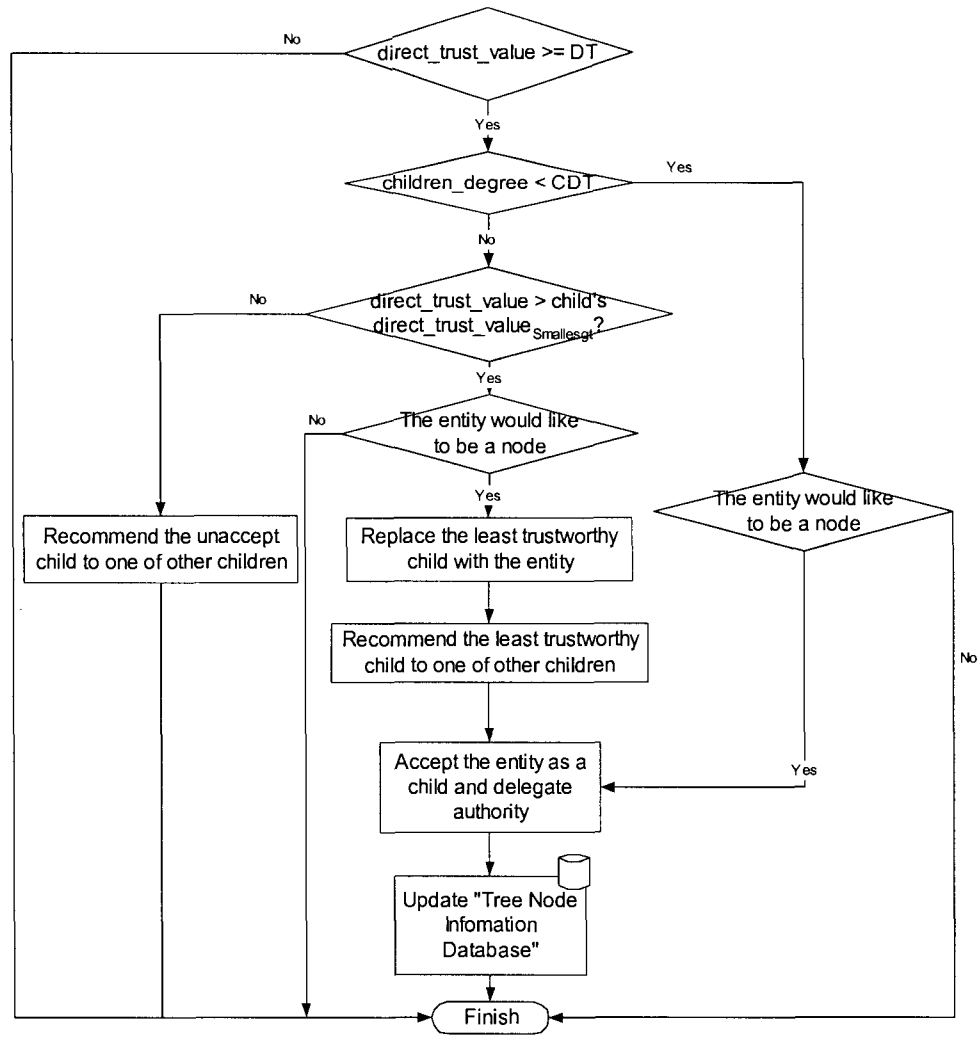


Figure 4.13 Delegation algorithm flow chart

4.5.2 Validation Operations

The validation operations are intended to make sure the delegation certificate of each node is valid. Each node of the TDT needs to obtain a new delegation certificate before its old one expires. In order to send a message only when necessary, each node is responsible for requesting its parent to update its delegation situation.

As shown in Figure 4.14, the parent node sends refreshed delegation certificates to its children when they are required.

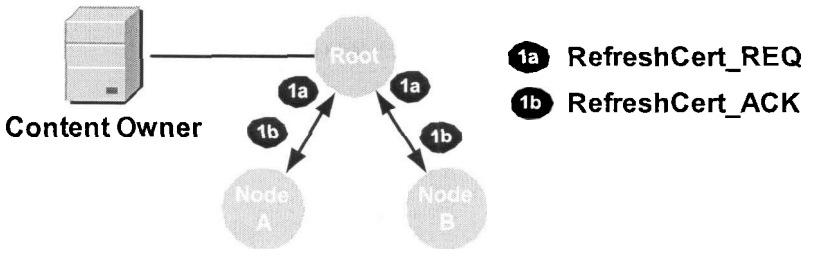


Figure 4.14: The operations for keeping nodes on the TDT valid

4.5.3 Reliability Operations

The purpose of the reliability operations on the TDT is to ensure that any node failure will not affect the working of other nodes. If a node crashes, all its children will ask to be added below one of the crashed node's siblings.

The following figure shows how to handle a node failure (Figure 4.15).

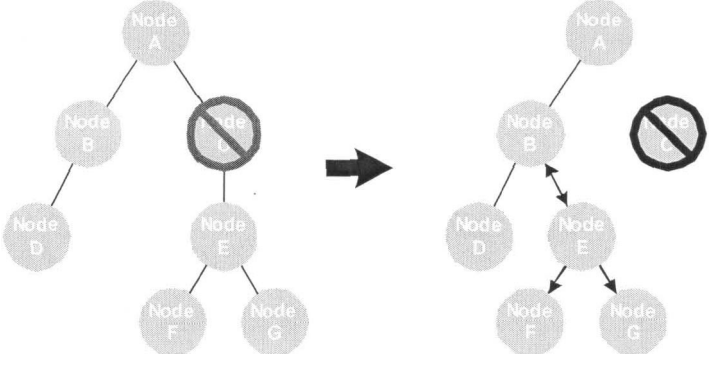


Figure 4.15: Changing a parent when a parent node crashes

4.5.4 Upgrade, Degradation and Remove Operations

A TDT represents the trust relationship and authorization hierarchy of a content owner. Any node of the TDT can be dynamically upgraded, downgraded or completely removed.

Upgrading a node and its sub-tree

First, a root can upgrade a node's authorization level of its TDT, since the content owner copies all the transaction information to its TDT root. The root will know about all the transactions that the content owner has with other entities. If a node frequently interacts with the content owner, it may have a higher direct trust value than its parent's authorization level on the TDT. Therefore, this node can get promoted directly by the root to an appropriate authorization level on the TDT that reflects its direct trust value.

This upgrading operation is shown in Figure 4.16.

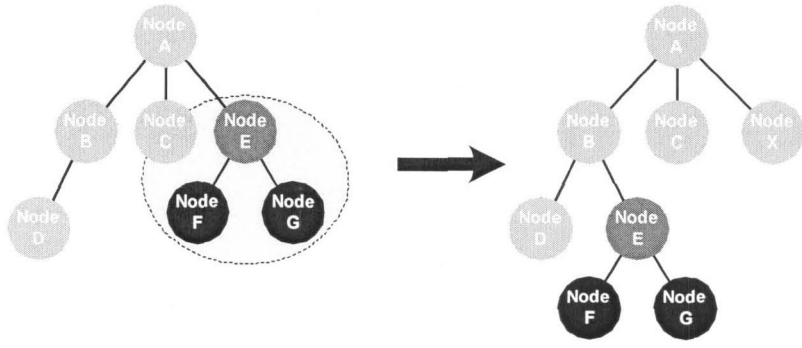


Figure 4.16: Upgrading the node E and its sub-tree operation

Second, a node may be upgraded by being added to a higher authorization level node. For each content owner's TDT, whenever a node is added to another parent, it must detach from its previous parent.

Degrading a node and its sub-tree

Since there is a limit to the maximum number of children that a node can have, the node may degrade one of its children in order to make room to accepte a node with a higher direct trust value.

The degradation operation is shown in Figure 4.17.

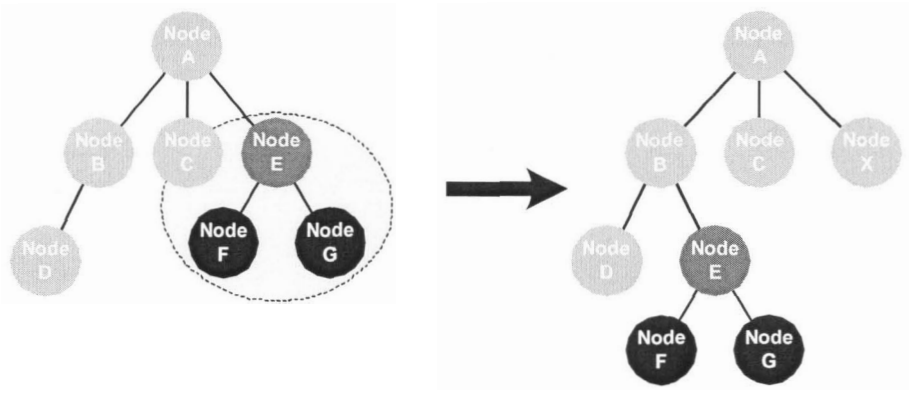


Figure 4.17: Degrading the node E and its sub-tree operation

- Algorithm for removing a node and its sub-tree

Based on the transaction history, a content owner is able to detect any malicious activities by a requestor and to maintain a black list of such entities. The content owner passes the black list to the TDT periodically. If a node finds that one of its children is on the list, the node removes this child and the child's sub-tree completely. The sub-tree under the compromised node is also removed because all these nodes are delegated certificate-issuing authority through the parent node that is un-trustworthy.

4.6 Scenario for Building Up A TDT

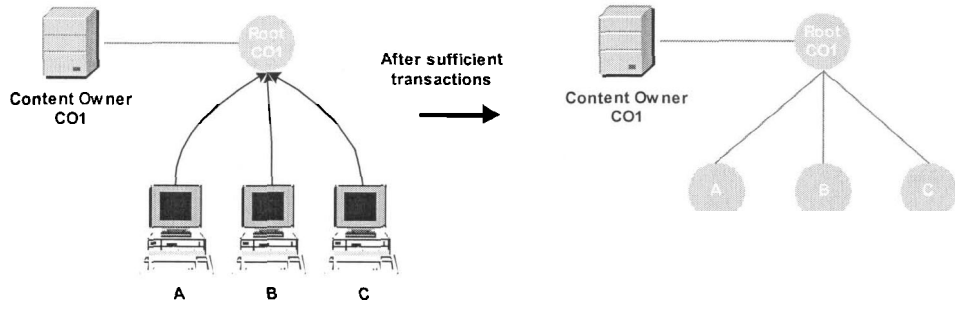
A content owner directly authorizes the root of a TDT, and every node on the TDT may authorize other entities to be its children. As soon as the TDT is established, the nodes on the TDT can issue recommendation certificates to requestors. Every node can have its own policy to decide how much it trusts another entity, but the content owner will constrain this value by the certificate-issuing node's trust authorization level and other subjective factors that may concern the content owner.

The following is a scenario showing how a TDT is built up, starting with a root and new requestors.

Content owner CO1 authorizes a root, RootCO1, for its TDT. Suppose the degree of the root node is 3 at the maximum. Requestors A, B, C and D are new requestors, which means that their "Issuer History Databases" are empty. When requestors A, B, C initially

send their first content requests to CO1, CO1 directs them to RootCO1 for obtaining certificates. The obtained recommendation certificates starting with initial direct trust value of 0 from RootCO1. After sufficient transactions, A, B and C can become the children of RootCO1.

Figure 4.18 shows the scenario of requestors A, B, C contacting RootCO1 for recommendation certificates and becoming the children of RootCO1.



Legend:

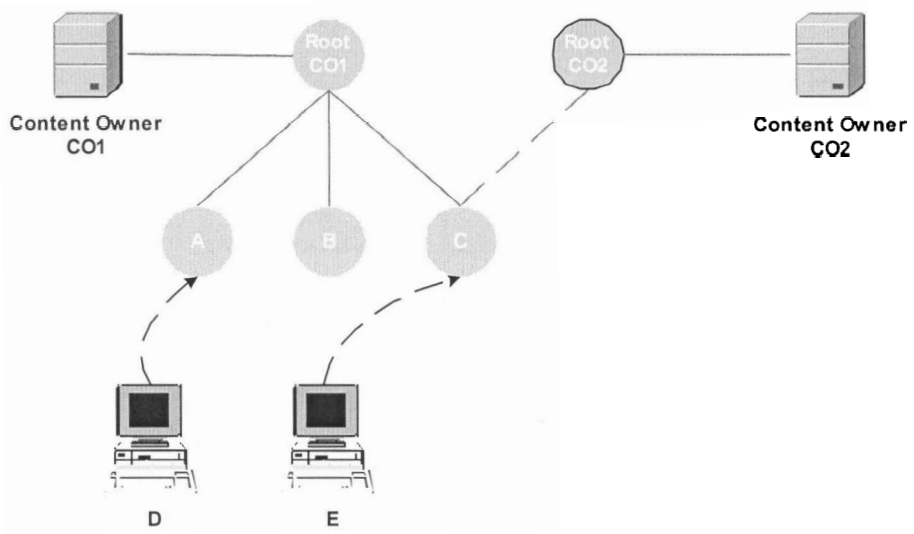
———	Tree connection in CO1's TDT
- - - - - - ▶	Request for a recommendation certificate

Figure 4.18: Requestors A, B and C become the nodes of CO1's TDT

When the maximum degree of RootCO1 is reached, a new requestor D may be directed to node A (Figure 4.19).

Assume node C is also a node on the TDT of the content owner CO2, and a requestor E has previous history with node C when C certifies E for accessing CO2. Thereafter, whenever requestor E requests a recommendation certificate, node C is more likely to be contacted. In this scenario, when requestor E contacts node C to request a recommendation certificate for accessing CO1, node C is able to issue a recommendation certificate to E with the direct trust value based on their previous transaction history.

Figure 4.19 shows the scenario of node C certifying requestor E.



Legend:

———	Tree connection in CO1's TDT
- - - - - - -	Tree connection in CO2's TDT
- - - - - - ▸	Request for a recommendation certificate

Figure 4.19: Requestor E contacts node C to request a recommendation certificate for accessing CO1

Chapter 5

Dynamic Distributed Trust Protocol Specification

The Dynamic Distributed Trust Protocol (DDTP) includes a set of protocols that govern the communication between a content-owner, requestors and certificate issuers. It is a general-purpose and flexible protocol that addresses only the trust value and not specific access rights associated with each requestor. It can be considered as a protocol framework to support the Dynamic Distributed Trust Model (DDTM).

In the DDTP, requestors and certificate issuers could be anywhere on the Internet. Considering the example of online movies, the movie provider needs to have confidence that a viewer will obey the regulations for viewing a movie. It is not possible for a content owner to know all the possible requestors in advance. Therefore, the DDTP establishes the mechanism to help a content owner to evaluate to what extent it can trust a requestor.

The DDTP in this thesis provides a means of helping a content owner to obtain confidence in a requestor. However, the DDTP can be used in many other applications, such as online access control within a geographically distributed enterprise, or resource sharing among collaborating enterprises.

5.1 Data Specification

There are different categories of data that are necessary to be defined and recorded at each entity, depending on whether the entity is a content owner, a certificate issuer or a requestor in the Dynamic Distributed Trust Protocol.

- **Issuer History Database**

Records the information about a requestor's previous issuers of recommendation certificates. The following indicates the information needed for a previous recommendation certificate issuer for each requestor:

Content Owner ID	*Issuer ID*	*Issuer's aL*	*Direct Trust Value*	*Valid Time*	*Recommendation Certificate*

- **Transaction History Database**

Records the information about the previous transactions of an entity with all other users. The Transactions History Database is kept at an issuer node, a content owner and the content owner's root. The information that is needed for transaction history is shown as following:

Requestor's ID	*Requestor's Issuer ID*	*Issuer's aL*	*Number of Transactions*	*Transaction Cost*	*Direct Trust Value*

- **Tree Node Information Database**

Records the information about a node as a certificate issuer on a Trust Delegation Tree.

Content Owner ID	*aL*	*Parent*	*Children List*	*Parent's Sibling List*	*Sibling List*	*Valid Time*	*Delegation Cert Chain*

- **TDT Global Threshold and Parameters**

TDT global threshold and parameters are defined by a content owner and passed to the root as well as to other nodes.

TDT Height Threshold (THT): TDT height threshold is a global threshold specified by a content owner to define the height of its TDT. Therefore, every node on a TDT must know the value of *THT* and will not add a child node at level *THT + 1* of a TDT.

Black List (BL): It is a content owner's responsibility to collect a list of misbehaving requestors or certificate issuers (by monitoring the frequency of transaction during a period of *time* and/or the routing through trusted or un-trusted intermediaries [Man98]) and to propagate the list to all of the certificate-issuing nodes on its TDT. In this way, a recognized malicious entity will never be able to be a node on a TDT.

- **Node's Local Thresholds and Parameters**

The following thresholds and parameters are defined by each node. This gives the flexibility to each node to handle certificate request according to the node's capacity.

Children Degree Threshold (CDT): Each node on a TDT decides its maximum degree of children that the node will delegate authority to.

Delegation Threshold (DT): Each node decides the threshold of direct trust value in order to delegate certificate-issuing authority to other nodes.

Other parameters, such as *Renew_Timeout* and *Kickoff_Timeout*, are used to set up timeout values for unacknowledged messages. *Renew_Timeout* is the bound time of a node waiting for its delegation certificate renewal from its parent; *Kickoff_Timeout* is the bound time of a detached node waiting for another node to take it as a child.

5.2 Protocol Specification

The DDTP is a set of protocols operating over a reliable transport protocol connection. The DDTP includes an authorization protocol, a content requisition protocol, a recommendation certificate requisition protocol and a TDT updating protocol. Those four

protocols work together to provide an authorization for a requestor to use certain content or services of a content owner (Figure 5.1).

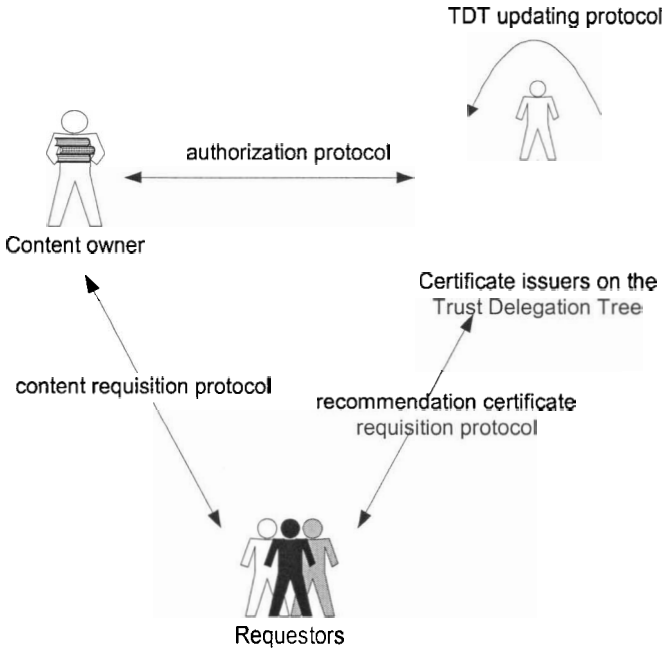


Figure 5.1: DDTP scope

5.2.1 The Authorization Protocol

The authorization protocol operates between a content owner and the potential root of the content owner's TDT. Its aim is to let the content owner authorize the root and pass on the information of the content owner to the TDT, so that every node on the TDT can work as a certificate issuer for the content owner. Figure 5.2 shows the message sequence in this authorization protocol.

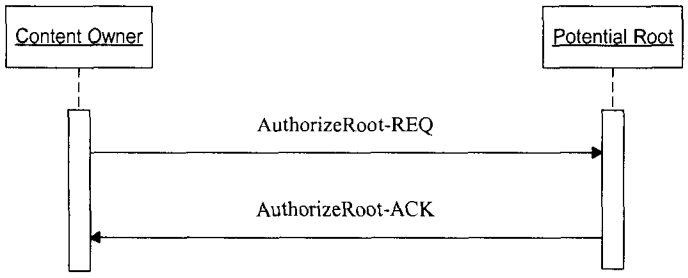


Figure 5.2: Message sequence in the authorization protocol

5.2.1.1 Vocabulary of Messages

Message Type: AuthorizeRoot-REQ
Data Field(s):

1. Content owner's ID
2. Authorization level
3. The TDT global threshold and parameters specified by the content owner

Message Description: By sending an AuthorizeRoot-REQ message, the content owner requests an entity to take responsibility as the root of its TDT. In data field 1, the content owner identifies itself to the potential root. The content owner specifies the authorization level of the potential root in field 2 and the TDT global threshold and parameters in field 3.

Receiver Actions: If the entity that receives an AuthorizeRoot-REQ message consents to be the root, it will update its "Tree Node Information Database" using the parameters in the data fields and sends an AuthorizedRoot-ACK message.

Message Type: AuthorizeRoot-ACK
Data Field(s):

1. Acceptance: true/false

Message Description: The entity that receives an AuthorizeRoot-REQ message responds to the content owner with an AuthorizeRoot-ACK message to either accept or reject being the root totally.

Receiver Actions: If the received AuthorizeRoot-ACK message advises that the entity isn't able to be the root, the content owner will send an

AuthorizeRoot-REQ message to another entity to request it to be a root.

5.2.2 The Content Requisition Protocol

This protocol mainly operates between a requestor and a content owner. There are two phases in this protocol: in phase one, a requestor submits its request and the content owner responds with the root information and the desired trust value; then in phase two, the requestor submits a recommendation certificate, and the content owner decides whether to grant the requested content to the requestor or not. After the transaction with the requestor is done, the content owner sends its transaction receipt to the root, the recommendation certificate issuer of the requestor and the requestor. Figure 5.3 shows the message sequence in this content requisition protocol.

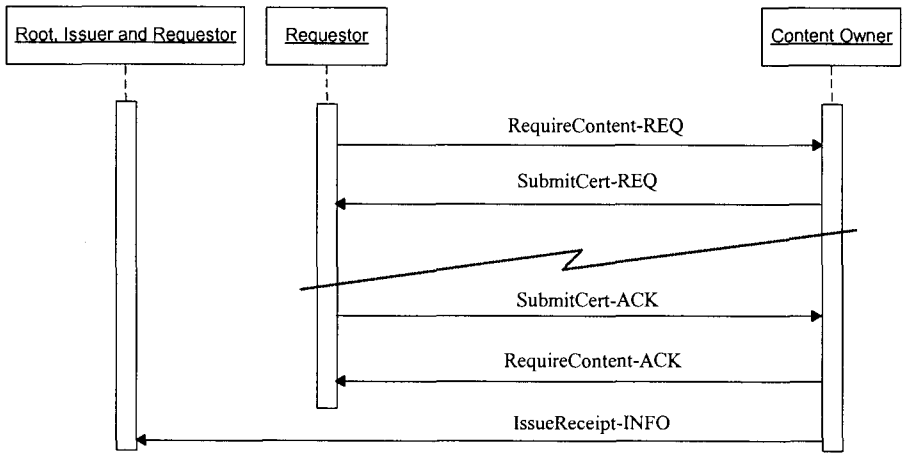


Figure 5.3: Message sequence in the content requisition protocol

5.2.2.1 Vocabulary of Messages

Message Type: **RequireContent-REQ**
Data Field(s):

1. Content item ID

Message Description: A requestor initializes a request for content by sending a RequireContent-REQ message to the content owner. The requestor specifies which content it requests in the data field.

Receiver Actions: The content owner that receives a RequireContent-REQ message will ask the requestor to provide a recommendation certificate.

Message Type: **RequireContent-ACK**
Data Field(s):

1. Granting Flag: true/false

Message Description: When a content owner receives a recommendation certificate from a requestor, the content owner will send a RequireContent-ACK message to the requestor to indicate whether to grant the content or not.

Receiver Actions: When a requestor receives a RequireContent-ACK message, the requestor finishes its request.

Message Type: **SubmitCert-REQ**
Data Field(s):

1. Root
2. Desired trust value

Message Description: Whenever a content owner receives a request for content, the content owner sends a SubmitCert-REQ message with its root information and the desired trust value to the requestor.

Receiver Actions: When a requestor receives a SubmitCert-REQ message, the requestor will request a recommendation certificate to meet the desired trust value as much as possible.

Message Type: **SubmitCert-ACK**
Data Field(s):

1. Issuing flag: true/false
2. A recommendation certificate

Message Description: When a requestor receives a recommendation certificate from a desired content owner's certificate issuer and decides to submit it, a SubmitCert-ACK message is sent from the requestor to the content owner. The recommendation certificate is included in the data field of the message.

Receiver Actions: When a content owner receives a SubmitCert-ACK message, the content owner will check the recommendation certificate inside the message and decide whether to grant the content to the requestor or not.

Message Type: **IssueReceipt-INFO**
Data Field(s):

1. Requestor ID
2. Transaction receipt

Message Description: After a content owner finishes a transaction with a requestor, the content owner will issue a transaction receipt to the root, the recommendation certificate issuer of the requestor and the requestor.

Receiver Actions: When an issuer receives an IssueReceipt-INFO message, the issuer will record this transaction receipt in order to evaluate the same requestor later on. The root and the requestor that receives an IssueReceipt-INFO message keep the transaction information for record.

5.2.3 The Recommendation Certificate Requisition Protocol

The recommendation certificate requisition protocol is used between a requestor and the root or any node on a TDT to help the requestor to obtain a recommendation certificate. Figure 5.4 shows the message sequence in this protocol.

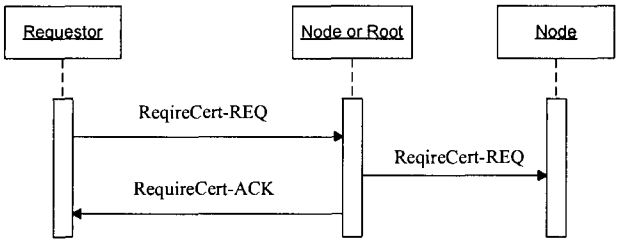


Figure 5.4: Message sequence in the recommendation certificate requisition protocol

5.2.3.1 Vocabulary of Messages

Message Type: **RequireCert-REQ**
Data Field(s):

1. Content owner's ID
2. Desired trust value

Message Description: A content requestor looks for previous recommendation certificate issuers by searching its "Issuer History Database". The requestor may send a RequireCert-REQ message with a specified content owner ID and desired trust value to any previous issuer to request a recommendation certificate.

Receiver Actions: The entity that receives a RequireCert-REQ message checks whether its delegation certificate is valid.

- If it is valid, the entity will calculate the trust value for the requestor based on the entity's "Transaction History Database" and will issue a recommendation certificate with a new valid time in a RequireCert-ACK message. If the calculated trust value is beyond this entity's *Delegation Threshold (DT)*, the entity will also send an AddChild-REQ message to the requestor.
- If the entity's delegation certificate is not valid, the entity will send a RenewCert-REQ message to its parent to renew its delegation certificate. If the entity is not able to renew its delegation certificate, the entity sends a RequireCert-ACK message without any signed recommendation certificate.

Message Type: **RequireCert-ACK**

Data Field(s):

1. Issuing Flag: true/false
2. Recommendation certificate

Message Description: Any node that receives a request for a recommendation certificate sends back a RequireCert-ACK message either with the signed recommendation certificate or with a rejection. The Issuing Flag indicates whether the recommendation certificate is issued or not.

Receiver Actions: If a requestor gets a satisfied recommendation certificate, the requestor will send a RequireContent-ACK message to the content owner; otherwise, the requestor will send a RequireCert-REQ message to another potential issuer until it gets a recommendation certificate.

5.2.4 The TDT Updating Protocol

The TDT updating protocol aims to adjust the relationship of the nodes on a TDT. Nodes are added or removed dynamically in order to represent a proper trust hierarchy for a content owner. Figure 5.5 shows the message sequence in this protocol.

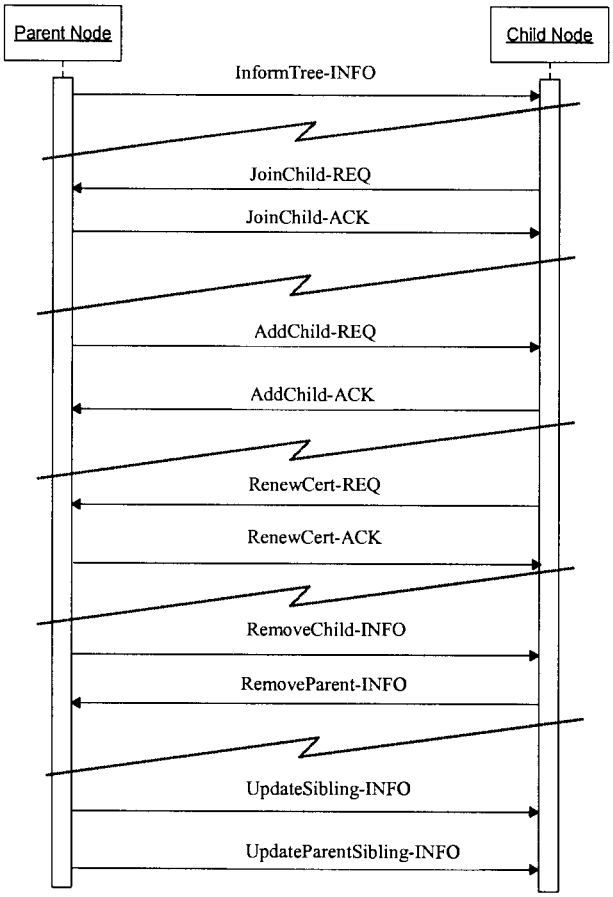


Figure 5.5: Message sequence in the TDT updating protocol

5.2.4.1 Vocabulary of Messages

Message Type: **JoinChild-REQ**
Data Field(s):

1. The evidence for joining a child. This could be:
- Either the direct trust value of the child with the content owner.
- Or the child's delegation certificate issued from its parent.

Message Description: A JoinChild-REQ message is sent to the node that will become a parent. There are two situations where an entity may send a JoinChild-REQ message:

- The root sends a JoinChild-REQ message when it directly promotes a node to a higher authorization level. The message is sent to one of its children that have a higher trust authorization level than the direct trust value of the node with the content

owner. The child will handle it accordingly. The node's direct trust value with the content owner is in the field 1.

- In another case, any node may send a JoinChild-REQ message to one of its parent's siblings when the parent is not available or may have crashed. The field 1 is the node's delegation certificate issued from its parent.

Receiver Actions: Whenever a node receives a JoinChild-REQ message, the node checks whether it has room to take a new child or whether it would replace any existing child with this new child. If not, the node will just continuously send a JoinChild-REQ message to one of its children. If the node will accept the new child, the node will update its own "Tree Node Information Database" and send an UpdateSibling-INFO message to all its children to let them update their sibling list. Finally, the node will send a JoinChild-ACK message to the new added child.

Message Type: **JoinChild-ACK**
Data Field(s):

1. Acceptance: true/false
2. Content Owner's ID
3. The TDT global threshold and parameters specified by the content owner.
4. Authorization Level
5. Parent's ID
6. Parent's Sibling list
7. A delegation certificate chain

Message Description: Whenever a node adds a new child, the node sends a JoinChild-ACK message to inform the potential child about whether it is added as a child and all the information needed for being a node.

Receiver Actions: The entity that receives a JoinChild-ACK message will updates its "Tree Node Information Database" with the parameters in the data fields.

Message Type: **AddChild-REQ**
Data Field(s):

1. Content Owner's ID
2. The TDT global threshold and parameters specified by the content owner
3. Authorization level
4. Parent's ID
5. Parent's sibling list
6. A delegation certificate chain

Message Description: An entity may send an AddChild-REQ message on its own initiative to take on another entity as its child. When a node has a direct trust value on some entity beyond the node's *Delegation Threshold (DT),* the node indicates that it would like to take on this entity to be a child by sending an AddChild-REQ message to the entity.

Receiver Actions: If the entity that receives an AddChild-REQ would like to be the child of the sender, it will reply with an AddChild-ACK message and update its "Tree Node Information Database" with the information in the data fields. If this entity has previous parent in this TDT, the entity will send a RemoveParent-INFO message.

Message Type: **AddChild-ACK**
Data Field(s):

1. Acceptance: True/False

Message Description: An entity that receives an AddChild-REQ message confirms with the sender whether it would like to be the child or not and updates its "Tree Node Information Database" accordingly.

Receiver Actions: If the acceptance field in an AddChild-ACK message is true, the entity that receives this message will update its "Tree Node Information Database" and send an UpdateSibling-REQ message to all its children to let them update their sibling list. If the entity has replaced one of its children with the new child, it sends JoinChild-REQ to one of its children to add the replaced child.

Message Type: **RemoveChild-INFO**
Data Field(s):

1. Content owner's ID

Message Description: This message is sent out under the following two situations:

- An entity receives a RenewCert-REQ message for renewing a delegation certificate from its child, but the entity will not grant the delegation certificate to the message sender any more. In this case, the entity sends RemoveChild-INFO to this child and updates its own "Tree Node Information Database" by deleting this child information.
- To limit the number of its children, an entity A may replace one of its children B with another entity C who has a higher direct trust value. At this point, the entity A sends RemoveChild-INFO to the replaced child B and recommends the entity B to one of A's children by sending JoinChild-REQ.

Receiver Actions: The entity that receives RemoveChild-INFO will delete the issuer's record in "Issuer History Database" and update its "Tree Node Information Database" for the specified content owner ID.

Message Type: **RemoveParent-INFO**
Data Field(s):

1. Content owner's ID

Message Description: When a child node consents to be a child of a new parent, this node sends a RemoveParent-INFO message to inform its old parent of its leaving.

Receiver Actions: The entity that receives RemoveParent-INFO will update its "Tree Node Information Database" for the specified content owner ID.

Message Type: **RenewCert-REQ**
Data Field(s):

Message Description: Each time a delegation certificate expires, the entity sends a RenewCert-REQ message to its parent to renew the delegation certificate.

Receiver Actions: When an entity receives a RenewCert message, it will check with the blacklist. If the sender is not on the blacklist, the entity will re-calculate the direct trust value for the sender and issue a new delegation certificate with a valid time. If the sender's ID is in the blacklist, the entity will not issue a delegation certificate to the sender any more.

Message Type: **RenewCert-ACK**
Data Field(s):

1. Issuing Flag: True/False
2. Delegation Certificate

Message Description: An entity replies to the RenewCert-REQ message sender with a renewed delegation certificate or nothing.

Receiver Actions: The RenewCert-ACK receiver will update its "Issuer History Database" and "Tree Node Information Database" with the renewed delegation certificate or does nothing without a renewed delegation certificate.

Message Type: **UpdateSibling-INFO**
Data Field(s):

1. Add/Remove
2. Added or removed child's ID

Message Description: Whenever an entity adds or removes a child, it sends out UpdateSibling-INFO messages to all its children to let them update their sibling list.

Receiver Actions: When an entity receives an UpdataSibling-INFO message, it will update its sibling list in the "Tree Node Information Database" by adding or removing the child's ID in message data field 2. If this entity still has a child, the entity will send an UpdateParentSibling-INFO message to its children to let them update their parent's sibling list.

Message Type: **UpdateParentSibling-INFO**

Data Field(s):

1. Add/Remove
2. Added or removed sibling's ID

Message Description: Whenever an entity receives an UpdateSibling-INFO message, if this entity still has a child, it sends UpdateParentSibling-INFO messages to all of its children to let them update their parent sibling's list.

Receiver Actions: When an entity receives an UpdateParentSibling-INFO message, it updates its parent's sibling list in the "Tree Node Information Database" by adding or removing the parent sibling's ID that is indicated in message data field 2.

Message Type: **InformTree-INFO**

Data Field(s):

1. Content owner ID
2. The TDT global threshold and parameters specified by the content owner.

Message Description: Whenever the global threshold is changed or a misbehaving requestor or certificate issuer is caught, a content owner will flood InformTree-INFO messages from the root to every node on a TDT.

Receiver Actions: When an entity receives an InformTree-INFO message, it will update the global threshold and parameters about the content owner.

Chapter 6

Validation of The Dynamic Distributed Trust Protocol

This chapter describes how to use SPIN to verify the Dynamic Distributed Trust Protocol (DDTP). SPIN is a tool used for the formal verification of distributed software systems. SPIN was first developed at Bell Labs in 1980, and was awarded the prestigious System Software Award for 2001 by the ACM.

6.1 SPIN Overview

SPIN (Simple Promela INterpreter) [Hol03] is a tool for analyzing asynchronous systems specified in the language PROMELA and leads to the detection of errors in the designed systems. Its verification models are focused on proving the correctness of process interactions and attempt to abstract as much as possible from internal sequential computations. PROMELA (PROcess Meta LAnguage) is the modeling language used to describe the concurrent (distributed) systems that SPIN accepts.

SPIN originated from the earliest protocol verification systems based on on-the-fly reachability analysis from the 1980s. However, the predecessors of SPIN at that time only supported the verification of standard safety properties and a limited range of liveness. The new generation of model checking tool is expanded with an automata theoretic model, and the automata theoretic model has become the formal basis for Linear Temporal Logic (TLT) model checking in the SPIN system.

6.1.1 Basic Functionalities of SPIN

- Simulation Mode

By performing the simulation, SPIN lets users gain basic confidence about the model's intended properties. There are three options for SPIN simulation: random, interactive and guided. Random simulation may produce a different execution at each run; interactive simulation lets the user select at each step to resolve non-deterministic choices in the model; guided simulation requires a trail file, which is generated when the verifier discovers a correctness violation, to indicate the execution sequence to the detected error.

- Verification Mode

SPIN can generate a verifier to search for violation of safety properties (assertion violations, deadlocks, etc.) or for liveness properties (e.g., to show the absence of non-progress cycles or acceptance cycles). The verifier could have an optimized on-the-fly verification with possible compile-time choices for the types of reduction algorithm. Partial order reduction is default in SPIN to avoid exploring redundant computation paths in the model due to interleaved executions of independent events. Verification can be done in exhaustive method to store each state in the memory, but it is is not suitable for large verification problems due to the state explosion problem. Bit-state hashing is an efficient approximation to increase the problem coverage when the state space exceeds the available memory.

6.2 DDTP State Machine

The DDTP provides communication among a content owner process, a root process, requestor processes and node processes, and aims to provide access control for the content at the content owner process.

6.2.1 State Diagram for The Content Owner Process

The complete behavior model for a content owner process is presented in state transition diagrams in Figure 6.1. The states of the content owner process are summarized as the follows:

- Start state:

An entity starts to take the content owner's role at this state. In this state, the content owner sends an AuthorizedRoot-REQ message to request another entity to be its root and waits for the reply indicating whether the entity would like to be the root.

- On_Duty state:

After the content owner process establishes its root, it is in its "On-Duty" state. The content owner process is standing by to wait for content requests only in this state. "On_Duty" state has four sub-states: When the content owner process receives a request for content, it informs the requestor process to submit a recommendation certificate, and waits in the "Wait_Cert" state. If a recommendation certificate is received from the requestor process, the content owner process changes its state to "Grant_Trans". Otherwise, if a recommendation certificate is not available from the requestor process, or if the requestor process does not reply within a timeout period, the content owner changes its state to "Refuse_Trans". From the "Grant_Trans" and "Refuse_Trans" states, the content owner process is automatically reset to the "Standby" state so as to be able to handle the next incoming request.

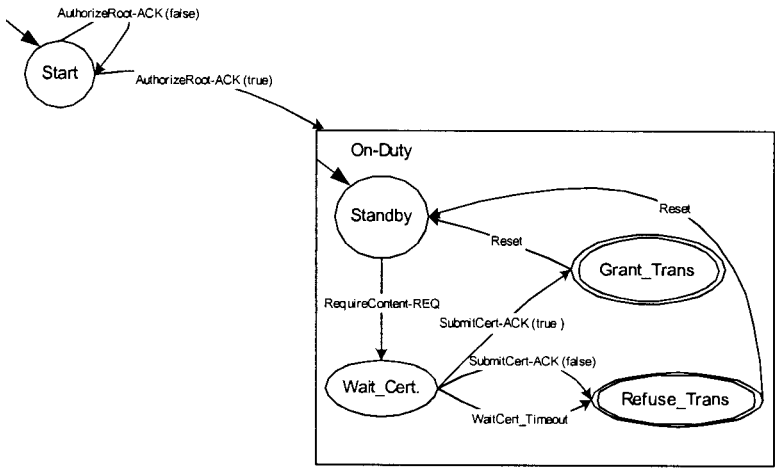


Figure 6.1: Content owner process state diagram

6.2.2 State Diagram for The Requestor Process

The behavior model for the requestor process is shown in the state diagram in Figure 6.2, and the following states are in the requestor process.

- Start state:

When the requestor process initializes a content request, it is in the "Start" state.

- Request_Cert state:

When the requestor process receives a request for submitting a recommendation certificate by the content owner, the requestor process is in the "Request_Cert" state. In this state, the requestor process contacts its previous certificate issuer nodes or the root for requesting a recommendation certificate. If the contacted node replies without issuing a recommendation certificate or doesn't reply within a timeout period, the requestor process stays in this state to contact the next available potential certificate issuer. When the requestor process receives a recommendation certificate and decides to stop to contact the next potential certificate issuer, it changes its state to "Request_Content".

- Request_Content state:

In this state, the requestor process submits a recommendation certificate to the content owner process and waits for the message for content granting from the content owner process.

- Obtain_Trans state:

If the requestor process obtains a message for confirming the content granting, the state is changed to "Obtain_Trans". This is one of the final states for the requestor process.

- Deny_Trans state:

If the content owner process refuses to grant content to the requestor process, the state is changed to the other final state "Deny_Trans".

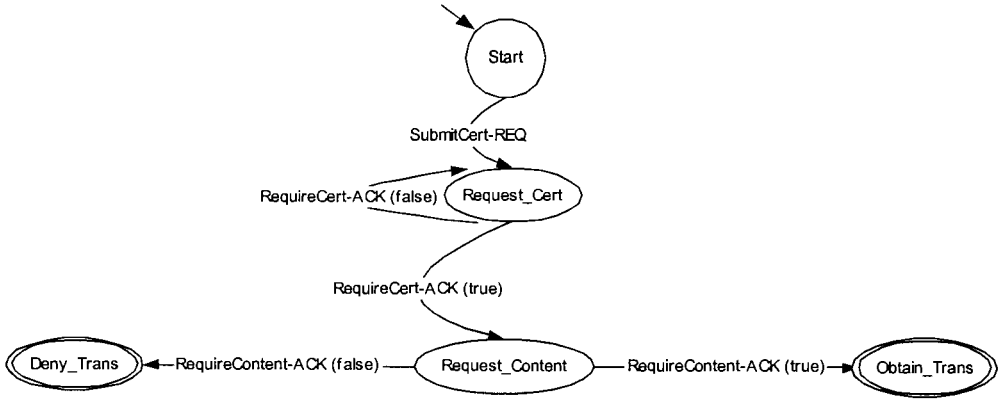


Figure 6.2: Requestor process state diagram

6.2.3 State Diagram for The Root and Node Processes

Root and node processes are started by other processes, which are different from the content owner and the requestor processes that are self-initialized. When the root process or the node process is in "Standby" state, it has message processors to handle different messages. The behavior models for the root and node processes are presented in the state transition diagrams in Figure 6.3 and 6.4.

The root and node processes have the following states in common:

- Idle state:

In the "Idle state", the root process waits for an AuthorizeRoot-REQ message from the content owner process to become a root. Similarly, the node process waits for an AddChild-REQ message from another node to become a child.

- Startup state:

In the "Startup state", the root process decides whether to accept the request to be the root of the content owner, and the node process decides whether to be the child of another node. If the root or node process accepts the request, it changes its state to "Standby". Otherwise, the process goes back to the "Idle" state.

- Standby state:

In the "Standby" state, the root or node process waits for messages and goes to "Msg_Processors" to handle each message with the corresponding message processor. For the root process, if it crashes, it is in the "End" state.

- End state:

"End" state is the final state for both the root and node processes. In this state, the root and node processes are no longer associated with and providing service to the content owner.

- Msg_Processors:

In this state, the root and node processes have the same behaviors. The message processors are triggered by the corresponding messages shown in Figures 6.4 and 6.5.

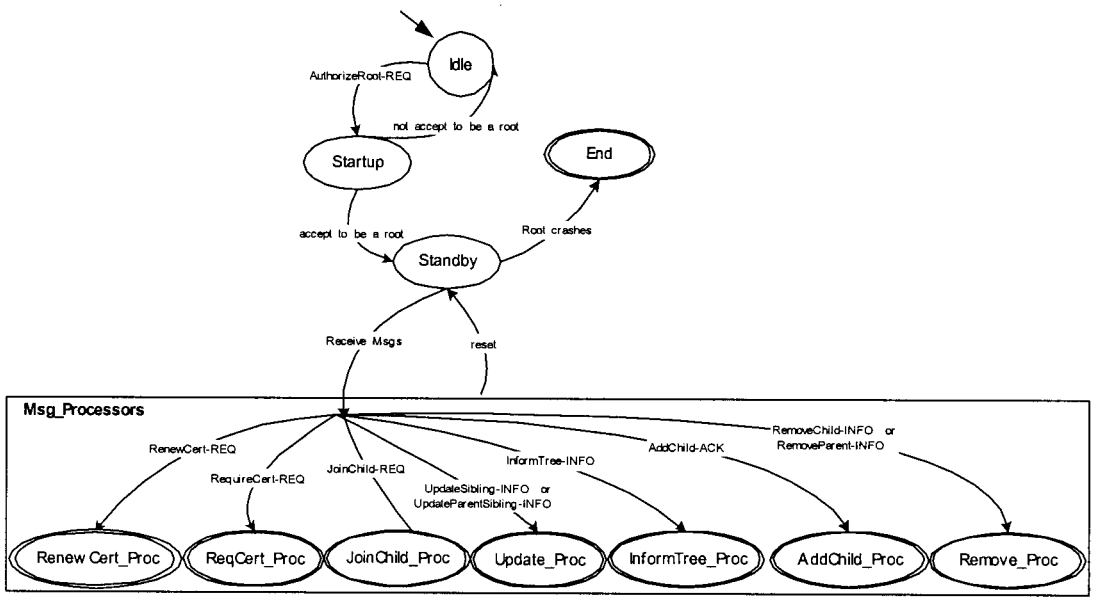


Figure 6.3 Root process state diagram

For the node process, it is necessary to incorporate the following states into the behavior model.

- Wait_Renew:

When the delegation certificate of the node process is expired, the node process changes its state to the "Wait_Renew" state. In this state, the node process sends a RenewCert-REQ message to its parent. If the parent grants a new delegation certificate, the node process goes back to its "Standby" state. Otherwise, if the parent doesn't grant a new delegation certificate, the node process goes to the final state "End". If there is no reply at all in this state upon timing out, that is, when the Renew Timer expires, the node process changes its state to "Kick_Off".

- Kick_Off state:

In this state, the node process is trying to contact any possible parent to accept it as a child. If there is a reply from another node that would like to accept the node to be a child, the node process goes back to the "Standby" state. Otherwise, when the node process times out, which in this case means that the Kickoff Timer has expired, the node process ends in the "End" state.

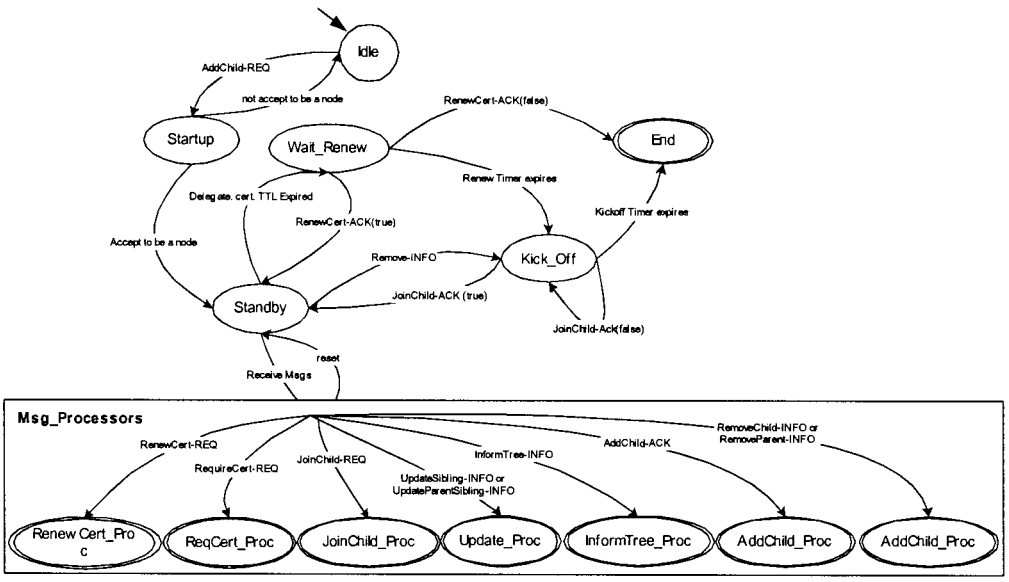


Figure 6.4: Node process state diagram

6.2.4 SPIN Verification for DDTP Properties

SPIN can be used to detect common design flaws, which is described in Appendix A. In case of the DDTP, SPIN is used to report design errors, such as invalid endstate, non-

progress cycle. Moreover, SPIN can check the design correctness requirements of the protocol specification. The following properties in the DDTP are set out and verified by SPIN Linear Temporal Logic (LTL) model checking:

- Property 1: A requestor's request for content will always be eventually acknowledged.

  [](requestorState==Request_Content-> <> (requestorState==Obtain_Trans or requestorState==Deny_Trans))

- Property 2: A requestor's request for a certificate will always be eventually acknowledged.

  []((requestorState ==Request_Cert)-> <>(requestorState==Request_Content))

- Property 3: The number of a node's children does not exceed the node's *Children Degree Threshold.*

  [](numberofchildren < ChildDegreeThreshold)

6.3 Simulation and Verification Using SPIN

Since much of the complexity of the DDTP is in managing the Trust Delegation Tree, DDTP communication is divided into two parts in SPIN simulation and verification. The first part has to do with the basic communication among a content owner, a requestor, a root and several nodes to be able to issue a recommendation certificate to the requestor, while the second part concerns the communication within a TDT for dynamically managing the nodes with the hierarchical authorization level. DDTP.pml and TDT.pml are PROMELA source code files for these two parts and are attached in Appendix B and C.

6.3.1 Simulation Message Sequence Chart

By running SPIN simulation, the basic communication for a requestor accessing the content owner is shown in the SPIN generated message sequence chart in Figure 6.5.

There are a content owner process, a requestor process, a root process and a maximum of 10 certificate issuer node processes running in this simulation.

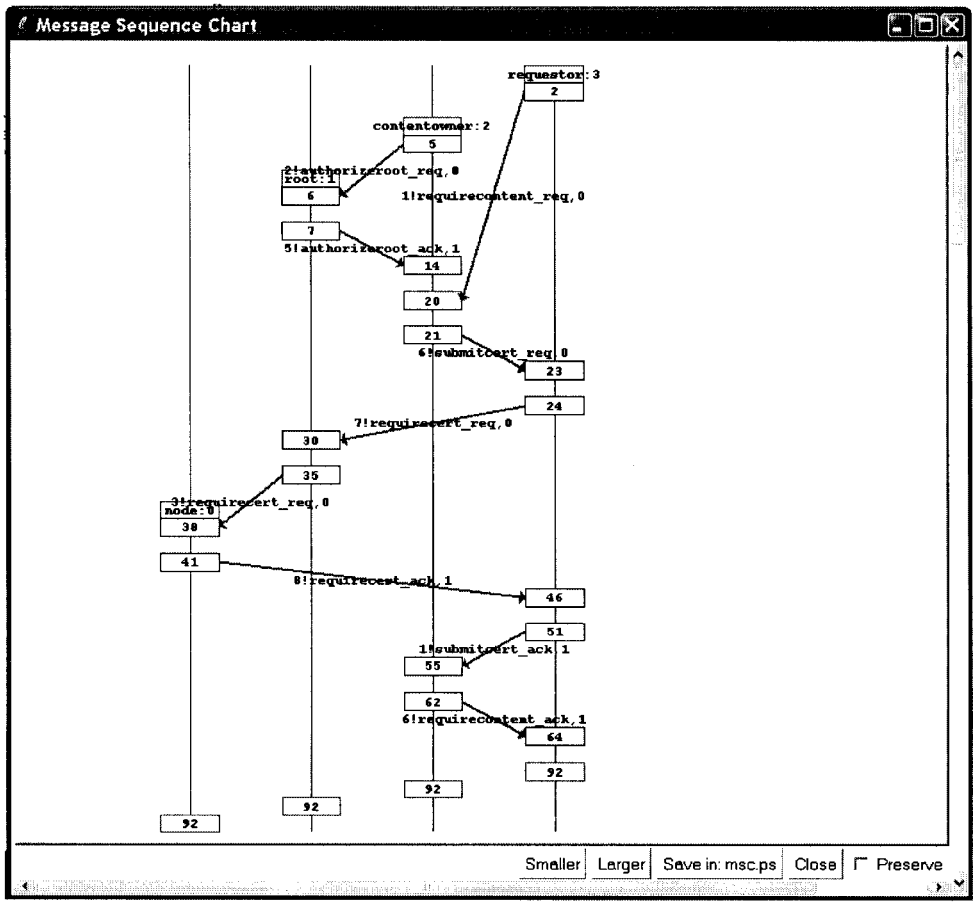


Figure 6.5 Message sequence chart generated by SPIN for the basic communication in the DDTP

The communication protocol for updating a TDT is shown in the SPIN generated message sequence chart in Figure 6.6. In this simulation, a parent node adds a node as its child, and the child should be able to renew its delegation certificate from its parent. The parent node could also remove the child node and recommend the removed child node to be a child of another node.

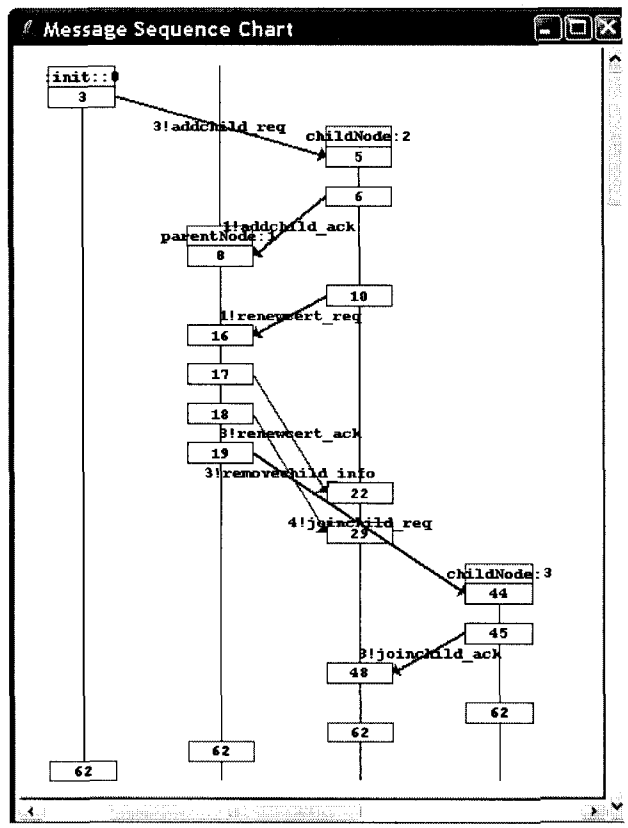


Figure 6.6: Message sequence chart generated by SPIN for updating TDT communication

6.3.2 SPIN Verification Result

Deadlock was firstly checked with SPIN's facility of checking for the occurrence of invalid endstates. Livelock was checked using the non-progress cycle detection algorithm of SPIN. LTL model checking was also employed to determine whether the protocol properties are invalid.

The verification about the above correctness requirements was performed on the AMD Atholom (tm) processor 1.3GHz and 514M memory. Partial order reduction and an exhaustive search mode were employed in the verification. After fixing all the detected errors, Table 6.1 shows the size of the state space and transition that SPIN constructs to prove that the correctness requirements are indeed satisfied.

Table 6.1: SPIN verification summary for the DDTP

Property	States	Transitions	Memory (Mbyte)	CPU Time (Sec)
No deadlock (communication for accessing the content owner with 10 issuer nodes)	5,906	13,563	3.44	16
No non-progress cycles (communication for accessing the content owner with 10 issuer nodes)	20,037	50,035	5.592	55
Property 1	8,948	33,606	3.95	20
Property 2	7,349	24,169	3.646	18
Property 3	7,657	13,563	3.441	10
No deadlock (TDT updating communication)	73	84	2.622	9
No non-progress cycles (TDT updating communication)	116	211	2.622	9

SPIN translates each process template into a finite automaton. By computing an asynchronous interleaving product of automata, the global behavior of the concurrent system is obtained. This interleaving product is referred to as the state space of the system. In concurrent protocols, the number of states usually grows exponentially with the number of processes and the data objects they access.

For example, the first row in Table 6.1 indicates that SPIN did not detect any deadlock in the communication for a requestor accessing the content owner, and the verification generated 5906 states, 13563 transitions with 3.44 MB memory usage and 16 sec CPU time.

Chapter 7

Prototyping

The goal of this prototype is to show how a TDT is established and how the nodes are dynamically changed within the TDT.

In this prototyping, we have simulated 20 entities, one of which is dedicated to be a content owner. After the content owner authorizes the root for itself, all the other entities can act as requestors to do transactions with the content owner. The interface window for the prototyping is divided into the graph part and the data part. All the entities are rectangle icons shown on the graph part, and the information for each entity is shown on the data part.

7.1 Program Environment

The DDTP prototyping were written in Microsoft Visual Basic 6.0, and the data were stored and retrieved from the database by Microsoft Access 9.0. Since the network delay is not an important concern in this protocol, the prototyping was designed to run on a single computer to simulate the behaviors of the entities in this protocol. The code is working under Window XP with an Inter Pentium 2.0 GHz processor and 256 MB of RAM.

7.2 Dynamic Topology of A TDT

The prototype is started by clicking on the content owner's icon to authorize the root of a TDT (as in Figure 7.1).

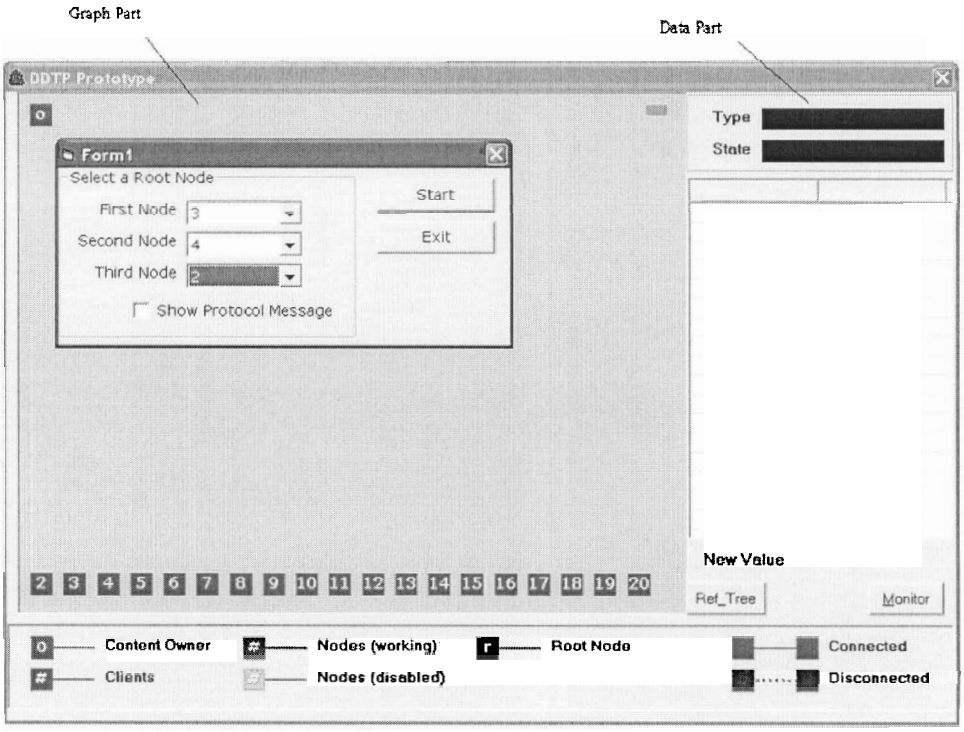


Figure 7.1: DDTP prototype interface – the content owner authorized the root

After the root is established, we can click on any other entity to select the recommendation certificate issuer and the number of transactions the entity will have with the content owner. Figure 7.2 shows that entity 2 would like to request recommendation certificates from the root and would have 180 transactions with the content owner.

Figure 7.2: DDTP prototype xinterface – entity 2 requested the root to issue a recommendation certificate for doing transactions with the content owner

As the number of transactions with the content owner increases, the certificate issuer gradually increases the direct trust value to the requestor and finally may add it as a child. In Figure 7.3, after 180 transactions, entity 2 was added as a child of the root.

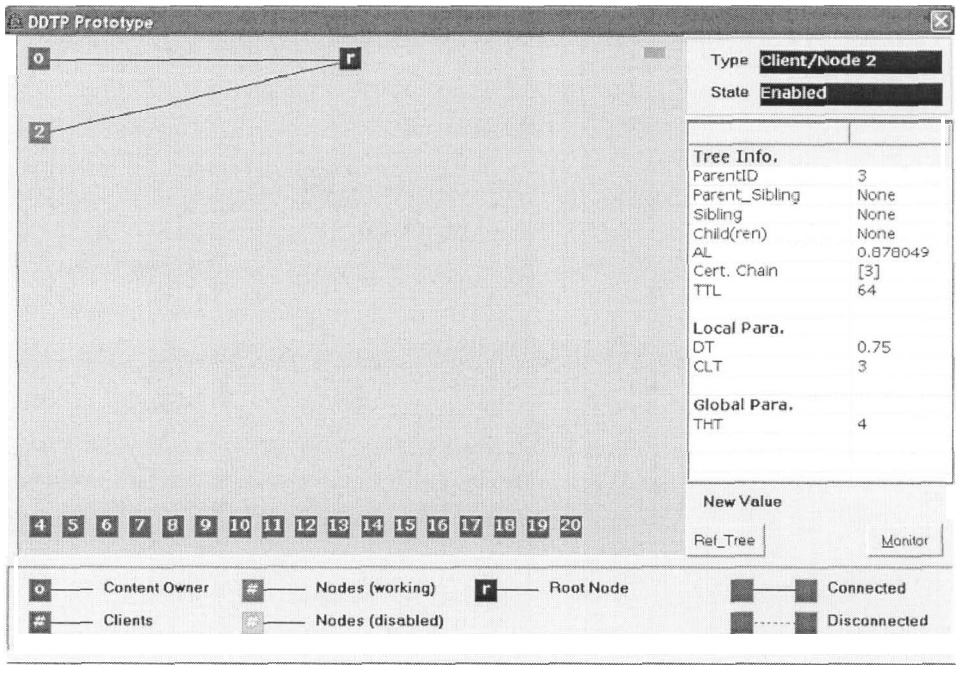


Figure 7.3: DDTP prototype interface – entity 2 was added as the child of the root

If the direct trust value of the requestor becomes higher than the authorization level of the requestor's parent, the root would upgrade the node to a higher level on the TDT. The root is able to know when to upgrade a node because the root keeps the transaction history information about the content owner and its requestors.

Figure 7.4 shows entity 5 being upgraded when its trust value exceeds that of entity 3, its parent.

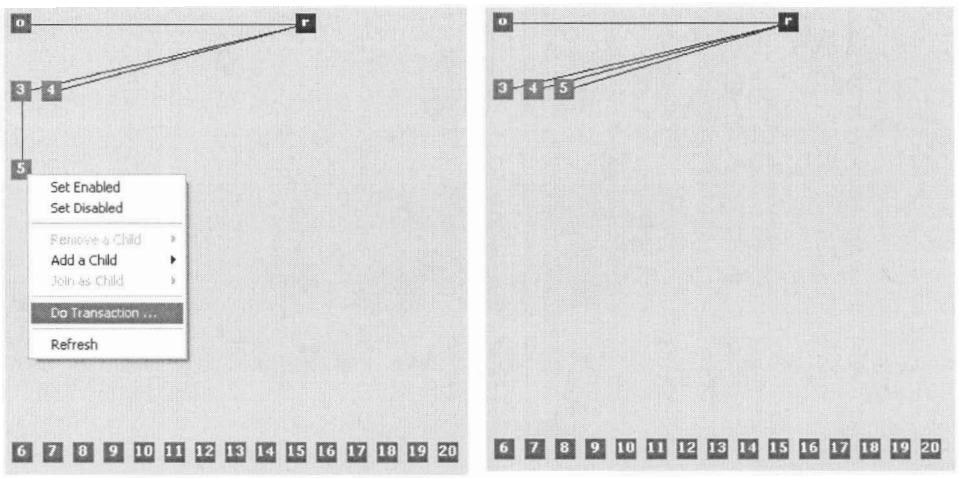


Figure 7.4: DDTP prototype interface – entity 5 was upgraded to a higher trust authorization level

Each node specifies its *Children Degree Threshold (CDT)*. A node can replace its existing child with another entity that has a higher direct trust value.

Figure 7.5 shows that the root reached its *CDT*. When the root developed a higher direct trust value on entity 6, the root replaced entity 4 (assume it is the least direct trust value of root's existing children) with entity 6. Entity 4 was recommended by the root to entity 3 and became the child of entity 3.

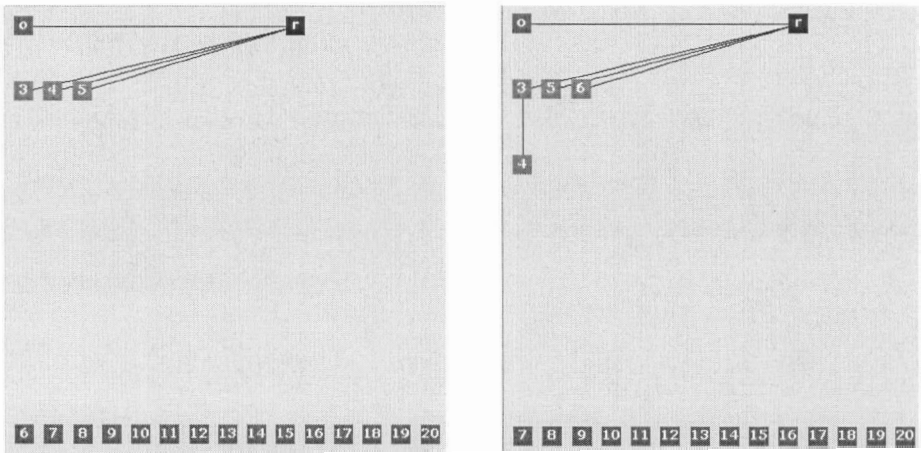


Figure 7.5: DDTP prototype interface – entity 4 was replaced with entity 6, which had a higher direct trust value from the root

Whenever a node changes its parent, it informs its previous parent about the change, and the information about the change is further passed around if it is necessary. Therefore, it is guaranteed that the information about any node is consistently recorded at the relative nodes on the TDT.

Figure 7.6 shows that when we clicked node 3 and set it to be disabled, the children of the node requested one of their previous parent's siblings, entity 5, to accept them as children. By doing this, the TDT will not lose the sub-tree under the disabled node.

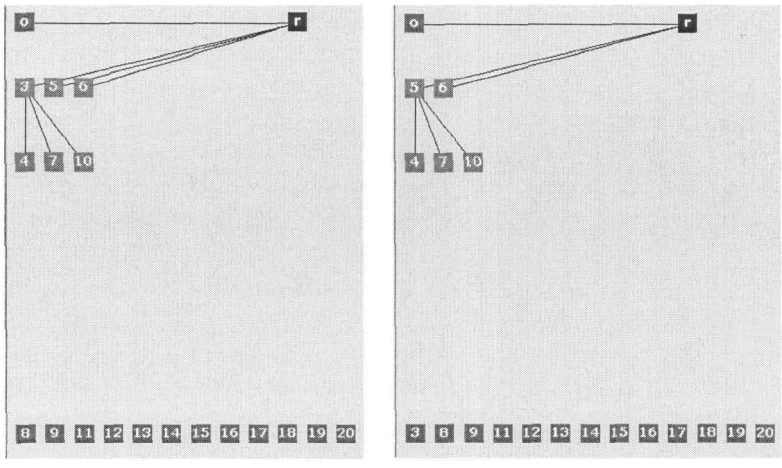


Figure 7.6: DDTP prototype interface – when entity 3 was set to be disabled, the children of entity 3 were shifted to entity 5

7.3 Viewing Node's data

In this prototype, each node's data can be viewed by clicking on the node's icon. Three types of the information are recorded at each node:

- **Tree Info Data**: This includes the following information for the current node:

  Parent, parent-sibling(s), sibling(s), child list, trust authorization level, the delegation certificate chain from the root, time to live of the recommendation certificate.

Local parameters: These are the parameters that can be set by the current node. The *Delegation Threshold (DT)* and the *Children Degree Threshold (CDT)* are examples.

- **Global parameters:** These are the parameters set by the content owner and passed over to each node of the TDT. The examples are the *Tree Height Threshold* (*THT*) and the Black List.

Figure 7.7 shows that entity 6 stores the delegation certificate chain from the root to itself. Therefore, whenever the node issues a recommendation certificate to other requestors, the issuer node can provide the complete delegation certificate chain without contacting any other nodes.

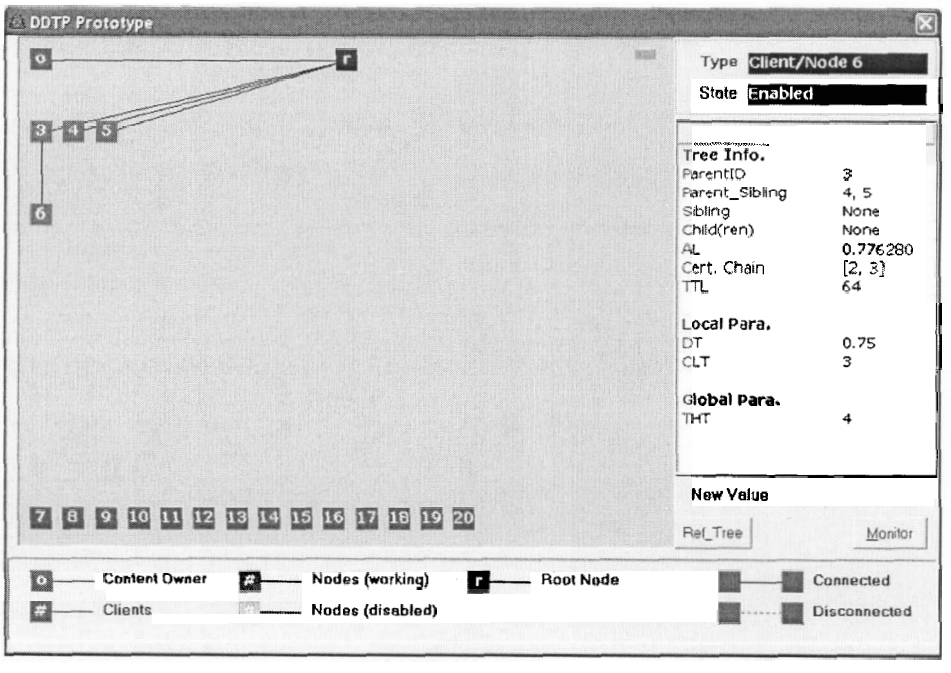


Figure 7.7: DDTP prototype interface – information shown for entity 6

To increase the scalability, each node can increase the *Children Degree Threshold* and decrease the *Delegation Threshold* to be able to take more children.

Chapter 8

Conclusions

As more content and software resources have been distributed on the Internet, the centralized access control infrastructures have been proven inadequate with respect to scalability and flexibility. We have introduced a Dynamic Distributed Trust Model, which addresses these concerns by providing a distributed key-oriented certificate-issuing mechanism with no centralized global authority. By utilizing a Trust Delegation Tree and delegation certificate chains, the DDTM can be used to provide a scalable and fault-tolerant infrastructure for granting access to the contents on the Internet.

8.1 Contributions

In this thesis, we have first proposed the parameters and methods used for calculating the trust values. After that, we have concentrated on applying the trust values to provide a general-purpose, application-independent access control mechanism for the content providers over a distributed network like the Internet. In addition, we have aimed to support scalability, flexibility, consistency and reliability of the system. The major contributions of our work are summarized as follows:

1 We have developed the formulas for quantifying trust values into explicit numerical numbers by using objective and subjective parameters. The Dempster-Shafer function was applied in our work to combine more than one subjective evidence in order to quantify the trust values.

2 We proposed an access control model that associates the authorization with the trust value instead of detail access right specification. This makes the model free

from the applications' access policy and flexible with respect to various applications.

3 Delegation certificate chains and a Trust Delegation Tree were proposed in our Dynamic Distributed Trust Model. The delegation certificate chains enable the certificate-issuing authority to be distributed over the Internet, and the delegated entities for a specific content owner are managed as nodes on a hierarchical Trust Delegation Tree. Each node in the tree is able to issue certificates to provide trust to other entities without contacting the root, so that it is scalable for a large number of clients accessing a content provider.

4 A set of operations on the Trust Delegation Tree was developed to accomplish the management of the dynamic trust relationship, enabling the Trust Delegation Tree to consistently and reliably issue certificates for trust authorization to clients. Moreover, this model is flexible with respect to the *Children Degree Threshold* and the *Delegation Threshold* that each node can choose on its own to increase the scalability.

5 A Dynamic Distributed Trust Protocol was specified in detail with the exchanged messages and data fields in each message. The protocol was simulated and verified by a verification tool, SPIN, and produced no errors in the system.

6 A prototype for the DDTP was written to create a visual demonstration about how the Trust Delegation Tree works.

7 This thesis has laid a feasible foundation for further development of the much needed trust mechanisms for an eCommerce and similar Internet applications.

8.2 Future Work

In this thesis, the Dynamic Distributed Trust Model and Protocol were fully developed. The direct extension to this work would be a protocol simulation in a network environment. When multiple Trust Delegation Trees are working for corresponding content owners in a network, the DDTM model will increase the efficiency of the system by utilizing the trust relationship in one tree for another tree. It would be interesting to

see the simulation result with multiple TDTs working for several corresponding content owners in future.

Since the DDTM is a unique and new trust model, we could not find another system that was similar enough for performance comparison. Further work should include performance study of the DDTM with regard to its computational complexity and resource requirements.

There is still much work that needs to be done in the "distributed trust management", including the method for real-time certificate revocation, method for reducing a certificate chain, standardization of a policy language, etc. Our DDTP will be useful as a good starting point to carry out such research.

Bibliography

[Abd97a] Alfarez Abdul-Raduman. "The PGP Trust Model," *In EDI-Forum: The Journal of Electronic Commerce*, April 1997.

[AH98] A. Abdul-Rahman, S. Hailes, "A Distributed Trust Model," in *1997 New Security Para-digms Workshop, Proceedings*, ACM Press, pp. 48–60, 1998.

[Al01] Eyas Al-Hajery, "Trust Model in PGP and X.509 Standard PKI," *SANS Institute 2000 – 2002,* April 9, 2001, *Available: http://www.giac.org/practical/gsec/Eyas_Al-Hajery_GSEC.pdf.*

[BFL96] Matt Blaze, Joan Feigenbaum, and Jacy Lacy. "Decentralized Trust Management," *In IEEE Conference on Security and Privacy.* Oakland, CA, May 1996.

[BFO01] Olav Bandmann, Babak Sadighi Firozabadi, Olle Olsson. "Decentralized Management of Access Control," *Swedish Institute of Computer Science.* Internal SICS project report, 2001.

[BFIK99] M. Blaze, J. Feigenbaum, J. Ioannidis, and A. Keromytis. "The Role of Trust Management in Distributed Systems Security," In Vitek and Jensen, editors, *Secure Internet Programming: Security Issues for Mobile and Distributed Objects*, Springer-Verlag, 1999.

[Dem68] A.P. Dempster. "A generalization of Bayesian inference," *Journal of the Royal Statistical Society,* Series B 30 205-47, 1968.

[EFL98] Carl M. Ellison, Bill Franz, Butler Lampson, Ron Rivest, Brain M.Thomas, Tatu Ylonen. "Simple Public Key Certificate," Internet Draft, draft-ietf-spki-cert-structure-05.txt, 1998.

[FKK96] Alan O. Freier, Philip Karlton, Paul C. Kocher, "The SSL Protocol, Version 3," Internet draft, Netscape Communications, March 1996.

[FPPKK01] Eric Freudenthal, Tract Pesin, Lawrence Port, Edward Keenan, and Vijay Karamcheti. "dRBAC: Distributed Role-based Access Control for Dynamic Coalition Environments," Technical Report TR2001-819.(to appear ICDCS 2002), 2001. 8.

[FSB01] Babak Sadighi Firozabadoi, Marek Sergot, olav Bandmann. "Using Authority Certificates to Create Management Structures," *In Proceeding of Security Protocols*, 9th International Workshop, Cambridge, UK, April 2001.

[BGS92] John A. Bull, Li Gong, and Karen R. Sollins. "Towards Security In An Open Systems Federation," *In Computer Security – Proc. ESORICS 92, volume 648 of LNCS,* pages 3-20, Toulouse, Franc, November 1992, Springer -Verlag.

[HMM00] Amir Herzberg, Yosi Mass, Joris Michaeli. "Access Control Meets Public Key Infrastructure, Or: Assigning Roles to Strangers," Security & Privacy, 2000.

[Hmd] Mark de Haas, Online Article. "E-business and trust," *Available: http://www.de-haas.nl/Articles/Compact Trust issues on the web.htm.*

[Hol97] G. J. Holzmann. "The model checker SPIN, " *IEEE Transactions on Software Engineering,* 23(5): 279-295, May 1997.

[Hol03] G. J. Holzmann. "The SPIN MODEL CHECKER, " Addison-Wesley, September 2003.

[Kan01] Pekka Kanerva. "Anonymous Authorization in Networked System: An Implementation of Physical Access Control System," Master's Thesis, Helsinki University of Technology. Espoo, 2001.

[KCFP01a] Lalana Kagal, Scott Cost, Timothy Finin and Yun Peng. "A Framework for Distributed Trust Management," *Second Workshop on Norms and Institutions in MAS, Autonomous Agents*, May 2001.

[KCFP01b] Lalana Kagal, Scott Cost, Timothy Finin and Yun Peng. "A Delegation Based Model for Distributed Trust," *Workshop on Autonomy, Delegation, and Control: Interacting with Autonomous Agents, International Joint Conferences on Artificial Intelligence*, August 2001.

[Ker01] Angelos Dennis Keromytis. "STRONGMAN: A SCALABLE SOLUTION TO TRUST MANAGEMENT IN NETWORKS," PhD Thesis. University of Pennsylvania, 2001.

[KFJ01] Lalana Kagal, Tim Finin and Anupam Joshi. "Moving from Security to Distributed Trust in Ubiquitous Computing Environments," *IEEE Computer*, December 2001.

[Kon78] Loren M. Kohnfelder. "Towards a practical public-key cryptosystem," Bachelor's thesis, Massachusetts Institute of Technology, Cambridge MA USA, 1978.

[KRM97] Petteri Koponen, Juhana Rasanen, and Olli Martikainen. "Calypso service architecture for broadband networks," *In Proc. IFIP TC6 WG6.7 International Conference on Intelligent Networks and Intelligence in Network,* Chapman & Hall, September 1997.

[Lam74] B.W. Lampson. "Protection," *ACM Operating Systems Review,* 8(1): 18-24, January 1974.

[Lug02] Geoge F. Luger. "Artificial Intelligence - Structures and Strategies for Complex Problem Solving," Fourth Edition, Addison-Wesley 2002.

[Man00] Daniel W. Manchala. "E-Commerce Trust Metrics and Models," *IEEE Internet Computing*, 4(2), 36-44, 2000.

[Man98] Daniel W. Manchala. "Trust Metrics, Models and Protocols for Electronic Commerce Transactions," *The 18th International Conference on Distributed Computing Systems,* Amsterdam, The Netherlands, May 26 - 29, 1998.

[MWTO] Merriam-Webster Thesaurus Online, *Available: http://www.m-w.com/cgi-bin/thesaurus.*

[Nik99] Pekka Nikander. "An Architecture for Authorization and Delegation in Distributed Object-Oriented Agent Systems," Doctoral Dissertation, Helsinkin University of Technology, 1999.

[PGP5.0] "Pretty Good Privacy PGP for Personal Privacy, Version 5.0. User's Guide," 1997 by PGP, Inc.

[Pkix02] A. Arsenault. "Internet X. 509 Public Key Infrastructure: Roadmap," *PKIX Working Group Internet Draft.* May 2002.

[Sha76] Glen Shafer. "A Mathematical Theory of Evidence," Princeton University Press, 1976.

[Sha90] Glen Shafer. "Perspectives on the theory and practice of belief functions," *International Journal of Approximate Reasoning 3 1-40,* 1990.

[SR86] Romualdas Skvarcius, William B.Robinson, "Discrete Mathematics with computer science applications," Benjamin-Cummings Publishing Co., Inc, 1986.

[T01] Grandison, T. "Trust Specification and Analysis for Internet Applications." Ph.D. Thesis, Im-perial College of Science Technology and Medicine, Department of Computing, London, 2001.

[Aur99] Tuomas Aura. "Distributed Access-Rights Management with Delegation Certificates," *Secure Internet Programming (LNCS 1603),* 1999 Springer-Verlag.

[Tyk00] Toni Nykane. "Attribute Certificate in X.509," Helsinki University of Technology. 2000.

[TPC] Yun Teng, Vir V. Phoha, Ben Choi. "Design of Trust Metrics Based on Dempster-Shafer Theory,"

[Wan98] Yulian Wang. "SPKI". *Helsinki University of Technology,* December, 1998.

[Ylö96] Tatu Ylönen, "SSH – Secure Login Connections over the Internet," *In the proceedings of The Sixth USENiX Security Symposium,* San Jose, California, USA, July 22-25, 1996.

Appendix A

Detecting Common Design Flaws By SPIN

SPIN checks for safety properties by default, for example, the absence of reachable system deadlock states, which is considered by default to be an unintended end state of the system. However, the system correctness properties should be claimed by some constructs, and SPIN tries to find at least a counterexample of the claims.

The following constructs in PROMELA are used to formalize correctness properties.

- Basic assertion: `assert (expression)`

PROMELA basic assertion is always executable, and it has no effect when the expression has a boolean value of true or a non-zero integer value. The basic assertion implies that it is never possible for the expression to evaluate to false (or zero).

- Progress state label: `progress, progress0, progression:`

The progress label marks statements in PROMELA, which is desirable rather than idling and waiting. SPIN has built-in checks through the labeled statements to verify that every potentially infinite execution cycle passes through at least one of the progress labels. If any cycle can be found without this property, the non-progress loop (also called starvation) is detected.

- Linear Temporal Logic Formula

SPIN accepts correctness properties expressed in linear temporal logic to formalize system behaviors that are claimed to be impossible. If a property is invalid, an error trace is provided by SPIN. SPIN accepts LTL formulae that consist of prepositional

symbols (including the predefined true and false terms), unary and binary temporal operators. The grammar of LTL is shown as following:

$$unop ::= \begin{array}{ll} [] & (always) \\ <> & (enventually) \\ ! & (logicaly\ negation) \end{array}$$

$$binop ::= \begin{array}{ll} U & (strong\ until) \\ \&\& & (logical\ and) \\ || & (logical\ or) \\ \rightarrow & (implication) \\ <-> & (equivalence) \end{array}$$

Figure Appendix A. 1: Linear Temporal Logic grammar [Hol97]

Appendix B

PROMELA Source Code – DDTM.pml

```
#define NUM 10 /*nr of certificate issuers.*/

/*msg types for "content requisition protocol"*/
mtype = {requirecontent_req, requirecontent_ack,submitcert_req, submitcert_ack};
/*msg types for "recommendation certificate requisition protoco"*/
mtype = {requirecert_req, requirecert_ack};
/*msg types for "authoriztion protocol"*/
mtype = {authorizeroot_req, authorizeroot_ack};

/*states for a content owner*/
mtype = {standby, wait_cert, grant_trans, refuse_trans};
/*states for a requestor*/
mtype = {start, request_cert, request_content, deny_trans, obtain_trans};
/*states for a node*/
mtype = {idle,/*start, standby,*/ reqcert_proc};

chan co_req = [1] of {mtype, bool};
chan req_co = [1] of {mtype, bool};

chan co_root = [1] of {mtype, bool};
chan root_co = [1] of {mtype, bool};

chan req_root = [1] of {mtype, bool};
chan node_req = [1] of {mtype, bool};

chan root_node = [1] of {mtype, bool};
chan req_node = [1] of {mtype, bool};
chan root_req = [1] of {mtype, bool};

chan nodeToNode = [1] of {mtype, bool}
chan nodeToReq = [1] of {mtype, bool}

byte CDT = 4;
byte THT = 10;
byte numofchild = 0;
byte height = 1;
byte nodePid = 1;

mtype requestorState;

proctype node(chan in; chan out)
{
        mtype state = standby;
        bool data_flag = false;

        do
        ::(state == standby)->
                if
                        ::in?requirecert_req(data_flag)->
                                atomic {out!requirecert_ack(true); state = reqcert_proc}
                        ::timeout->skip
                fi
        ::(state==reqcert_proc)->state = standby
        od
}

active proctype root(){
```

```
	bool data_flag = false;
	mtype state = idle;

	do
	::(state==idle)->
		if
		::co_root?authorizeroot_req(data_flag)-> state = start
		::timeout->skip
		fi;
	::(state == start)->
		if
		::atomic {root_co!authorizeroot_ack(true); state = standby}
		::atomic {root_co!authorizeroot_ack(false); state = idle}
		fi;
	::(state==standby)->
		if
		::numofchild=0
		::numofchild=1
		::numofchild=2
		::numofchild=3
		::numofchild=4
		fi;
		if
		::req_root?requirecert_req(data_flag)->
			if
			::((numofchild==CDT)&& nodePid < NUM) ->
				atomic {run node(nodeToNode,nodeToReq);
	nodeToNode!requirecert_req(data_flag); nodePid = nodePid+1 }
			::else
				atomic {root_req!requirecert_ack(true); state =
reqcert_proc}
			fi
		::timeout->skip
		fi
	::(state==reqcert_proc)->progress: state = standby
	od
}

active proctype		contentowner()
{
	mtype state = start;
	bool data_flag = false;
	co_root!authorizeroot_req(data_flag);

	do
	::(state==start)->
		if
		::root_co?authorizeroot_ack(data_flag)->
			if
			:: (data_flag==true)->
			progress0:		state = standby
			:: (data_flag == false)->
			progress1: atomic{state =
start;co_root!authorizeroot_req(data_flag)}
			fi
		::timeout->skip
		fi;
	::(state==standby)->
		if
		::req_co?requirecontent_req(data_flag)->
			atomic {co_req!submitcert_req(data_flag); state=wait_cert}
		::timeout->skip
		fi;
	::(state==wait_cert)->
		if
		:: req_co?submitcert_ack(data_flag)->
			if
			:: (data_flag==true)->
				atomic {co_req!requirecontent_ack(true);accept:
state = grant_trans}
			:: (data_flag == false)->
```

```
                                        atomic {co_req!requirecontent_ack(false);state =
refuse_trans}
                        fi
                :: timeout->skip
                fi;
        ::(state==grant_trans)-> state = standby
        ::(state==refuse_trans)-> state = standby
        od
}

active proctype requestor()
{
        bool data_flag;

loop:
        requestorState = start;
        data_flag = false;
        req_co!requirecontent_req(data_flag);

        do
        :: (requestorState == start)->
                if
                ::co_req?submitcert_req(data_flag)->
                        progress2:atomic {req_root!requirecert_req(data_flag);
requestorState = request_cert}
                ::timeout->skip
                fi
        ::      (requestorState == request_cert)->
                if
                ::nodeToReq?requirecert_ack(data_flag)->
                        if
                        ::(data_flag == true)->
                                atomic {req_co!submitcert_ack(true);
requestorState=request_content}
                        ::timeout->skip
                        fi;
                ::root_req?requirecert_ack(data_flag)->
                        if
                        ::(data_flag==true)->
                                atomic {req_co!submitcert_ack(true);
requestorState=request_content}
                        ::timeout->skip
                        fi;
                fi
        ::(requestorState == request_content)->
                if
                ::co_req?requirecontent_ack(data_flag)->
                                if
                                ::(data_flag == true)->       requestorState = obtain_trans
                                ::(data_flag == false) -> requestorState = deny_trans
                                fi;
                fi
        ::(requestorState == deny_trans)-> goto loop
        ::(requestorState == obtain_trans)-> goto loop
        od
}
```

Appendix C

PROMELA Source Code – TDT.pml

```
#define NUM 2
mtype = {addchild_req,addchild_ack,joinchild_req,
joinchild_ack,renewcert_req,renewcert_ack,removechild_info};
mtype = {standby, wait_renew,kick_off};

/*
parent_in                          child_in
-------------->parent node<--------------
<--------------child node -------------->
parent_out                         child_out
*/

chan childToparent[NUM] = [2] of {mtype};
chan parentTochild[NUM] = [2] of {mtype};


proctype childNode(chan parent_in; chan parent_out){
        mtype state = standby;
        progress2: if
        ::parent_in?addchild_req->
                parent_out!addchild_ack;
                atomic {parent_out!renewcert_req;     state = wait_renew}
        ::timeout->skip
        fi;

        do
        :: (state==standby)->
                if
                ::parent_in?removechild_info->
                        progress: atomic {state = kick_off}
                ::parent_in?joinchild_req->
                        progress0: parentTochild[0]!joinchild_ack
                ::timeout -> goto done
                fi
        :: (state==wait_renew)->
                if
                ::parent_in?renewcert_ack->
                        state = standby
                ::timeout->skip
                fi
        :: (state==kick_off)->
progress3:      if
                ::parent_in?joinchild_ack->
                         state = standby
                ::timeout->skip
                fi
        od;
        done:
        skip
}

proctype parentNode(chan child_in; chan child_out)
{
        mtype state = standby;
        do
        :: (state==standby)->
```

```
progress1:      if
                ::child_in[0]?renewcert_req->
                        atomic
{child_out[0]!renewcert_ack;parentTochild[0]!removechild_info;parentTochild[1]!joinchild_
req}
                ::child_in[0]?addchild_ack->
                ::timeout -> goto done
                fi
        od;
        done:
                skip
}

init{
        /*parent node*/
        run parentNode(childToparent, parentTochild);
        parentTochild[0]!addchild_req;

        /*child node*/
        run childNode(parentTochild[0], childToparent[0]);
        run childNode(parentTochild[1], childToparent[1])
}
```