TRUST RELATIONSHIPS, NAMING, AND SECURE COMMUNICATION IN LARGE DISTRIBUTED COMPUTER SYSTEMS

A DISSERTATION
SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF DOCTOR OF PHILOSOPHY IN COMPUTER SCIENCE IN THE GRADUATE DIVISION OF THE UNIVERSITY OF CALIFORNIA, BERKELEY

by

P. Venkata Rangan

September 1988
Computing systems are evolving into distributed systems that interconnect competing organizations and individuals, and even countries, using high-speed global networks. The relationships among these entities are characterized by the need for competition and cooperation without a common trusted agent. To build such distributed systems that incorporate lack of global trust in them, it is necessary first to understand precisely what trust consists of and then to categorize it. This thesis develops an axiomatic theory of trust in distributed systems. The theory is based on modal logics of belief. We present systematic methods for synthesizing protocols that implement a given trust specification. Trust is primarily required to establish channels for secure communication. We present methods for reasoning about trusts required by various channel establishment mechanisms. Channel establishment mechanisms are commonly based on either public key encryption (PKE) or single key encryption (SKE). PKE-based mechanism require ternary trust relationships known as authenticity trusts. SKE-based mechanisms have much larger trust requirements. Starting from the differences in trust requirements of PKE and SKE, we derive several advantages of the former over the latter. Our analyses provide insight into the trust structure and limitations of various mechanisms. We show that a distributed system must provide a tree of channels at system configuration time, and that this tree also represents the system’s global name space. We develop polynomial-time algorithms for synthesizing name spaces so as to satisfy an a priori given set of trust specifications. We present some interesting duality results and NP-completeness results with regard to some variations of the synthesis problems. Sample runs of the polynomial-time algorithms show that small differences in trust relationships can cause substantial differences in the structure of the name spaces. Trust requirements and the performance of channel establishment can be traded for each other. If channels are PKE-based, slightly increasing the trust requirements can greatly increase the performance of channel establishment. However, if channel composition is SKE-based, global trusts, which may not be satisfied in the system’s name space, are required for significant improvements in performance.
| 15. SUBJECT TERMS |  
| 16. SECURITY CLASSIFICATION OF: |  
| a. REPORT | b. ABSTRACT | c. THIS PAGE | 17. LIMITATION OF ABSTRACT | 18. NUMBER OF PAGES | 19a. NAME OF RESPONSIBLE PERSON |
| unclassified | unclassified | unclassified | Same as Report (SAR) | 134 |  |
Trust Relationships, Naming, and Secure Communication in Large Distributed Computer Systems

Copyright © 1988
by
P. Venkata Rangan
Trust Relationships, Naming, and Secure Communication in Large Distributed Computer Systems

by

P. Venkata Rangan

ABSTRACT

Computing systems are evolving into distributed systems that interconnect competing organizations and individuals, and even countries, using high-speed global networks. The relationships among these entities are characterized by the need for competition and cooperation without a common trusted agent. To build such distributed systems that incorporate lack of global trust in them, it is necessary first to understand precisely what trust consists of and then to categorize it. This thesis develops an axiomatic theory of trust in distributed systems. The theory is based on modal logics of belief. We present systematic methods for synthesizing protocols that implement a given trust specification.

Trust is primarily required to establish channels for secure communication. We present methods for reasoning about trusts required by various channel establishment mechanisms. Channel establishment mechanisms are commonly based on either public key encryption (PKE) or single key encryption (SKE). PKE-based mechanisms require ternary trust relationships known as authenticity trusts. SKE-based mechanisms have much larger trust requirements. Starting from the differences in trust requirements of PKE and SKE, we derive several advantages of the former over the latter. Our analyses provide insight into the trust structure and limitations of various mechanisms.

We show that a distributed system must provide a tree of channels at system configuration time, and that this tree also represents the system's global name space. We develop polynomial-time algorithms for synthesizing name spaces so as to satisfy an a priori given set of trust specifications. We present some interesting duality results and NP-completeness results with regard to some variations of the synthesis problems. Sample runs of the polynomial-time algorithms show that small differences in trust relationships can cause substantial differences in the structure of the name spaces.

Trust requirements and the performance of channel establishment can be traded for each other. If channels are PKE-based, slightly increasing the trust requirements can greatly increase the performance of channel establishment. However, if channel composition is SKE-based, global trusts, which may not be satisfied in the system's name space, are required for significant improvements in performance.
I cannot find words to express my utmost gratitude to Professor Domenico Ferrari for his constant guidance, encouragement and support. Domenico is a perfect researcher and teacher. I will always cherish all that I learned from him. His warmth as a human being has been limitless.

I am highly thankful to Professor Luis-Felipe Cabrera for his ideas and guidance. I would like to thank Professor David Anderson for his guidance in the initial stages of my research. David's insistence on precision played a major role in shaping the directions of my research. The work reported in Chapter 5 on ADP is joint with David, Domenico and Bruno Sartirana. I highly enjoyed working with Bruno on the ADP software and experiments in DASH. I am thankful to Professor Ramamoorthy for his guidance and encouragement, and to Professor Charles Stone for serving on my thesis committee. I would like to thank Professor Richard Kemmerer of UCSB for his encouragement.

I have benefited very much from discussions with numerous friends. Joe Pasquale and Stuart Sechrest have provided me with guidance and help on innumerable occasions. I will always treasure their friendship. Shin-Yuan Tzou was highly helpful during the writing of the ADP software. I thank all the members of the Progres research group at Berkeley, in particular, Hamid Bahadori, Peter Danzig, Kevin Fall, Vijay Garg, Riccardo Gusella, Diane Hernek, Harry Rubin, Keshav Srinivasan, Mark Sullivan, Dinesh Verma and Songnian Zhou for their friendship and help. I cannot think of a more dynamic and friendly research group than the Progres Group. Stuart has been instrumental in making Progres a very enjoyable and friendly group to be part of. Discussions with Stuart have greatly contributed to my appreciation of the American culture and society.

I owe my life's work and inspiration to my parents, Nagarathnamma and Prasanna Kumar, who are models of perfection in every walk of life, whose teaching and training have brought me where I am today, and whose love and care are so abundant that I can never repay, and to Chinmayanandaji, whose exposition of the Bhagavad Geetha has been the driving force through out my life. My brother, Sreerang Rajan, has been a source of wisdom, and, my little brother, Srijari Sampath Kumar, has been a constant source of enthusiasm.

This work was partially sponsored by an IBM Doctoral Fellowship, by the Defense Advanced Research Projects Agency (DoD), ARPA Order No. 4871, monitored by the Naval Electronic Systems Command under Contract No. N00039-84-C-0089, by the IBM Corporation, by Olivetti S.p.A., by MICOM-Interlan, Inc., by CSELT S.p.A., by Hitachi, by Cray Research, and by the University of California under the MICRO Program.
Invocation

The inspiration for my pursuit of doctoral research comes from the challenge of discovering an answer to the following question posed in the ancient Indian philosophical texts of the Mundaka Upanyshadhd and the Bhagavad Geetha.

Shaunako hy vai mahāśātho angyrasam vydhyvad upāsannah prapaccha:
"Kasmyny bhagavo vygniithe, sarvamydam vygnātham bhavathythy ?"

In ancient times, Shaunaka approached Angyras and asked:
"What is That, by knowing Which, everything else becomes known ?"
To my parents: Amma and Anna
TABLE OF CONTENTS

Chapter 1. INTRODUCTION ........................................................................................................ 1
  1.1. Motivation ....................................................................................................................... 1
  1.2. Trust Relationships, Naming and Secure Communication in Distributed Systems .... 2
  1.3. Relation to Previous Work ............................................................................................. 5
  1.4. Outline of the Thesis ..................................................................................................... 7

Chapter 2. AXIOMATIZATION OF TRUST ............................................................................. 10
  2.1. Introduction ................................................................................................................. 10
  2.2. An Approach to Axiomatization ................................................................................... 11
    2.2.1. Theories .................................................................................................................. 11
    2.2.2. A Logic for Trust ..................................................................................................... 12
    2.2.3. Logics of Belief ........................................................................................................ 15
  2.3. Modal Logic of Belief: A Review .................................................................................. 15
    2.3.1. Syntax ..................................................................................................................... 15
    2.3.2. Semantics .................................................................................................................. 16
    2.3.3. The Kripke Structure: A Formal Semantic Interpretation of Belief ...................... 17
  2.4. A Distributed System Model .......................................................................................... 18
    2.4.1. States ....................................................................................................................... 18
    2.4.2. Possibility Relations ............................................................................................... 19
    2.4.3. Belief Acquisition ..................................................................................................... 21
    2.4.4. Sending Beliefs ........................................................................................................ 22
  2.5. A Logic of Belief for the Distributed System Model .................................................... 22
  2.6. A Theory of Trust ......................................................................................................... 24
  2.7. Synthesizing Protocols from Abstract Trust Specifications ........................................ 25
  2.8. Conclusion .................................................................................................................... 34

Chapter 3. ANALYSIS ............................................................................................................. 35
  3.1. Introduction .................................................................................................................... 35
  3.2. Channels ........................................................................................................................ 36
    3.2.1. Definition of Channel .............................................................................................. 36
    3.2.2. Channel Establishment ............................................................................................ 38
  3.3. Atomic Propositions ....................................................................................................... 40
  3.4. Composition of Two PKE-based Independent Channels ............................................. 42
    3.4.1. Trusts in PKE-based Channel Composition ............................................................. 42
    3.4.2. Necessity and Sufficiency of Authenticity Trust .................................................... 45
    3.4.3. Semantic Interpretation of the Authenticity Trust .................................................. 47
3.5. Composition of Two SKE-based Independent Channels ........................................... 49
3.6. Composition of Two Dependent Channels ............................................................. 56
3.7. Composition of More Than Two Channels ............................................................ 61
  3.7.1. Sequence of PKE-based Channel Compositions ................................................. 63
  3.7.2. Sequence of SKE-based Channel Compositions ................................................... 63
3.8. Differences between PKE and SKE Schemes ......................................................... 64
  3.8.1. Differences Arising from the Forwarding Trust ..................................................... 66
  3.8.2. Differences Arising from the Message Privacy Trust ............................................. 68
  3.8.3. Differences Arising from the Trust Against Masquerading ..................................... 68
  3.8.4. Differences Due to the Key Privacy Trust ............................................................ 68
  3.8.5. Differences with Regard to Replication ............................................................. 69
3.9. Conclusion ............................................................................................................. 71

Chapter 4. SYNTHESIS ................................................................................................. 72
  4.1. Introduction ............................................................................................................. 72
  4.2. Necessity of Fast Channel Establishment Procedures in a VLDS ......................... 73
  4.3. Channel Composition Algorithms .......................................................................... 79
    4.3.1. Iterative Channel Composition ........................................................................... 80
    4.3.2. Recursive Channel Composition ....................................................................... 81
  4.4. Trust Specifications ............................................................................................... 86
  4.5. Two-Agent Trust Specifications ............................................................................. 87
    4.5.1. Iterative and Recursive Channel Composition ..................................................... 88
    4.5.2. Duality .................................................................................................................. 93
  4.6. Name Space Synthesis Given Two-Agent Trusts and Iterative Composition ............ 94
    4.6.1. The Synthesis Algorithm ............................................................................... 94
    4.6.2. Correctness and Complexity of Algorithm 4.1 ............................................... 97
    4.6.3. NP-Completeness Results ................................................................................. 99
  4.7. 3-Agent Trust Specifications .................................................................................. 106
    4.7.1. Iterative and Recursive Composition ............................................................... 107
    4.7.2. Duality ................................................................................................................ 110
  4.8. Name Space Synthesis Given Three-Agent Trusts and Iterative Composition .......... 111
    4.8.1. The Synthesis Algorithm ............................................................................... 111
    4.8.2. Leaf Node Deletion Algorithm ......................................................................... 117
    4.8.3. Independence Properties of Duplicate Elimination in Algorithm 4.3 ................. 118
    4.8.4. Correctness of Algorithm 4.3 ............................................................................ 123
    4.8.5. Complexity of Algorithm 4.3 ............................................................................ 125
  4.9. An Example ........................................................................................................... 126
  4.10. Conclusion ......................................................................................................... 128

Chapter 5. TRADING TRUST REQUIREMENTS FOR PERFORMANCE ............ 130
1.1. Motivation

A moment's reflection is sufficient to realize that computing systems are evolving into very large distributed systems that interconnect competing organizations and individuals, and even countries, using global networks (see Figure 1.1). The relationships among these entities are characterized by the need for competition and cooperation, and by inherent conflicts of interests. There are few policies that are agreeable to all of the entities, and, even in the case of policies on which all the entities agree, there are no globally acceptable administrative authorities to enforce the policies. Consequently, a very large distributed system (VLDS) spanning all these entities will be characterized by the absence of globally trusted agents. The interconnecting networks, owing to their ultra-high bandwidths [SIP86], will be capable of supporting secure and integrated, but extremely fast access to non-local resources. A scenario in which workstations and high-speed fibers replace telephones and telephone wires, and a VLDS interconnecting these workstations replaces the functions of most media (telephone, physical mail, printed media, audio and video media) is not far from the real possibilities of the medium-term future. The use of a VLDS for carrying out commercial operations such as bank transactions, monetary transactions, and airline flight reservations is being seriously explored. Consequently, the issues of security and trust become critical in a VLDS. A VLDS must maintain the security and autonomy of its components without restricting the sharing of resources and without requiring its components to place global trust in any entity.

Figure 1.1: A sample space of entities that might be spanned by a future distributed system
To build such distributed systems that allow partial trust, it is necessary first to understand precisely what trust consists of, and then to characterize it. In literature, the term "trust" is used frequently but rarely defined [CGH81, EKW74, Sal74, Wei69] [Dif82, KIP79, Lan81]. The kind of trust that underlies expressions such as "Alice trusts Bob" has never been adequately characterized. Unless we make an effort to investigate trust and security, their inadequate understanding will be a major obstacle to the commercial realization of very large distributed systems.

1.2. Trust Relationships, Naming and Secure Communication in Distributed Systems

To see how trust is needed in a distributed system, consider the case of secure communication between two users, Alice and Ibaraki. Alice (actually, a process belonging to Alice) on a host $H_A$ needs to communicate securely with Ibaraki on another host $H_B$ (see Figure 1.2(a)).

In distributed systems, secure communication between two agents is based on the notion of a logical secure channel (or just a channel) between the two agents. A secure channel has associated with it algorithms for securely sending and receiving messages on it. To communicate securely, Alice must establish a secure channel to Ibaraki. Secure channels are based on encryption, and hence, to establish a secure channel to Ibaraki, Alice must obtain the encryption key of Ibaraki [Den82, FNS75]. In a large distributed system, the database of encryption keys cannot be replicated at each host, and hence encryption keys are stored and managed by authentication servers [Lu86, NeS78, Ter]. Thus Alice must obtain Ibaraki’s encryption key from an authentication server (Figure 1.2(b)). Since the security of communication between Alice and Ibaraki depends on the validity of the encryption key that Alice obtains from the authentication server, informally we can say that Alice is placing trust in the authentication server with respect to Ibaraki.

Figure 1.2(c) shows a more general scenario in which the nodes labeled IBM, IBM-J, USA, SONY-US, JAPAN and SONY are authentication servers managed by the respective organizations. In the sequel, we will use the term agent to abstractly denote either a user or an authentication server. We can draw some generalizations regarding secure channel establishment between agents, such as that between Alice and Ibaraki. In any system, given a set of existing secure channels (denoted by solid lines in Figure 1.2(c)), the only way to establish a new secure channel is by composing adjacent secure channels. Intuitively, considering the example shown in Figure 1.2(c), to establish a new channel to Ibaraki, Alice must receive Ibaraki’s encryption key on one of Alice’s existing channels, which in this case is Alice’s channel to IBM. If we go back a step in the itinerary of Ibaraki’s key, Ibaraki’s key must have arrived at IBM on one of the channels incident on IBM, and so on. Thus for Alice to establish a channel to Ibaraki, a sequence of adjacent channels must exist forming a path between Alice and Ibaraki. In fact, we will formally show in Chapter 3 that any secure channel establishment consists of a sequence of channel compositions, with each composition involving two adjacent channels.

There are two distinct problems in channel composition: (1) which are the channels to be composed, and (2) in what order should these channels be composed. These two aspects of channel composition give rise to different trust requirements in a distributed system. These aspects are related to naming because of a reciprocal association between naming and channel establishment: (1) resolving a name (i.e., translating human-readable names of agents to attributes such as their location) requires the establishment of channels to name servers, and (2) channel establishment requires translating names of agents to their encryption keys, for which a name resolution procedure must be used. Thus, even though naming and channel establishment can be realized separately, combining them into a single mechanism can result in higher
performance. In the sequel, we will use the term name server synonymously with the term authentication server.

It should be noted that the problems of trust in a distributed system are interesting in their own right, irrespective of whether the distributed system is large or not. However, efforts to solve those problems are justified by their crucial importance to very large distributed systems. Small distributed systems, such as those that do not span more than one organization, do not have significant trust problems. In distributed systems that span more than one organization but with a small number of organizations, trust problems are sufficiently simple that they can be solved using informal methods. Only when a distributed system spans a large number of autonomous organizations does the need for systematically characterizing the trust relationships in the system arise. In fact, only in large distributed systems is there a need for storing encryption keys at name servers. In small distributed systems, the database containing the encryption keys of all agents can be replicated at each host, and the problem of trust in secure communication disappears.

1.3. Relation to Previous Work

Most existing and proposed distributed systems make trust-related assumptions, though often implicitly. A sample of these assumptions is as follows:

- All system-level components trust one another. Hosts trust each other, and agents trust all name servers [Che84, MuT84, STB86]. Systems such as Amoeba [MuT84] further assume that both the network and the network interfaces are secure.

Figure 1.2: Problem of trust in a VLDS. (a) Logical secure channel. (b) Adjacent channel composition. (c) Channel establishment as a sequence of adjacent channel compositions.
Hosts may not trust each other, but name servers are globally trusted [BLN82, SJR86]. Clearly these trust-related assumptions are incompatible with one or more of the characteristics of very large distributed systems we projected in Section 1. Many of the trust-related assumptions made in current systems are based on the supposition that each distributed system has a logically centralized administrative authority that can enforce policies and punish violators. The absence of such a single administrative authority in a VLDS has significant trust-related consequences. The agents in a VLDS have to cope with the inherent existence of lack of trust and the associated possibilities for losses. A recent case involving a foreign company over which the United States was unable to enforce its laws is an appropriate example. Thus, a VLDS must not only allow lack of global trust but also incorporate explicit patterns of lack of trust.

The Arpanet [8la, 8lb] is similar to a VLDS in that it spans a large number of organizations and individuals, and has a global name server [TPR84]. However, the Arpanet does not have global authentication mechanisms, and hence it does not provide secure integrated access to non-local resources. An agent must have independent accounts and passwords on each host that it uses. The total lack of integration of services in the Arpanet prevents it from being a true VLDS.

Birrell et al. [BLN86] consider naming and authentication in large distributed systems without global trust. They suggest that an agent must be able to specify the sequence of name servers to be trusted for establishing a channel. However, they neither give a precise meaning to the notion of trust nor provide a precise analysis of the trust properties of various secure channel establishment mechanisms. We give a precise notion of trust in Chapter 2, in Chapter 3 we analyze the trust properties of the various channel establishment mechanisms, and in Chapter 5 we investigate trust properties of various network protocols for channel establishment. These analyses reveal surprising differences among, and limitations of, the various mechanisms and protocols with regard to their trust properties. For instance, we show that, if secure channels are based on single key encryption (rather than public key encryption), handling channel establishment at the host-to-host level of the network protocol hierarchy requires global trust.

Birrell et al. [BLN86] also suggest that, in order to choose the name servers trusted in establishing a new channel, the channels to be composed must be chosen appropriately, and the burden of choosing these channels is left to the users. In Chapter 4, we will develop algorithms for synthesizing name servers so as to satisfy an a priori given set of trust specifications of agents. Thus, the user is no longer burdened with choosing channels based on whom he or she trusts; the design of the name server automatically takes care of the user's trust relationships.

Popek and Kline [KIP79, PoK79] compare trust properties in secure communication using single key encryption and public key encryption, and conclude that the two have the same trust properties. In Chapter 3, we will show that this conclusion, which they arrived at using an informal notion of trust, is incorrect.

1.4. Outline of the Thesis

The goal of this thesis is to develop techniques by which distributed systems can be synthesized so as to satisfy a given set of trust specifications. Such a synthesis will have to employ some basic channel composition mechanisms. Before we can employ a channel composition mechanism in a synthesis, we must know the trust relationships inherent in (and hence required by) the mechanism. Thus, we must analyze the various channel composition mechanisms and investigate the trust relationships they require. But before we can analyze a mechanism from
the viewpoint of the trust relationships it requires, we must precisely define what we mean by trust. Thus, the dissertation will consist of the following sequence of steps.

Theory of Trust: To capture and incorporate lack of trust into a distributed system, it is necessary first to understand precisely what trust consists of, and then to characterize it. Basic foundations are necessary to clarify our understanding and to reason adequately about lack of trust in distributed systems. A clear definition of trust in a distributed system can reveal subtle distinctions that may not be otherwise apparent. Formal descriptions of security have traditionally avoided any explicit treatment of trust. It is desirable to unify security and trust into a single theory. To satisfy all these requirements, we develop an axiomatic theory of trust in Chapter 2.

Analysis: Formal analysis of trust can offer insights into the basic structure and the limitations of mechanisms with regard to their trust requirements. Zero-trust mechanisms may be possible. Chapter 3 analyzes trust relationships in various channel composition mechanisms. We develop methods for reasoning about trust requirements in various mechanisms, and discuss methods for arriving at the minimal trust requirements in a given distributed system. While doing so, we will encounter some surprising differences among various mechanisms with regard to their trust requirements.

Synthesis: The eventual goal of the thesis is to provide algorithms for synthesizing distributed systems so as to satisfy a priori trust specifications of agents. As was shown in Section 2, trust requirements arise in naming. Chapter 4 develops algorithms for synthesizing name servers from trust specifications. A sample set of trust specifications may be as follows: "Alice never sends false information to Bob about Fred, and Bob never sends false information to Riccardo about Alice. Fred and Riccardo cannot be trusted to secretly store information about Bob or Alice. Fred cannot be trusted for any information about any agent that Alice or Bob trust for information about Riccardo." Or it may involve organizations as in, "IBM trusts DEC for Hita-chi but not for AT&T".

No synthesis methodology is complete without performance considerations. Under some conditions, trust requirements and performance of channel establishment mechanisms can be traded for each other. Chapter 5 shows that, if channel composition is based on public key encryption, slightly increasing the number of trust relationships that are satisfied can greatly increase the performance of channel establishment mechanisms, and that the additional trust relationships still form a subset of the set of trust specifications from which the distributed system name space has been synthesized. We also show that, if channel composition is based on single key encryption, global trust, which may not be satisfied in the system name space, is required for significant improvements in performance.

Clearly the goals of the thesis are pragmatic, but the approach is partly formal. This work was carried out as part of the DASH project at Berkeley [AFV87c], which is investigating issues in the design and implementation of very large distributed systems.
CHAPTER 2

AXIOMATIZATION OF TRUST

This chapter develops an axiomatic theory of trust in distributed systems. The chapter discusses what it means to develop a logic or a theory, and shows that modal logics of belief with their semantic interpretation based on the possible worlds semantics of Kripke, are appropriate as a starting point for a theory of trust. We review a modal logic of belief that is an enhancement of propositional logic with a belief operator, and construct a model of a distributed system so that the logic is sound and complete with respect to the model. Any sentences in the logic may then be added to the logic as axioms, and these axiomatic sentences are considered as trust specifications. The logic and the trust specifications, together with the model, constitute a formal theory of trust for the target distributed system. However, a theory of trust is of practical significance only if abstract trust specifications can be implemented in a real distributed system. We present formal techniques for synthesizing protocols that are necessary and sufficient for implementing a given trust specification in a distributed system.

2.1. Introduction

To build distributed systems that capture and account for lack of global trust, it is first necessary to understand precisely what trust consists of, and then to characterize it. Basic foundations are needed to clarify our understanding and to reason adequately about lack of trust in distributed systems. This chapter develops an axiomatic theory of trust for such systems. In Section 2.2, we discuss basic notions of what it means to develop a logic or a theory, and show that modal logics of belief are appropriate as bases for a theory of trust. Section 2.3 reviews modal logics of belief, and Section 2.4 presents a distributed system model. Section 2.5 presents a modal logic of belief that is sound and complete with respect to this model. In Section 2.6, we develop a formal theory of trust, and, in Section 2.7, we present methods for synthesizing protocols that implement a given abstract trust specification. Finally, Section 2.8 concludes the chapter.

2.2. An Approach to Axiomatization

2.2.1. Theories

Our first step, as we saw, is to capture the highly informal notion of trust into a formal theory. The term theory is used here in the sense of Mendelson [Men87]. A theory is based on a logic. Briefly, a logic consists of a language, which defines the set of well formed formulas (WFFs) or valid sentences, a set of axioms, and a set of rules of inference. An axiom is a WFF and a rule of inference is a transformation from one WFF to another. (We will shortly define the roles played by axioms and rules of inference in a logic.) The logic provides a framework for reasoning and abstracts some fundamental notions. A theory consists of a logic enhanced with a set of assumptions that are particular to the real-world problem being modeled by the theory. These assumptions are called proper axioms. To develop a theory of trust, we must first start with a logic on which to base the theory.

A proof starts out from the axioms and the proper axioms, and repeatedly uses rules of inference to arrive at a WFF. A WFF that is the result of a proof is called a theorem. The
theory is said to be consis tent if for no WFF are both the WFF and its negation theorems in the logic.

Each logic (theory) has a set of models which provide the semantic interpretation for the logic (theory). The semantic interpretation is outside the logic and corresponds to the real-world situation being modeled by the logic (theory). Assuming two-valued logics, a semantic interpretation of a WFF under any assignment to variables in the WFF yields one of the two values: true or false. A WFF whose semantic interpretation is true in a model under any assignment to the variables in the WFF is said to be valid in the model. A WFF whose semantic interpretation is true in a model under at least one assignment to the variables is said to be satisfiable in the model.

A logic (theory) is said to be sound with respect to a model if and only if the theorems in the logic (theory) are valid in the model. A logic can also be shown to be sound with respect to a model if and only if the axioms of the logic are valid in the model and the rules of inference are validity preserving. It can be shown that a theory is sound with respect to a model if and only if the logic on which the theory is based is sound with respect to the model and the proper axioms of the theory are valid in the model. Thus, the set of models with respect to which a theory is sound is a subset of the set of models with respect to which the logic the theory is based on is sound. A logic (theory) is said to be complete with respect to a model if and only if all WFF that are valid in the model are theorems in the logic (theory).

To develop a formal theory of trust, we must first start with a logic on which to base the theory.

2.2.2. A Logic for Trust

What kind of a logic is suitable for modeling trusts in distributed systems? To answer this question, consider the simplest case of secure communication.

To communicate securely, agents encrypt messages using keys belonging to other agents. We may say that each agent must make assertions of propositions of the form owner(key of the other agent, the other agent), where the semantic interpretation of owner(key, agent) is that it returns true if and only if key belongs to agent. Thus, it seems appropriate to consider propositions of this type as forming the atomic propositions of a logic of trust. (An atomic proposition is a proposition representing a basic notion in the model at a given level of abstraction.)

In a system in which the complete database of keys is securely replicated at every agent using mechanisms external to the system (such as telephone conversations between agents, or couriers), two agents can communicate securely without placing trust in a third agent. In a distributed system with distributed name servers, an agent has to obtain the keys of other agents from name servers.

Figure 2.1 represents a distributed system in which there are agents $A_i$, $A_j$, and $A_k$, and where $A_j$ is a name server. There are two channels in the initial state, $A_i$-$A_j$, and $A_j$-$A_k$. $A_k$ sends a message $msg_{kj}$ containing its key $key_k$ to $A_j$. When $A_j$ sends a request to $A_j$ asking for $A_k$'s key, $A_j$ sends a message $msg_{ji}$ to $A_i$ containing $key_k$. If the proposition owner($key_k$, $A_k$) cannot be proved false (i.e., if it is satisfiable and hence possible) at $A_i$, $A_i$ may now consider accepting $key_k$. However, $A_i$ is not prepared to use $key_k$ for secure communication to $A_k$ unless it is able to prove that owner($key_k$, $A_k$).

Suppose, to start with, that we make no assumptions whatsoever about security or about the validity of the messages sent by one agent to another, and that the belief in a proposition is used to represent an attitude by which the believer may not be able to prove the proposition
valid but thinks the proposition might be valid. Thus, the belief of an agent corresponds to the notion of having in the agent's database a proposition that is satisfiable so that, even though there may be a measure of uncertainty about the validity of the proposition, the agent has strong reasons to conjecture that the proposition might be valid in the real world. Specifically, let the belief in a proposition \( p \), denoted by \( B_i p \), indicate a relationship between an agent \( A_i \) and a proposition \( p \) such that (a) \( A_i \) may not be able to assert the truth of \( p \), (b) \( A_i \) cannot prove \( p \)'s falsity, and (c) \( A_i \) expects and desires \( p \) to be true. Thus, there may be a measure of uncertainty about \( p \)'s validity, but there is a high likelihood that \( p \) might be valid.

Using this notion of belief, we might say that the message from \( A_k \) to \( A_j \) creates a belief \( B_j \text{owner}(key_k, A_k) \). The message from \( A_j \) to \( A_i \) creates a belief \( B_i B_j \text{owner}(key_k, A_k) \). Thus, \( A_i \) believes that \( A_j \) believes \( \text{owner}(key_k, A_k) \). This can be extended to a scenario in which the key successively passes through several name servers during name resolution.

However, \( A_j \) is not prepared to use \( key_k \) for secure communication to \( A_k \) unless it is able to prove that \( \text{owner}(key_k, A_k) \) is valid. To infer \( \text{owner}(key_k, A_k) \) from its belief, \( A_j \) has to use some assumptions such as that \( A_k \) does not usually send a false key, the key-parts of messages \( msg_{kj} \) and \( msg_{ji} \) are identical, and so on. Such assumptions are encapsulated into the notion of trust, and are precisely abstracted by proper axioms. In this example, one possible proper axiom may be "\( B_i B_j \text{owner}(key_k, A_k) => \text{owner}(key_k, A_k) \)". While a belief is an operator, a trust is a proper axiom, i.e., a WFF that is assumed to be true in the system. Thus for instance, a belief such as \( B_i p \) denotes that \( A_i \) believes that proposition \( p \) is true, while a WFF such as \( B_i p => p \) is a trust denoting the assumption that, \( A_i \) believes in \( p \) only when \( p \) is true. Agents use trusts to make inferences about the validity of their beliefs.

It is easy to see that reasoning about trusts involves reasoning about the notion of belief, and that a theory of trust may be based on a logic of belief. After all, one of the most desirable properties of a formal theory is its ability to capture what people intend to say, and we have arrived at the suitability of a logic of beliefs in the course of making natural statements about secure communication. At this point it is useful to pause, and review the kinds of logics available to us for reasoning about beliefs in general.
Belief represents an attitude of an agent towards a proposition. A logic for expressing propositional attitudes must be able to express the appropriate relations between believers and attitudes [Hin62]. Classical first order logic does not handle these attitudes properly [FaH85, HaM85]. Modal logics [HuC68], which enhance propositional and first-order predicate logics with modal operators such as belief, have been found suitable for modeling belief.¹

2.3. Modal Logic of Belief: A Review

2.3.1. Syntax

In the language of the modal logic of belief, agents are named $A_1, \ldots, A_m$, and the atomic propositions are denoted by $p, q, \ldots$. Let "\&", "\lor" and "\neg" denote conjunction, disjunction and complementation, respectively. For $i = 1, \ldots, m$, let $B_i$ be an operator, read as "agent $A_i$ believes". The set of WFFs is the smallest set that contains atomic propositions, is closed under boolean connectives, and contains $B_i F$ ($i = 1, \ldots, m$) if it contains $F$. Since quantified modal logics are not well understood, we restrict ourselves to using the propositional modal logic for belief. However, when a variable $x$ varies over a finite set $X = \{x_1, \ldots, x_n\}$, "$\forall x F(x)$" is used as a short-hand notation for $F(x_1) \land \cdots \land F(x_n)$. Thus, if $F$ is a WFF, and $x$ varies over a finite set, $\forall x F(x)$ is a WFF.

2.3.2. Semantics

Unlike classical logic operators, modal operators such as belief do not allow a truth-functional semantic interpretation. (An operator is truth-functional if and only if, given any WFF that is a result of applying the operator to some arguments, the truth value of the WFF can be deduced solely from the truth values of the arguments. Belief is not a truth-functional operator because, belief in a proposition may be true or false irrespective of the truth value of the proposition.) Thus, modal logics use a possible-worlds semantics [HaM85, Kri63] in which the notions of possibility and necessity are used, and the notion of a possible world is used in the semantic interpretation.² A set of possible worlds is postulated, and a belief is true if it is true in a set of possible worlds. The real world may be one of the possible worlds.

An agent's belief arises primarily because of the agent's ignorance about the global state of the distributed system. An agent's state of belief relates to the level to which the agent can determine the system's global state based on its local state. In each global state of the system, one can associate with each agent a set of possible global states that are determined as follows: if the agent's beliefs are true, any of them could possibly be the real global state. In other words, based on its local state, an agent cannot determine the real global state that it is in; it can only conclude that some global states are possible. An agent believes $p$, denoted by $B_i p$, if and only if $p$ is true in all the global states that the agent considers possible. An agent does not believe $p$ if and only if, in at least one of the global states that the agent considers possible, $p$ is not true. Since this semantic interpretation of belief uses the notion of possible global states, it is called the possible worlds semantics [Kri63].

¹ As we will see later, modal logics use the notions of possibility and necessity. In medieval logic, possibility, necessity, and so on, were thought of as modes in which a proposition could be true or false.

² Halpern and Moses wrote an excellent paper on modal logics of belief and knowledge [HaM85]. The review of modal logics presented in Section 2.3 of this chapter is based heavily on this paper.
In a future section we show that the addition of any proper axiom to a logic (giving rise to a theory) requires the construction of a new set of models from the old set of models for which the logic was sound and complete. This is to ensure that the theory is sound and complete for the new set of models. If this cannot be ensured, then the theory may be ill suited for the new set of models of the system, i.e., some statements that are provable as being true in the theory may be actually false in the system, and some that are true in the system may be provable as being false in the theory. Thus, it is necessary to ensure that the theory is sound and complete for the new set of models. In the modal logic approach, by making minor changes to the possible worlds semantics, we can capture different problem situations. By imposing various constraints on what global states are considered possible by an agent in a given real global state, one can capture a number of interesting notions of belief. For example, if the relation between the global state and the set of possible global states is restricted to be transitive, then an agent believes that it believes \( p \), if it believes \( p \). Since it lends itself to easy translation between a proper axiom of a theory and its semantic interpretation in the model, the possible worlds approach is a powerful tool for developing theories. Thus, we say that this approach is customisable, i.e., with little effort a theory can be derived from a logic for a given security environment. We have chosen this approach for our theory of trust.

2.3.3. The Kripke Structure: A Formal Semantic Interpretation of Belief

Kripke [Kri63] introduced what is known as a Kripke structure as a formal model for possible worlds semantics. Let \( S \) be the set of all global states, and \( \Phi \) be the set of all atomic propositions. A Kripke structure \( K \) is a tuple \((S, \pi, \rho_1, \ldots, \rho_m)\), where \( \pi \) is a truth assignment to the atomic propositions of \( \Phi \) for each global state \( s \) in \( S \) (i.e., \( \forall p, s \text{ such that } p \in \Phi \text{ and } s \in S, \pi(s, p) \in \{\text{true, false}\} \)), \( m \) is the number of agents, and \( \rho_i, i = 1, \ldots, m \), is a relation on the global states in \( S \). \( \rho_i \) is \( A_i \)'s possibility relation; \((s, t) \in \rho_i \) if and only if in global state \(^3 s \) \( A_i \) considers the global state \( t \) as possible.

We will now review a formal definition of the truth of a WFF given using the relation \( \models \), a relation between states and WFFs [HaM85]. ‘‘s \models p’’ stands for ‘‘p is true in s’’ (which is equivalent to, ‘‘w satisfies s’’).

\[
\forall p \in \Phi, s \models p \text{ if and only if } \pi(s, p) = \text{true}
\]
\[
s \models p \land q \text{ if and only if } s \models p \text{ and } s \models q
\]
\[
s \models \neg p \text{ if and only if } s \not\models p
\]
\[
s \models B_i p \text{ if and only if } \forall t \text{ such that } (s, t) \in \rho_i, t \models p
\]

The last definition above formalizes the idea that an agent \( A_i \) believes \( p \) in global state \( s \) if and only if \( p \) is true in all the states that \( A_i \) considers possible when the system is in state \( s \). A WFF \( p \) is valid (or satisfiable) if and only if \( s \models p \) for all states \( s \) (or for some state \( s \), respectively). It may be observed that \( p \) is satisfiable if and only if \( \neg p \) is not valid.

In order to use this logic for reasoning about communication security, we have first to relate distributed systems to Kripke structures.

\(^3\) In the sequel, we use the term state synonymously with the term global state except when explicitly mentioned that it is local state. There is a distinction between a world and a global state, but, in the sequel, this distinction is unimportant, and we use them synonymously.
2.4. A Distributed System Model

2.4.1. States

A distributed system can be modeled as a set of agents communicating with each other via messages. The state of the distributed system consists of the states of all its agents. The state of an agent consists of its message history, which is the sequence of messages received or sent by the agent. A message in the message history consists of a WFF, a sender, and a receiver. At least one of either the sender or the receiver is the agent itself. Messages that are not of this format are not of interest to the agent and are not interpreted by the agent. To start with, we make no assumptions about the security in the system: i.e., an agent may send or receive any message, may masquerade as any other agent, and so on. An agent may impose any conditions for accepting a message, such as a test for message authenticity. A non-accepted message does not become a part of the agent's message history. The state of an agent uniquely determines the agent's beliefs.

Having defined states, we now define the possibility relations in a Kripke structure.

2.4.2. Possibility Relations

Consider a global state $s$ in which the state of agent $A_i$ is $s_i$. The following definitions will be used:

- $MS(s_i, A_j) =$ the sequence of messages in the message history of $A_i$ in state $s_i$ that were sent to $A_j$.
- $MR(s_i, A_j) =$ the sequence of messages in the message history of $A_i$ in state $s_i$ that were received from $A_j$.
- $BR(s_i, A_j) =$ the set of WFFs sent in the message sequence $MR(s_i, A_j)$.
- $Bel(s_i) =$ the set of beliefs of $A_i$ in state $s_i$.

Let the symbol $\leq$ denote the subsequence relationship between sequences or the subset relationship between sets. The possibility relation $\rho_i$ consists of all pairs of states $s$ and $t$ such that $\forall j, j \neq i:$

- (PC1) $s_i = t_i$, i.e., the $i$th components of state $s$ and state $t$ are the same,
- (PC2) $MR(s_i, A_j) \leq MS(t_j, A_i)$ (i.e., there is an authenticated channel from $A_j$ to $A_i$), and
- (PC3) $BR(s_i, A_j) \leq Bel(t_j)$ (i.e., $A_j$ has not lied about its beliefs to $A_i$).

We will refer to these three conditions as the possibility conditions with respect to $s_i$. Note that each possibility relation is specific to an agent.

In effect, the possible states from the viewpoint of $A_i$ are those states of the distributed system in which all the messages in $A_i$'s message history are authentic and the senders of those messages have not sent false messages to $A_i$ (see Figure 2.2). Thus, $A_i$ holds a limited optimistic view of the possible distributed system states: the possible states are secure as far as $A_i$ is concerned. In the real state, agents might have sent false messages to, or masqueraded to $A_i$. Note that states in which $A_j$ has masqueraded to $A_k$, $k \neq i$, or sent false beliefs to $A_k$ are possible from the viewpoint of $A_i$. As we will see, we will make use of trusts to turn a possible secure state into the real state.
2.4.3. Belief Acquisition

What should agent $A_i$’s beliefs be? The semantic interpretation of a belief in a state $s_i$ is that the believed proposition is true in all the states in the possibility relation $p_i$ corresponding to $s_i$, i.e., in all the possible states. The system is not necessarily in one of the states considered possible by $A_i$, and hence the believed proposition need not be true in the real world. Any event in the system may trigger a belief acquisition or a belief revision. The events of interest depend on the particular application. For simplicity, we only consider the reception of a message as resulting in a belief acquisition. When $A_i$ receives a WFF $f$ from an agent $A_j$, $A_i$ adds the belief $B_iB_jf$ to its belief database if and only if $B_jf$ is consistent with the beliefs that $A_i$ has previously acquired as a consequence of a message from $A_j$ (i.e., if $B_jf$ cannot be proved from $A_i$’s current beliefs). Thus, incoming messages may cause an agent to add to its beliefs.

In the logic, WFFs received by $A_i$ from two different agents will not be inconsistent with each other. For example, let $A_j$ send a WFF $f$ to $A_i$, let $A_j$ send $\neg f$ to $A_k$, and let $A_k$ send $B_j\neg f$ to $A_i$. $B_iB_jf$ and $B_iB_kB_j\neg f$ are consistent with each other, and $A_i$ will add both the beliefs. The possible states from the viewpoint of $A_i$ in this example include those in which $B_jf$ is true at $A_j$ and $A_j$ has sent a WFF $\neg f$ to $A_k$ (thus, $B_j\neg f$ is true at $A_k$), as well as those in which $A_j$ has sent a WFF $f$ to $A_i$. To see why agents may need to send beliefs in a real system, in the example of Section 2.2.2 in which $A_i$ establishes a channel to $A_k$ using a name server $A_j$, $A_k$ sends its key $key_k$ to $A_j$, and the reception of $key_k$ creates a belief $B_jowner(key_k,A_k)$ in $A_j$. $A_j$ sends this belief to $A_i$, and this creates a belief $B_iB_jowner(key_k,A_k)$ in $A_i$. Notice that $A_j$ must send its belief, and not owner($key_k,A_k$) to $A_i$, because, that $key_k$ belongs to $A_k$ is only a belief of $A_j$, and it is not certain if the key really belongs to $A_k$. Thus for instance, since we are not making any assumptions about the security behavior of various agents at this point, some other agent may have sent $key_k$ masquerading as $A_k$. Figure 2.2: A possible state from the viewpoint of $A_i$ and the real global state. (a) A possible global state from agent $A_i$’s viewpoint: the possible global state consists of the real local state $s_i$ of $A_i$ and a possible local state $t_j$ of $A_j$. In this possible global state, $A_i$ has sent messages $msg_1$ and $msg_2$ to $A_i$ and the WFFs that $msg_1$ or $msg_2$ contain are true. (b) The real global state consists of the real local states $s_i$ and $s_j$. In $s_j$, $A_j$ has sent message $msg_1$, the WFF contained in $msg_1$ is not true ($A_j$ has lied to $A_i$), and $A_j$ did not send $msg_2$. Masquerading as $A_j$, some other agent has sent $msg_2$ to $A_i$. 

---
2.4.4. Sending Beliefs

What beliefs can an agent $A_i$ send to another agent $A_j$? $A_i$ can send any WFF, whether or not the WFF is a belief of $A_i$. However, $A_j$ accepts a belief sent by $A_i$ only if the belief is consistent with the beliefs that $A_j$ has previously received from $A_i$. Thus, $A_i$ is not allowed to change its mind. It is not necessary that a WFF sent by $A_i$ to $A_j$ (1) be one of $A_i$'s beliefs, (2) be consistent with $A_i$'s beliefs, or (3) be consistent with the beliefs that $A_i$ sends to other agents.

2.5. A Logic of Belief for the Distributed System Model

The axioms and the inference rules for a logic of belief depend on the properties of the possibility relations. The properties of interest are transitivity, euclidean property, serial property, and reflexivity. A relation $\rho$ is transitive if and only if $\forall s, t, u, ((s, t) \in \rho$ and $(t, u) \in \rho) \Rightarrow (s, u) \in \rho$. A relation $\rho$ is euclidean if and only if $\forall s, t, u, ((s, t) \in \rho$ and $(s, u) \in \rho) \Rightarrow (t, u) \in \rho$. A relation $\rho$ is serial if and only if $\exists s, t$ such that $(s, t) \in \rho$. A relation $\rho$ is reflexive if and only if $\forall s, (s, s) \in \rho$. We now show that the possibility relations for our distributed system model satisfy exactly the first three of these properties.

\[ \forall i, i = 1, \ldots, m: \]

(1) **Transitivity**: for any states $s, t,$ and $u$, suppose $(s, t)$ and $(t, u)$ are in $\rho_i$. By the possibility conditions, $s_i = t_i$ and $t_i = u_i$. Thus, $s_i = u_i$. For all $j, j \neq i$, $t_j$ satisfies the last two possibility conditions with respect to $s_i$, and $u_j$ satisfies the last two possibility conditions with respect to $t_i$. However $s_i = t_i$, and hence $u_j$ satisfies the last two possibility conditions with respect to $s_i$. Thus $(s, u)$ belongs to the possibility relation, and $\rho_i$ is transitive.

(2) **Euclidean property**: consider any two pairs $(s, t)$ and $(s, u)$ in $\rho_i$. We have $s_i = t_i$ and $s_i = u_i$. Thus, $t_i = u_i$. For all $j, j \neq i$, $u_j$ satisfies the last two possibility conditions w.r.t. $s_i$. Since $s_i = t_i$, $u_j$ satisfies the last two possibility conditions w.r.t. $t_i$. Thus, all the three possibility conditions are satisfied for the pair $(t, u)$. Hence $(t, u)$ belongs to $\rho_i$, and consequently $\rho_i$ is euclidean.

(3) **Serial property**: the possibility conditions are constructive. Thus, for every state $s_i$ of $A_i$, for every $j, j \neq i$, $t_j$ can be constructed directly from the possibility conditions and independently of any $k, k \neq j, k \neq i$. This is because the possibility conditions impose constraints only on the messages between $A_i$ and other agents, and on the beliefs of other agents. There are no constraints on the messages between agents $A_j$ and $A_k$, if $j \neq i$ and $k \neq i$. Thus, for every state of $s, \exists t$ such that $(s, t) \in \rho_i$. Hence $\rho_i$ is serial.

The actual state may not be one of the possible states. Thus, the possibility relation is not reflexive, and a believed WFF may not be true in the real world.

Given these properties of the possibility relations, an axiom scheme must be chosen for the modal logic of belief. Several axiom schemas are possible. An axiom schema not only provides a sound and complete formal system but also determines whether the satisfiability of WFFs is decidable or not in the logic, and hence must be chosen carefully. The following axiom schema is known to provide a sound and complete characterization of our notion of belief, and a decidable satisfiability of WFFs [HaM85]. The axiom schema consists of the following axioms:
for all $i, i = 1, \ldots, m$:

A1. All substitution instances of propositional tautologies.
A2. $B_i p \land B_i (p \Rightarrow q) \Rightarrow B_i q$.
A3. $B_i p \Rightarrow B_i B_i p$ (introspection of positive belief).
A4. "$B_i p \Rightarrow B_i B_i p$ (introspection of negative belief).
A5. "$B_i (\text{false})$ (agent $i$ does not believe a contradiction).

and the following inference rules:

for all $i, i = 1, \ldots, m$:

R1. From $p$ and $p \Rightarrow q$ infer $q$ (modus ponens).
R2. From $p$ infer $B_i p$ (generalization).

Some of these axioms directly correspond to the properties of the possibility relations $P_1, \ldots, P_m$: A3 corresponds to transitivity, A4 to the euclidean property and A5 to the serial property.

2.6. A Theory of Trust

The beliefs that an agent has may not be true in the real world. Trusts, encoded as proper axioms, are used to derive the truth (or falsity) of beliefs. A trust is any proper axiom added to the modal logic of belief presented in Section 2.3, i.e., any WFF that is assumed to be valid in addition to the axioms in the logic. In the logic, we started with no assumptions about the security behavior of agents. In the theory, we explicitly add the necessary assumptions in a precisely codified manner, and these assumptions are regarded as trusts. The simplest trusts are implications of the form, $B_i F \Rightarrow F$, where $F$ is a WFF. Since any WFF can be regarded as a trust specification, this approach gives a lot of power and generality to expressing trust relationships [Ven88]. Security theories [Lan81], which are first-order, can be incorporated into our theory of trust.

As was observed in Section 2.2, adding proper axioms to a logic results in a theory. Let us assume that we have a theory consisting of the modal logic of belief and some trusts (which are the proper axioms). What are the effects of adding a new trust to an existing theory? If the theory is decidable 4 and the new trust can be proved as a theorem in the theory, there is no need to add the new trust. If the new trust is not a theorem, adding it gives rise to a new theory. If the new trust does not invalidate the old theory 5, the monotonicity of the logic is retained and the new theory will continue to be complete with respect to the old model. However, the new theory is no longer sound with respect to the old semantic model (i.e., the semantic model corresponding to the old theory). Thus, a new model has to be constructed so that the new theory is sound and complete with respect to the new model. Proving soundness of the new theory w.r.t. the new model is usually easy, but proving completeness is more often than not cumbersome, non-intuitive, and difficult. However, the possible worlds semantic model is highly amenable to incremental modification such that the new theory is sound and complete with respect to the modified model. The modification consists of adding new constraints to possibility relations $P_1, \ldots, P_m$ in the Kripke structure (this will be illustrated in the next section). This is exactly the reason why the possible worlds model was chosen: it provides a powerful

---

4 A theory is decidable if, for any WFF in the theory, it can be determined whether the WFF is satisfiable or not.

5 The following three statements are equivalent to each other: (1) a WFF does not invalidate a theory, (2) the negation of a WFF cannot be proved as a theorem, (3) a WFF is satisfiable in the theory.
and flexible framework for customizing a logic.

A trust specification can be thought of as an abstract representation of a policy in the distributed system. The next step is pragmatic: Given a trust policy, what are the protocols necessary to ensure that the policy holds in the system? The following section illustrates the method for synthesizing protocols from a given abstract trust specification.

2.7. Synthesizing Protocols from Abstract Trust Specifications

Consider a distributed system in which there are three agents $A_i, A_j$ and $A_k$. Let $Q$ be any WFF. Let $\forall Q, B_j B_i B_k Q \Rightarrow B_i B_j B_k Q$ be the given trust specification that is to be implemented in the distributed system. Informally, this trust specification says that, for all well-formed formulas $Q$, if $A_j$ believes that $A_i$ believes that $A_k$ believes $Q$, it is necessary that $A_i$ believes that $A_j$ believes that $A_k$ believes $Q$. This particular trust specification may not correspond to any particularly useful notion of trust in a real system. However, it serves as a good example for illustrating how an abstract trust specification can be translated into concrete distributed system protocols.

To see clearly the advantages of the formal theory of trust, let us first look at the solution that an informal analysis might yield. For the trust to hold, $A_i$ has to believe $B_j B_i B_k Q$ before $A_j$ believes $B_i B_j B_k Q$. $A_i$'s belief can be created by a message from $A_j$ containing $B_i B_j B_k Q$. Thus, for the trust to hold, $A_j$ sends a message containing $B_i B_j B_k Q$ to $A_i$ before $A_i$ sends a message containing $B_i B_k Q$ to $A_j$.

Let us now see what the formal techniques yield. The outline of the formal method is as follows: Given a WFF such as $B_j B_i B_k Q \Rightarrow B_i B_j B_k Q$ as the trust specification, we semantically interpret it. For this WFF, interpreting the beliefs in the antecedent and the consequent yields possibility relations, which contain pairs of states. For the consequent to be true whenever the antecedent is true, the possibility relations of the consequent must be subsets of those of the antecedent. Possibility relations are nothing but sets of pairs of states, and hence we obtain relationships between the set of states of the antecedent and the set of states of the consequent. States are nothing but message histories, and thus we obtain relationships between message histories of agents, which with some more manipulation are reduced to such conditions as, for instance, that an agent must send/receive a particular message before or after sending/receiving some other message, and so on.

The given trust specification is required to be true in all system states. Let the state of the system be $s$, and the states of $A_i$, $A_j$ and $A_k$ be $s_i$, $s_j$ and $s_k$ respectively (see Figure 2.3). If the antecedent $B_j B_i B_k Q$ of the trust specification is not true in $s$, the trust specification $B_j B_i B_k Q \Rightarrow B_i B_j B_k Q$ is trivially true in $s$. Suppose the antecedent is true in $s$. The antecedent is created by a message received by $A_j$ such that the message sender field is $A_i$ and the message contains $B_i B_k Q$. Thus, the message exchanges specified by condition $M_s$ below must have taken place in $s$:

Message Condition $M_s$: $A_j$ receives $B_i B_k Q$ from $A_i$.

Since $s$ is the real state (as opposed to a possible state) of the system, we say:

State Condition $C_s$: $s$ is the real system state.

The antecedent $B_j B_i B_k Q$ is a belief. Its semantic interpretation is that there is a possible state $t$ such that $(s, t) \in \rho_j$, and the message exchanges specified by condition $M_t$ below must have taken place in $t$:

Message Condition $M_t$: $A_i$ sends $B_i B_k Q$ to $A_j$.  

Trust Spec.: For all Q, BjBiBkQ => BiBjBkQ

Figure 2.3: The Kripke structure semantic interpretation of the trust $B_jB_iB_kQ => B_iB_jB_kQ$. The local states of $A_i$, $A_j$, and $A_k$ in the various global states have been derived using the possibility conditions PC1, PC2 and PC3. For example, in global state $t$, the local state of $A_i$ is $t_i$, the local state of $A_j$ is $s_j$, and the local state of $A_k$ is $t_k$. States $s$ and $x$ are real states. All other states are possible states.

Since $t$ is a possible state, the possibility conditions PC1, PC2 and PC3 must be satisfied in $t$:

State Conditions $C_t$:
(a) $t_j = s_j$.
(b) There is an authenticated channel from $A_i$ to $A_j$.
(c) $A_i$ sends a message $B_jB_kQ$ only if it believes $B_kQ$, i.e., only if $A_i$ has received $B_kQ$ from $A_k$. In other words, $A_i$ does not lie about its beliefs to $A_j$.

Now consider state $t$. By $M_t$ and $C_t$, $A_i$ believes $B_kQ$, i.e., $B_iB_kQ$ is true. If we further interpret $B_iB_kQ$, we obtain that in state $t$ there is a possible state $u$ such that $(t, u) \in \rho_i$, and the message exchanges specified by condition $M_u$ must have taken place in $u$:

Message Condition $M_u$: $A_k$ sends $B_kQ$ to $A_i$.

Since $u$ is a possible state, the possibility conditions PC1, PC2 and PC3 must be satisfied in $u$:

State Conditions $C_u$:
(a) $u_k = t_k$.
(b) There is an authenticated channel from $A_k$ to $A_i$. 

Antecedent

Consequent
(c) $A_k$ believes $Q$, i.e., $A_k$ has not lied about its beliefs to $A_i$.

This concludes the interpretation of the antecedent. Let us now interpret the consequent, $B_i B_j B_k Q$. For $B_i B_j B_k Q$ to be true in a real system state $x$, $A_i$ must have received $B_j B_k Q$ from $A_j$ in $x$:

**Message Condition** $M_x$: $A_i$ has receives $B_j B_k Q$ from $A_j$.

Since $x$ is a real state of the system, we say:

**State Condition** $C_x$: $x$ is the real state of the system.

The belief $B_i B_j B_k Q$ of the consequent is interpreted in exactly the same way as that of the antecedent, and we obtain that there must be a possible state $y$ such that $(x, y) \in \rho_i$, and the message exchanges specified by $M_y$ below must have taken place in $y$:

**Message Condition** $M_y$: $A_j$ sends $B_j B_k Q$ to $A_i$.

Since $y$ is a possible state, the possibility conditions must be satisfied in $y$:

**State Conditions** $C_y$:

(a) $y_t = x_t$.

(b) There is an authenticated channel from $A_j$ to $A_i$.

(c) $A_j$ believes in $B_k Q$, i.e., $A_j$ has received $B_k Q$ from $A_k$. In other words, $A_j$ has not lied about its beliefs to $A_i$.

Interpreting $B_i B_j B_k Q$ in $y$ yields that there must be a possible state $z$ such that $(y, z) \in \rho_j$, and the following message exchange must have taken place in $z$:

**Message Condition** $M_z$: $A_k$ sends $B_k Q$ to $A_j$.

Since $z$ is a possible state, the possibility conditions must be satisfied in $z$:

**State Conditions** $C_z$:

(a) $z_k = y_k$.

(b) There is an authenticated channel from $A_k$ to $A_j$.

(c) $A_k$ believes $Q$ is true, i.e., $A_k$ has not lied about its beliefs to $A_j$.

This concludes the interpretation of the consequent.

Let $\rho_{i-rhs}$, $\rho_{j-rhs}$, and $\rho_{k-rhs}$ denote the possibility relations of $A_i$, $A_j$, and $A_k$, respectively, on the antecedent side, and $\rho_{i-rhs}$, $\rho_{j-rhs}$, and $\rho_{k-rhs}$ denote the possibility relations of $A_i$, $A_j$, and $A_k$, respectively, on the consequent side. The trust specification requires that the consequent be true whenever the antecedent is true. Since the beliefs of the antecedent and consequent are determined by their respective possibility relations $^6$, the possibility relations required for the consequent must be subsets of the possibility relations required for the antecedent. This yields two constraints, R1 and R2:

**Constraint R1**: $\rho_{i-rhs}$ must be a subset of $\rho_{i-rhs}$: $\rho_{i-rhs}$ contains $(x, y)$, and $\rho_{i-rhs}$ contains $(t, u)$. Thus, the set of pairs of states, $\{(x, y)\}$ must be a subset of $\{(t, u)\}$. Thus set of states, $\{x\}$ must be a subset of $\{t\}$, and $\{y\}$ must be a subset of $\{u\}$. $^7$

---

$^6$ See Section 2.3.3.

$^7$ Sets of states are used in place of single states because, the various state and message conditions do not uniquely specify the states, rather they define sets of states.
subset of a set of states \( \{r_2\} \), the state conditions of \( r_1 \) must be satisfied in \( r_2 \) and the message exchange conditions of \( r_1 \) must be satisfied prior to the message exchange conditions of \( r_2 \) (i.e., the message history of \( r_1 \) must be a subset of the message history of \( r_2 \)). Thus "\( \{x\} \) must be a subset of \( \{t\} \)" yields:

**R1.1:** \( C_x \) must be satisfied in \( C_t \). Thus, in the system’s real state, \( A_i \) must have an authenticated channel to \( A_j \), and \( A_i \) must send \( B_i B_k Q \) to \( A_j \) only after receiving a message containing \( B_k Q \) from \( A_k \).

**R1.2:** \( M_x \) must be satisfied prior to \( M_t \). Thus, \( A_i \) must receive \( B_j B_k Q \) from \( A_j \) before sending \( B_i B_k Q \) to \( A_j \).

Since we are only interested in constraints that affect real system states (rather than possible states), and since both \( y \) and \( u \) are possible states, we will not elaborate on the result that \( \{y\} \) must be a subset of \( \{u\} \).

**Constraint R2:** \( \rho_{j=r} \) must be a subset of \( \rho_{j=t} \). Thus, \( \{(y,z)\} \) must be a subset of \( \{(s,t)\} \). Therefore, \( \{y\} \) must be a subset of \( \{s\} \), and \( \{z\} \) must be a subset of \( \{t\} \). "\( \{y\} \) must be a subset of \( \{s\} \)" yields:

**R2.1:** \( C_y \) must be satisfied in \( C_s \). In \( A_j \)'s real state, \( A_j \) must have an authenticated channel to \( A_i \), and \( A_j \) must send \( B_j B_k Q \) to \( A_i \) only after receiving a message \( B_k Q \) from \( A_k \).

**R2.2:** \( M_y \) must be satisfied prior to \( M_s \). Thus, \( A_j \) must send \( B_j B_k Q \) to \( A_i \) before receiving \( B_i B_k Q \) from \( A_i \).

Notice that from R1 we have "\( \{y\} \) must be a subset of \( \{u\} \)"; and from R2 we have "\( \{y\} \) must be a subset of \( \{s\} \)". Thus, the state conditions of \( u \) and \( s \) must be identical:

**R2.3:** \( u \) must be a real state, and hence, in \( A_k \)'s real state, \( A_k \) must have an authenticated message channel to \( A_i \), and \( A_k \) must send \( Q \) to \( A_i \) only if it believes \( Q \).

R2 also yields that \( \{z\} \) must be a subset of \( \{t\} \). Thus:

**R2.4:** \( C_z \) must be satisfied in \( C_t \). In the system's real state, by R1.1, \( C_t \) is satisfied, and hence \( C_z \) must be satisfied. Hence in the system's real state, \( A_k \) must have an authenticated message channel to \( A_j \), and \( A_k \) must send \( B_k Q \) to \( A_j \) only if it believes \( Q \).

**R2.5:** \( M_z \) must be satisfied prior to \( M_t \). But \( M_t \) may become satisfied at any instant after \( M_u \). Hence, \( M_z \) must be satisfied prior to \( M_u \). Thus, \( A_k \) must send \( B_k Q \) to \( A_j \) before sending it to \( A_i \).

This concludes the derivation of the constraints that are necessary and sufficient for \( B_j B_i B_k Q \Rightarrow B_j B_k Q \) to hold in the distributed system. Notice that the constraints we have derived are in fact the protocols that agents \( A_i, A_j \) and \( A_k \) must follow if the trust is to be satisfied in the system. Since in this procedure we map a trust to its semantic interpretation, we obtain the protocols that are necessary and sufficient for the trust to hold. Any WFF can be mapped to its semantic interpretation, and hence semantic interpretation can be carried out for any trust resulting in the protocols necessary and sufficient for the trust to hold. Since we have not made any particular assumptions with respect to the nature of the system, its models, or the trusts, this methodology of obtaining protocols necessary and sufficient for a given trust to hold is general in its applicability to trusts, systems, and their models. Even for such a simple trust specification which involves just three agents, the constraints that we obtained earlier using a casual interpretation form a small subset of those we have obtained using a formal interpretation. Thus, formalism is essential, and mere intuition is not dependable.
It can be shown that, if any of the constraints R1.1, R1.2, R2.1, R2.2, R2.3, R2.4 and R2.5 are not satisfied, this may result in a violation of the trust. Let us illustrate this by an example. Suppose part of constraint R2.1, namely,

"A_j sends B_j B_k Q to A_i only after receiving a message containing B_j Q from A_k ",

is not satisfied in a system. The trust specification "\( \forall Q, B_j B_i B_k Q \Rightarrow B_i B_j B_k Q \)" is falsified at the end of the following sequence of steps (see Figure 2.4):

Step 1 (Figure 2.4(a)): \( B_k Q \) is true. \( A_j \) sends "\( B_j B_k \)" to \( A_i \), and this creates a belief "\( B_i B_j B_k \)" in \( A_i \).

---

**Figure 2.4:** A scenario which shows that the trust can be falsified if constraint R2.1 is not satisfied. (a) The sending of \( B_j B_k \) by \( A_j \) to \( A_i \) creates a belief \( B_i B_j B_k \) in \( A_i \). (b) The sending of \( B_k Q \) by \( A_k \) to \( A_i \) and \( A_j \) creates beliefs \( B_i B_k \) and \( B_j B_k \). (c) The sending of \( B_j B_k \) by \( A_j \) to \( A_i \) is rejected by \( A_i \) as it is inconsistent with the message \( A_i \) received from \( A_j \) in (a). (d) The sending of \( B_i B_k \) by \( A_i \) to \( A_j \) creates a belief \( B_j B_i B_k \). Since \( B_i B_j B_k \) is not true, the trust is falsified at this juncture.
Step 2 (Figure 2.4(b)): $A_k$ sends its belief "$B_k \neg Q"$ first to $A_j$ and then to $A_i$. At this point, both $A_j$ and $A_i$ have belief "$B_k \neg Q"$.

Step 3 (Figure 2.4(c)): $A_j$ sends "$B_j B_k \neg Q"$ to $A_i$. Since $A_i$ has earlier received "$B_j B_k \neg Q"$ from $A_j$ in step 1, and since "$B_k \neg Q"$ is inconsistent with "$B_k Q"$, $A_i$ rejects the message from $A_j$ containing "$B_j B_k \neg Q"$. Notice that the trust $B_j B_i B_k \neg Q \Rightarrow B_i B_j B_k \neg Q$ is not violated. This is because the trust is violated only when the antecedent of the trust $B_j B_i B_k \neg Q$ is true but the consequent $B_i B_j B_k \neg Q$ is not true. However, at the end of this step the antecedent is not true.

Step 4 (Figure 2.4(d)): $A_i$ sends its belief "$B_i B_k \neg Q"$ to $A_j$. This creates a belief "$B_j B_i B_k \neg Q"$ in $A_j$. Thus the antecedent of the trust $\forall Q, B_j B_i B_k \neg Q \Rightarrow B_i B_j B_k \neg Q$ is true. But the consequent $B_i B_j B_k \neg Q$ is not true (recall that $B_i B_j B_k \neg Q$ is true at the end of step 1), and hence the trust "$B_j B_i B_k \neg Q \Rightarrow B_i B_j B_k \neg Q"$ becomes false.

2.8. Conclusion

We have developed an axiomatic theory of trust in distributed systems. The theory of trust is based on modal logics of belief. Any well formed formula assumed to be valid in addition to the axioms of the logic is considered as a trust specification. This gives us much power and generality in expressing trust relationships. We have given a formal method for synthesizing protocols which are necessary and sufficient for implementing a given trust specification in a distributed system. In comparison, even for some simple trust specifications, informal methods do not yield all the required protocols. In Kripke's theory, any well formed formula can be given a semantic interpretation, and since our method of synthesizing protocols that are necessary and sufficient for a trust to hold is based on giving a semantic interpretation to the trust, our method is general in its applicability to trusts, systems, and their models.
CHAPTER 3

ANALYSIS

Trust arises primarily in establishing channels for secure communication. This chapter analyzes the trust properties of various channel establishment mechanisms. We define a channel precisely and show that the only way to establish a new channel is by composing a sequence of existing adjacent channels. Channel composition mechanisms are commonly based on either public key encryption (PKE) or single key encryption (SKE). We present methods for reasoning about the trust characteristics of PKE- and SKE-based channel composition mechanisms. PKE-based channel composition requires 3-agent trust predicates called authenticity trusts. The trust requirements of SKE-based channel composition are much more extensive than those of PKE-based channel composition. The differences in trust properties of PKE and SKE-based channel compositions are used to compare these two methods, and derive several advantageous properties of the former over the latter.

3.1. Introduction

It was observed in Chapter 1 that in any system, given a set of existing channels, the only way to establish new channels is by composing a sequence of adjacent existing channels. Channel composition mechanisms may require the satisfaction of some trust relationships. Having given a precise meaning to the notion of trust in the previous chapter, we analyze the trusts inherent in various channel composition mechanisms in this chapter. The next chapter discusses the synthesis of distributed systems so as to satisfy a given set of trust relationships. The analyses presented in this chapter must precede the design methodology of the next chapter because, to be able to make use of a mechanism in a distributed system that is designed to satisfy a given set of trusts, one has to know the trust properties of the mechanism. These analyses provide insight into the basic structure and the limitations of mechanisms with regard to their trust requirements.

In Section 3.2 we define a channel precisely, and prove that the only way to establish new channels is by composing existing channels. In order to analyze trusts formally in various channel composition mechanisms, the fundamental actions in the mechanisms must be encoded in the language of the logic of trust. Section 3.3 introduces the atomic propositions that encode these fundamental actions. Channels are based on encryption, and there are two commonly used encryption techniques, namely, public key encryption (PKE) and single key encryption (SKE). Sections 3.4-3.7 analyze the trust relationships required in PKE- and SKE-based channel composition mechanisms, with Sections 3.4 and 3.5 considering the composition of two channels, and Sections 3.6 and 3.7 considering the composition of more than two channels. Making use of the results of these formal analyses, Section 3.8 discusses the advantages of PKE-based mechanisms over SKE-based mechanisms for channel composition. Finally, Section 3.9 concludes the chapter.

3.2. Channels

In order to analyze trusts in channel composition mechanisms, we first have to define a channel precisely.
3.2.1. Definition of Channel

A channel $(A_i, A_k)$ is said to exist between two agents $A_i$ and $A_k$ if and only if the following two conditions are satisfied (see Figure 3.1):

1. **Authenticity Condition**: $A_i$ can authenticate messages coming from $A_k$, i.e., $A_i$ can determine whether a message it received has been really sent by $A_k$, and

2. **Privacy Condition**: $A_i$ can send a secret message to $A_k$, $A_i$ knows the identities of agents other than $A_k$ that can decrypt the secret message (because these agents might possess the key with which the secret message is encrypted), and the decryption of the secret message by those agents is acceptable to $A_i$.

Algorithms for ensuring both these conditions are executed at $A_i$ using information such as encryption keys associated with $A_k$. The privacy condition is motivated by the observation that agents other than $A_i$ and $A_k$ may have been involved in establishing channel $(A_i, A_k)$, in which case those agents may possess the channel encryption key and hence have the capability to decrypt secret messages on the channel. The privacy condition requires that $A_i$ precisely know the identities of those agents. As we shall see in later sections, the agents that may possess the channel encryption key are those that are in the path between $A_i$ and $A_k$ in the system's name space, and with each such agent, $A_i$ has to have a trust relationship that guarantees that the agent will not compromise the security of channel $(A_i, A_k)$. In Chapter 4, we shall see how $A_i$ can control the set of such agents. Notice that channel $(A_i, A_k)$ involves messages in either direction and does not imply that messages can only flow from $A_i$ to $A_k$. Channel $(A_i, A_k)$ and channel $(A_k, A_i)$ together form a **bidirectional channel** between $A_i$ and $A_k$.

3.2.2. Channel Establishment

We now show that in any system, given a set of existing channels, the only way to establish a new channel is by composing a sequence of adjacent existing channels. Even though this result seems very intuitive, it is a very powerful result. As we shall see in Sections 3.4-3.7, this result greatly simplifies the analysis of trust properties of channel establishment mechanisms. A channel established by composing other channels is called a **dependent channel**. On the

![Figure 3.1: Channel$(A_i, A_k)$](image-url)
other hand, an *independent channel* does not use any other channels for its establishment. The system provides independent channels at the time of system configuration. Independent channels are established using mechanisms external to the system, such as courier-exchanges between agents.

**Theorem 3.1 (Channel Composition Theorem):** Suppose that, in a system consisting of agents $A_i, A_j^1, A_j^2, ..., A_j^n, A_k$, the only existing channels are $\text{channel}(A_i, A_j^1), \text{channel}(A_j^1, A_j^2), \ldots, \text{channel}(A_j^n, A_k)$, which form a path between $A_i$ and $A_k$. Any mechanism that establishes $\text{channel}(A_i, A_k)$ necessarily involves messages on all the existing channels, and the mechanism must necessarily consist of a succession of compositions of two adjacent channels.

**Proof:** The proof is by induction on the number of agents $A_j^1, A_j^2, ..., A_j^n$ in the path between $A_i$ and $A_k$.

**Base Case:** Suppose $n = 0$. The result is trivially true.

**Induction Step:** Assuming that the theorem holds for $n < m$, we show that it holds for $n = m$. Let the sequence of channels form the only path between $A_i$ and $A_k$ (see Figure 3.2). The establishment of $\text{channel}(A_i, A_k)$ requires that $A_k$'s encryption key be received by $A_i$. Since $A_i$ can receive messages only on its existing channels, and since its only existing channel is $\text{channel}(A_i, A_j^1)$, for $\text{channel}(A_i, A_k)$ to be established, $A_i$ must receive $A_k$'s key in a message from $A_j^1$ on $\text{channel}(A_i, A_j^1)$. Thus, a message on $\text{channel}(A_i, A_j^1)$ is necessary, and $A_j^1$ must have possessed $A_k$'s key prior to sending it to $A_i$. But $A_j^1$ possessing $A_k$'s key implies that $\text{channel}(A_j^1, A_k)$ exists. Consequently, $\text{channel}(A_j^1, A_k)$ must have been established prior to $A_j^1$'s sending of $A_k$'s key to $A_i$. Hence, $\text{channel}(A_i, A_k)$ was composed from two adjacent channels, $\text{channel}(A_i, A_j^1)$ and $\text{channel}(A_j^1, A_k)$, and the composition involved a message on $\text{channel}(A_i, A_j^1)$. However, from the induction hypothesis, $\text{channel}(A_j^1, A_k)$ involved messages

![Figure 3.2: The Channel Composition Theorem](image-url)
on all the channels in the path between $A_j^1$ and $A_k$, and hence channel($A_j^1$, $A_k$) was the result of a succession of adjacent two-channel compositions between $A_j^1$ and $A_k$. Thus, any mechanism that establishes channel($A_i$, $A_t$) must necessarily involve messages on all the existing channels, and must necessarily consist of a succession of adjacent two-channel compositions. This completes the proof of the Channel Composition Theorem.

Channel establishment, which is a sequence of adjacent channel compositions, requires trust relationships. The trust relationships required are such that, when they hold in the system, the authenticity and privacy conditions of the newly established channel are satisfied, and, when they do not hold, these conditions are not satisfied.

In the next few sections, we analyze the trust requirements of various channel composition mechanisms. The trust requirements depend on the algorithms used for ensuring the two channel conditions. We will consider the two commonly used kinds of these algorithms, namely, public key encryption algorithms (PKE) [DiH76, RSA78] and single key encryption algorithms (SKE) [NBS77].

In order to do a formal analysis of the various mechanisms, we have to encode the fundamental actions of the mechanisms into the language of the logic of trust. This is accomplished in the next section.

3.3. Atomic Propositions

Fundamental actions at a given level of abstraction are encoded in the language of the logic of trust by atomic propositions. Atomic propositions are so called because they are the most basic well formed formulas in the language of the logic, and an atomic proposition cannot be described in terms of any other atomic propositions. Thus, the set of atomic propositions is not unique to a system, but depends on the level of abstraction at which we are analyzing the system. For our analysis of trust relationships in PKE- and SKE-based channel composition mechanisms, the fundamental actions are key-generation, message-sending and message-reception. These are abstracted by the following atomic propositions:

(1) $\text{owner}(\text{key}_x, A_i)$ (Ownership Proposition): PKE and SKE algorithms make use of encryption keys belonging to agents. Let a key denote an encryption key from a finite key space $KEY$. An ownership proposition encodes the generation of a key. $\text{Owner}(\text{key}_x, A_i)$ returns true if and only if agent $A_i$ generated the encryption key $key_x$. Given a notion of feasible computation, it is assumed that an agent can generate keys (perhaps using random number generators) that cannot be generated using a feasible computation by any other agent [Den82]. Note that the generator of the key is always its owner. Thus, if a key server generates a key and hands it over to an agent, the key server retains the ownership of the key. In Section 3.4, we shall describe using beliefs the relationship between an agent and a key that the agent receives from a key server.

(2) $\text{send}(A_i, msg_x)$: This atomic proposition abstracts the sending of a message $msg_x$ by an agent $A_i$. Suppose $msg_x$ can be derived using a feasible computation from another message $msg_y$. From the viewpoint of security, sending $msg_y$ on a channel has also the effect of sending $msg_x$ on the channel. Thus $\text{send}(A_i, msg_x)$ returns true if and only if $A_i$ sends a message $msg_y$ such that at the time of sending, $A_i$ can derive $msg_x$ from $msg_y$ using a feasible computation. $Msg_x$ can be identical to $msg_y$. Notice that a second agent $A_j$ may be able to derive $msg_z$ from $msg_y$; if $A_i$ cannot derive $msg_z$ from $msg_y$, then the
proposition is not true. In practice, \( \text{msg}_x \) is a message that can be obtained by decrypting \( \text{msg}_y \) using a key that \( A_i \) possesses.

(3) \text{receive}(A_i, \text{msg}_x): \) This atomic proposition abstracts the receiving of a message \( \text{msg}_x \) by an agent \( A_i \). Suppose \( \text{msg}_x \) can be derived from another message \( \text{msg}_y \) using a feasible computation. From the viewpoint of security, receiving \( \text{msg}_y \) on a channel has also the effect of receiving \( \text{msg}_x \) on the channel. Thus \text{receive}(A_i, \text{msg}_x) \) returns true if and only if \( A_i \) receives a message \( \text{msg}_y \) such that, at the time of receiving, \( A_i \) can derive \( \text{msg}_x \) from \( \text{msg}_y \) using a feasible computation.

We are now fully equipped to proceed with the analysis of trust relationships in channel composition mechanisms. For ease of understanding, the analysis considers the following cases separately (in the order of increasing complexity):

1. Composition of two independent channels using PKE,
2. Composition of two independent channels using SKE,
3. Composition of two dependent channels using PKE or SKE, and
4. Composition of more than two independent/dependent channels using PKE or SKE.

In the sequel, the composition of two independent channels will be termed \textit{independent channel composition} and that of two dependent channels will be termed \textit{dependent channel composition}.

3.4. Composition of Two PKE-based Independent Channels

Suppose channel\((A_i, A_k)\) is to be obtained from channel\((A_i, A_j)\) and channel\((A_j, A_k)\). In the PKE scheme [DiH76, RSA78], each agent has a two keys, namely, a public key and a private key that form a pair. A message sent encrypted with a public key can only be received by decrypting it with the corresponding private key, and vice versa. To compose channel\((A_i, A_k)\) from channel\((A_i, A_j)\) and channel\((A_j, A_k)\), \( A_k \) selects a (public key, private key) pair and sends the public key to \( A_j \) on channel\((A_j, A_k)\) (see Figure 3.3). In practice, \( A_j \) may be a name server that stores \( A_k \)'s public key. When \( A_i \) sends a request for \( A_k \)'s public key to \( A_j \), \( A_j \) forwards \( A_k \)'s public key to \( A_i \) on channel\((A_i, A_j)\). For channel\((A_i, A_k)\) to be established, it is necessary and sufficient for \( A_i \) to know \( A_k \)'s public key.

3.4.1. Trusts in PKE-based Channel Composition

Let us look at the PKE-based channel composition mechanism more formally. When \( A_k \) selects a public-key private-key pair \( (\text{key}^\text{pub}_x, \text{key}^\text{priv}_x) \), it adds a belief \( "B_k\text{owner}(\text{key}^\text{pub}_x, A_k)" \). Since the public key uniquely determines the private key, a second belief claiming ownership of the private key is redundant. Note that it is not necessary for \( A_k \) to be the owner of the key pair, it suffices if \( A_k \) believes to be the owner. Thus, an agent that does not have the ability to generate keys can obtain a key pair from some other agent and at its risk, use that key pair as its own.

\( A_k \) then sends its belief, \( B_k\text{owner}(\text{key}^\text{pub}_x, A_k) \) to \( A_j \), and this creates a belief, \( B_jB_k\text{owner}(\text{key}^\text{pub}_x, A_k) \) in \( A_j \). When \( A_j \) receives a request for \( A_k \)'s public key from \( A_i \), \( A_j \) replies with its belief, \( B_jB_k\text{owner}(\text{key}^\text{pub}_x, A_k) \). This reply from \( A_j \) to \( A_i \) creates a belief, \( B_iB_jB_k\text{owner}(\text{key}^\text{pub}_x, A_k) \) in \( A_i \).

However, the belief \( B_jB_k\text{owner}(\text{key}^\text{pub}_x, A_k) \) is not sufficient for channel\((A_i, A_k)\) to be established at \( A_i \). For channel\((A_i, A_k)\) to be established at \( A_i \), \( A_i \) must prove that \( A_k \) believes \( \text{key}^\text{pub}_x \) to be its public key, i.e., \( A_i \) must prove that \( B_k\text{owner}(\text{key}^\text{pub}_x, A_k) \) is true. \( A_i \) is not
required to prove owner (key\textsuperscript{pub}, A\textsubscript{k}) because it is permissible for A\textsubscript{i} to obtain a key pair from a key server and to use the key pair as its own. In such a case the key server retains the ownership of the key pair, whereas A\textsubscript{i} adds a belief B\textsubscript{k} owner (key\textsuperscript{pub}, A\textsubscript{k}). The required trust relationship must be such that A\textsubscript{i} can infer B\textsubscript{k} owner (key\textsuperscript{pub}, A\textsubscript{k}) from its belief B\textsubscript{i} B\textsubscript{j} B\textsubscript{k} owner (key\textsuperscript{pub}, A\textsubscript{k}). Rewriting this requirement as a formula, we obtain the following trust for PKE-based channel composition:

\[ T\textsubscript{A}(A\textsubscript{i}, A\textsubscript{j}, A\textsubscript{k}) \text{ (Authenticity Trust): } \forall \text{ key\textsuperscript{pub} in the public-key space PKEY, } B\textsubscript{i} B\textsubscript{j} B\textsubscript{k} \text{ owner(key\textsuperscript{pub}, A\textsubscript{k}) } \Rightarrow B\textsubscript{k} \text{ owner(key\textsuperscript{pub}, A\textsubscript{k}).} \]

The trust is called \textit{authenticity trust} because its validity requires A\textsubscript{j} to forward correctly to A\textsubscript{i} the public key that A\textsubscript{j} received from A\textsubscript{k}. In other words, A\textsubscript{j} has to forward \textit{authentic} information about A\textsubscript{k} to A\textsubscript{i}.

Interpreting \( T\textsubscript{A}(A\textsubscript{i}, A\textsubscript{j}, A\textsubscript{k}) \) as "A\textsubscript{i} trusts A\textsubscript{j} for A\textsubscript{k}" gives a connotation that A\textsubscript{i} is inevitably the only loser if \( T\textsubscript{A}(A\textsubscript{i}, A\textsubscript{j}, A\textsubscript{k}) \) is not true. We will now show that falsity of \( T\textsubscript{A}(A\textsubscript{i}, A\textsubscript{j}, A\textsubscript{k}) \) may be disadvantageous to either A\textsubscript{i} or A\textsubscript{k}, and hence \( T\textsubscript{A}(A\textsubscript{i}, A\textsubscript{j}, A\textsubscript{k}) \) should be interpreted as an agreement involving A\textsubscript{i}, A\textsubscript{j} and A\textsubscript{k} in which A\textsubscript{j} has agreed to correctly forward A\textsubscript{k}’s key to A\textsubscript{i}.

If we use a client-server model of a distributed system, there are two cases of interest with respect to the interaction between A\textsubscript{i} and A\textsubscript{k} (see Figure 3.4):

1. A\textsubscript{i} is a client and A\textsubscript{k} is a server. As an example, let A\textsubscript{k} be a time server. Suppose A\textsubscript{i} sends a time-of-day request to A\textsubscript{k}, and A\textsubscript{i} receives a reply purporting to be from A\textsubscript{k}. Falsity of \( T\textsubscript{A}(A\textsubscript{i}, A\textsubscript{j}, A\textsubscript{k}) \) can result in A\textsubscript{i} accepting a reply from another agent A\textsubscript{m} which in collusion with A\textsubscript{j} is masquerading as A\textsubscript{k}. Thus, A\textsubscript{i} accepts an incorrect time-of-day reply, and A\textsubscript{i} is the loser for the falsity of \( T\textsubscript{A}(A\textsubscript{i}, A\textsubscript{j}, A\textsubscript{k}) \).
(2) $A_i$ is a server and $A_k$ is a client. As an example, let $A_i$ be a file server. Falsity of $T_A(A_i, A_j, A_k)$ can result in an agent $A_m$ colluding with $A_j$ to masquerade as $A_k$ in sending a file-write request to the file server $A_i$. $A_i$ accepts an incorrect write to $A_k$'s file, and thus $A_k$ is the loser for the falsity of $T_A(A_i, A_j, A_k)$. ($A_i$ might be a loser too if it has to pay damages.)

In summary, falsity of $T_A(A_i, A_j, A_k)$ can be disadvantageous to either $A_i$ or $A_k$ or both, and hence $T_A(A_i, A_j, A_k)$ should be interpreted as an accord involving $A_i$, $A_j$ and $A_k$, in which $A_j$ has agreed to forward correctly $A_k$'s key to $A_i$.

3.4.2. Necessity and Sufficiency of Authenticity Trust

Trust requirements depend on the channel composition mechanism under consideration. A set of trust relationships is sufficient w.r.t. a channel composition mechanism if, by using the trusts as assumptions, the two channel conditions presented in Section 3.2 can be shown to be satisfied for the newly composed channel. A set of trusts is necessary w.r.t. a channel composition mechanism if, for each assignment to the variables in the trusts that makes at least one of the trusts false, the same assignment also makes at least one of the two channel conditions not satisfied. If a set of trusts is necessary and sufficient w.r.t. a channel composition mechanism, the trusts in the set exactly encode the assumptions inherent in the channel composition mechanism. The following theorem proves that the authenticity trust is necessary and sufficient in PKE-based channel composition mechanism.

**Theorem 3.2:** The authenticity trust is necessary and sufficient w.r.t. PKE-based channel composition mechanism.

---

Figure 3.4: Interpreting authenticity trust: Illustration of how either $A_i$ or $A_k$ may lose owing to falsity of $T_A(A_i, A_j, A_k)$. $A_m$ in collaboration with $A_j$, masquerades as $A_k$ to $A_i$: (a) $msg-1$ is a time request, $msg-2$ is a time reply from $A_m$ $msg-3$ is a request for $A_k$'s public key, and $msg-4$ is a reply containing $A_m$'s public key. (b) $msg-1$ is a write-request to $A_k$'s file, $msg-2$ is a request for $A_k$'s public key, and $msg-3$ is a reply containing $A_m$'s public key.
Proof: Suppose channel($A_i$, $A_j$) and channel($A_j$, $A_k$) are composed to form channel($A_i$, $A_k$) using the PKE-based channel composition mechanism. We first show that the authenticity trust is sufficient, and then show that it is necessary.

The proof that authenticity trust is sufficient is straightforward. For channel($A_i$, $A_k$) to have been established, $A_i$ must have received $key^\text{pub}_x$ on its existing channel($A_i$, $A_j$), hence $B_iB_jB_k\owner(key^\text{pub}_x, A_k)$ is true. By the authenticity trust, $B_k\owner(key^\text{pub}_x, A_k)$ must be true. Thus, $A_i$ alone uses $key^\text{priv}_x$, and the properties of public key encryption ensure the satisfaction of both the channel conditions. Thus, the authenticity trust is sufficient for channel($A_i$, $A_k$) to be established.

Let "V", "\land", and "\lor" denote inclusive OR, AND, and complementation respectively. To show that the authenticity trust is necessary, notice that this trust can be written as:

$$B_k\owner(key^\text{pub}_x, A_k) \lor B_iB_jB_k\owner(key^\text{pub}_x, A_k).$$

(3.1)

The negation of formula (3.1) is:

$$\neg B_k\owner(key^\text{pub}_x, A_k) \land B_iB_jB_k\owner(key^\text{pub}_x, A_k).$$

(3.2)

The only variable in the above formula is $key^\text{pub}_x$. Suppose that, for some assignment to $key^\text{pub}_x$, formula (3.2) is satisfied, i.e., both $\neg B_k\owner(key^\text{pub}_x, A_k)$ and $B_iB_jB_k\owner(key^\text{pub}_x, A_k)$ are true. Since $B_iB_jB_k\owner(key^\text{pub}_x, A_k)$ is true, $A_i$ has received $key^\text{pub}_x$ during the creation by composition of channel($A_i$, $A_k$), and hence $A_i$ uses $key^\text{pub}_x$ as the key of channel($A_i$, $A_k$). Thus, when $A_i$ receives a message purporting to be from $A_k$, $A_i$ uses $key^\text{pub}_x$ for authenticating the received message. If the received message has been encrypted with $key^\text{priv}_x$, $A_i$ determines that the message sender is $A_k$. However, since $\neg B_k\owner(key^\text{pub}_x, A_k)$ is true, $A_k$ does not use $key^\text{priv}_x$ to encrypt messages, and hence the message was not sent by $A_k$. Thus, the authenticity condition of channel($A_i$, $A_k$) becomes false. Thus, if any assignment to the only variable in the authenticity trust makes the trust false, the same assignment can make the authenticity condition of channel($A_i$, $A_k$) false. This completes the proof that authenticity trust is necessary w.r.t. the PKE-based channel composition mechanism.

There is another method by which we can show that the authenticity trust exactly encodes the assumptions inherent in PKE-based channel composition mechanism, and that is by viewing the trust as a well-formed formula in the logic of belief and carrying out its formal semantic interpretation. Techniques developed in Section 2.7 are used to carry out the formal semantic interpretation. The following section illustrates the method.

3.4.3. Semantic Interpretation of the Authenticity Trust

Let the state of the system be $s$. Consider the antecedent of the authenticity trust, $B_iB_jB_k\owner(key^\text{pub}_x, A_k)$. The semantic interpretation of this belief is that there is a possible state $t$ such that $(s, t) \in \rho_i$, and the following conditions are satisfied in $t$ (see Figure 3.5):

- **(A1)**: The state of $A_i$ is the same as that in $s$.

\footnote{$\rho_i$ is $A_i$'s possibility relation. The method of semantically interpreting a belief was described in Section 2.7.}
(A2): there is an authenticated channel from $A_j$ to $A_i$,

(A3): $A_j$ has sent $B_j B_k owner(key_{x}^{pub}, A_k)$ to $A_i$, and

(A4): $B_j B_k owner(key_{x}^{pub}, A_k)$ is true, i.e., $A_j$ has received "$B_k owner(key_{x}^{pub}, A_k)$" from $A_k$.

Now consider state $t$. By condition A4, $B_j B_k owner(key_{x}^{pub}, A_k)$ is true in $t$, and interpreting this belief yields that there is a possible state $u$ such that $(t, u) \in \rho_j$, and the following conditions are satisfied in $u$:

(A5): The state of $A_j$ is same as that in $t$, hence conditions A2, A3 and A4 are satisfied in $u$,

(A6): there is an authenticated channel from $A_k$ to $A_j$,

(A7): $A_k$ has sent $B_k owner(key_{x}^{pub}, A_t)$ to $A_j$, and

(A8): $B_k owner(key_{x}^{pub}, A_k)$ is true.

Now consider the consequent of the authenticity trust, $B_k owner(key_{x}^{pub}, A_k)$. The trust requires that in any state in which the antecedent is true, the consequent also be true. Since the antecedent is true in state $s$, we need also the consequent to be true in state $s$. Comparing condition A8 and the consequent, we obtain that the consequent is true in $u$. Thus, for the consequent to be true in $s$, it must be the case that $s = u$. However, in $u$, conditions A5, A6, A7 and A8 are satisfied. Thus, in the real state $s$, conditions A5, A6, A7 and A8 must be satisfied. But A5 requires that A2, A3 and A4 be satisfied. Thus, in real state $s$, A2, A3, A4, A6, A7 and A8 must be satisfied. Notice that these conditions are exactly the assumptions on which the PKE-based channel composition mechanism is founded.

![Figure 3.5: Semantic interpretation of authenticity trust](image-url)
A2: Existence of channel($A_i, A_j$),
A3 and A4: $A_j$ correctly forwards $A_k$'s key,
A6: Existence of channel($A_j, A_k$), and
A7 and A8: $A_k$ selects a public-key private-key pair and sends the public key to $A_j$.

This concludes the verification that the authenticity trust exactly encodes the assumptions inherent in the PKE-based channel composition mechanism, and hence the authenticity trust is necessary and sufficient w.r.t. the PKE-based channel composition mechanism.

3.5. Composition of Two SKE-based Independent Channels

Suppose that channel($A_i, A_k$) is to be composed from channel($A_i, A_j$) and channel($A_j, A_k$). In the SKE scheme [NBS77], there is one key belonging to each agent, and the key is referred to as the agent's single key. A message sent encrypted with a single key can only be received by decrypting it with the very same single key, and vice versa. To obtain channel($A_i, A_k$) from the composition of channel($A_i, A_j$) and channel($A_j, A_k$), $A_k$ selects a single key and sends the single key to $A_j$ in a message $msg_{kj}$ on channel($A_j, A_k$) (see Figure 3.6). When $A_i$ sends a request for $A_k$'s key to $A_j$, $A_j$ forwards $A_k$'s single key to $A_i$ in a message $msg_{ji}$ on channel($A_i, A_j$). It is necessary for $A_i$ to know $A_k$'s single key for channel($A_i, A_k$) to be established. Thus, the authenticity trust is necessary as in the PKE scheme.

There is a major difference between the PKE and SKE schemes with regard to trust requirements. In the PKE scheme, an agent such as $A_j$, even though it has obtained $A_k$'s public key during channel composition, cannot masquerade as $A_k$ or decrypt secret messages on channel($A_i, A_k$). But in the SKE scheme, knowing $A_k$'s single key enables $A_j$ to masquerade as

\[\text{Figure 3.6: SKE-based channel composition mechanism}\]
As and to decrypt secret messages by eavesdropping on channel\((A_i, A_k)\). Thus, authenticity and privacy conditions of channel\((A_i, A_k)\) are not satisfied. Hence, the authenticity trust is not sufficient w.r.t. the SKE-based channel composition mechanism.

To determine the remaining trust requirements of the SKE scheme, notice that the definition of channel\((A_i, A_k)\) requires \(A_i\) to know the set of agents other than \(A_i\) and \(A_k\) that can receive secret messages sent on channel\((A_i, A_k)\). The set of agents that can decrypt secret messages on channel\((A_i, A_k)\) are those that have obtained \(A_k\)'s single key. During the channel establishment process, an agent may have obtained \(A_k\)'s single key either when the key traversed the path from \(A_k\) to \(A_j\) in message msg\(_{kj}\) or when the key traversed the path from \(A_j\) to \(A_i\) in message msg\(_{ji}\).

Consider an agent \(A_m\) that may have obtained the key by receiving msg\(_{kj}\). Notice that, by definition, channel\((A_k, A_j)\) allows \(A_k\) to send a secret message to \(A_j\) so that \(A_k\) knows the identity of agents who can decrypt the secret message. Thus the identity of agent \(A_m\) is known to \(A_k\) if \(A_k\) sends msg\(_{kj}\) as a secret message on channel\((A_k, A_j)\).

Now consider an agent \(A_m\) that may have obtained the key by receiving msg\(_{ji}\). By definition, channel\((A_j, A_i)\) allows \(A_j\) to send a secret message to \(A_i\) so that \(A_j\) knows the identity of agents who can decrypt the secret message. Thus, the identity of agent \(A_m\) is known to \(A_j\) if \(A_j\) sends msg\(_{ji}\) as a secret message on channel\((A_j, A_i)\).

To ensure that \(A_i\) knows the identity of \(A_m\) and \(A_m\), the following mechanism can be used: \(A_k\) sends the string "\(A_m, B_k\)owner(key\(_x\), A\(_k\))" to \(A_j\) in a secret message msg\(_{kj}\) on channel\((A_k, A_j)\). When \(A_j\) receives msg\(_{kj}\), it decrypts msg\(_{kj}\) using the key of channel\((A_k, A_j)\) and authenticates msg\(_{kj}\) using the key of channel\((A_j, A_k)\). At this juncture, \(A_j\) knows the identities of both \(A_m\) and \(A_m\). \(A_j\) sends the string "\(A_m, A_m, B_jB_k\)owner(key\(_x\), A\(_k\))" to \(A_i\) in a secret message msg\(_{ji}\) on channel\((A_j, A_i)\). When \(A_i\) receives msg\(_{ji}\), it decrypts msg\(_{ji}\) using the key of channel\((A_j, A_i)\) and authenticates msg\(_{ji}\) using the key of channel\((A_i, A_j)\).

The assumptions in the above mechanism are formally captured in the following trust definition:

\[
T_F(A_i, A_j, A_k) \text{(Forwarding Trust): } \forall \text{ key}_x, A_m, A_m, B_jB_k \text{owner(key}_x, A_k) \Rightarrow (\text{send}(A_j, \text{key}_x) \land \text{receive}(A_m, \text{key}_x)) \Rightarrow B_jB_k \text{receive}(A_m, \text{key}_x) \land (\text{send}(A_k, \text{key}_x) \land \text{receive}(A_m, \text{key}_x)) \Rightarrow B_jB_k \text{receive}(A_m, \text{key}_x) \land (B_jB_k \text{receive}(A_m, \text{key}_x) = B_jB_k \text{receive}(A_m, \text{key}_x)).
\]

The antecedent in the above definition of forwarding trust encodes that \(A_i\) has received \(A_k\)'s key through \(A_j\).

The first term of the consequent, "\(\text{send}(A_j, \text{key}_x) \land \text{receive}(A_m, \text{key}_x) = B_jB_k \text{receive}(A_m, \text{key}_x)\)" encodes that, if \(A_m\) is able to obtain \(A_k\)'s key by decrypting msg\(_{ji}\), \(A_j\) informs \(A_i\) of \(A_m\)'s identity.

The first factor in the second term of the consequent, "\(\text{send}(A_k, \text{key}_x) \land \text{receive}(A_m, \text{key}_x) \Rightarrow B_jB_k \text{receive}(A_m, \text{key}_x)\)" encodes that, if \(A_m\) is able to obtain \(A_k\)'s key by

---

2 Notice that, even if channel\((A_i, A_k)\) does not physically go through \(A_j\), \(A_i\) can still decrypt secret messages by eavesdropping on the channel.

3 The set of agents \(\{A_m\}\) may be empty, but that is only a special case.

4 Notice that channel\((A_k, A_j)\) is necessary for \(A_k\) to send a secret message to \(A_j\), whereas channel\((A_j, A_k)\) is necessary for \(A_j\) to authenticate a message sent by \(A_k\).
decrypting \(msg_{kj}\), \(A_k\) informs \(A_j\) of \(A_m\)'s identity. The second factor, \("B_jB_kreceive(A_m, key_x) \Rightarrow B_jB_kB_kreceive(A_m, key_x)"\) encodes that \(A_j\) correctly forwards the identity of \(A_m\) to \(A_i\).

To recapitulate, the forwarding trust allows \(A_i\) to determine the identities of all agents \(A_m\) that may have obtained \(key_x\) by decrypting \(msg_{kj}\) or \(msg_{ji}\). Three more assumptions are necessary for the conditions of channel\((A_i, A_k)\) to be satisfied. Firstly, \(A_m\) must not reveal \(key_x\) to any other agent \(A_j\). This assumption is captured by the Key Privacy Trust below. Secondly, for the authenticity condition of channel\((A_i, A_k)\) to be satisfied, \(A_m\) must not use \(key_x\) to masquerade on channel\((A_i, A_k)\). This assumption is captured by the Trust against Masquerading defined below. Lastly, for the privacy condition of channel\((A_i, A_k)\) to be satisfied, \(A_m\) must not decrypt and reveal a secret message sent on channel\((A_i, A_k)\). This final assumption is captured by the Message Privacy Trust defined below.

\[\text{T}_{\text{KP}}(A_i, A_m, A_k)\ (\text{Key Privacy Trust}) : \forall \ key_x, B_jB_kowner(key_x, A_k) \Rightarrow \text{send}(A_m, key_x).\]

\[\text{T}_{\text{AMasq}}(A_i, A_m, A_k)\ (\text{Trust against Masquerading}) : \forall \ msg_x, (\text{receive}(A_i, msg_x) \land B_i \text{send}(A_k, msg_x)) \Rightarrow \text{send}(A_m, msg_x).\]

\[\text{T}_{\text{MP}}(A_i, A_m, A_k)\ (\text{Message Privacy Trust}) : \forall \ msg_x, (\text{send}(A_i, msg_x) \land B_i \text{receive}(A_k, msg_x) \land \text{receive}(A_m, msg_x)) \Rightarrow \text{send}(A_m, msg_x).\]

These three trusts are required of any agent that possesses \(A_k\)'s key. The three trusts together form the Key User-Possessor Trust, denoted by \(\text{T}_{\text{KUP}}\).

The following theorem summarizes all the above results about SKE-based channel composition.

**Theorem 3.3:** Suppose that in a system there are agents \(A_i\) and \(A_k\), and channel\((A_i, A_k)\) is to be established using SKE-based composition. There must exist an agent \(A_j\) such that there are four channels, channel\((A_i, A_j)\), channel\((A_j, A_i)\), channel\((A_j, A_k)\) and channel\((A_k, A_j)\). The authenticity trust, the forwarding trust and the key user-possessor trust are necessary and sufficient w.r.t. SKE-based channel composition.

**Proof:** By the Channel Composition Theorem, there must exist an agent \(A_j\) such that there are channels \((A_i, A_j)\) and \((A_j, A_k)\). By the definition of forwarding trust, the forwarding trust requires privacy of messages from \(A_k\) to \(A_j\), and from \(A_j\) to \(A_i\). Thus, channel\((A_k, A_j)\) and channel\((A_j, A_i)\) are required for forwarding trust. We show below that the forwarding trust is necessary, and hence channel\((A_k, A_j)\) and channel\((A_j, A_i)\) are necessary.

We first show that the authenticity trust, the forwarding trust and the key user-possessor trust are sufficient, and then that they are necessary.

In SKE-based composition of channel\((A_i, A_k)\) from channels \((A_i, A_j)\) and \((A_j, A_k)\), when \(A_i\) uses \(key_x\) for channel\((A_i, A_k)\), it must have received \(key_x\) on its existing channel with \(A_j\), and hence \(B_jB_k\text{owner}(key_x, A_k)\) is true. Applying the authenticity trust, \(B_k\text{owner}(key_x, A_k)\) is true. Thus, \(A_k\) uses \(key_x\) to send messages to \(A_i\) and to receive secret messages from \(A_i\). We have to show that no other agent possessing \(key_x\) compromises the two conditions of channel\((A_i, A_k)\).

Consider a fourth agent that possesses \(key_x\). The agent must have obtained \(key_x\) either (1) directly from \(A_j\) or \(A_k\), or (2) indirectly from \(A_j\) or \(A_k\), i.e., through a sequence of agents, starting with some agent that received directly from \(A_j\) or \(A_k\). Since the forwarding trust \(T_F(A_i, A_j, A_k)\) is true, \(A_i\) knows the identities of all agents \(A_m\) that have obtained \(key_x\) directly from \(A_j\) or \(A_k\). Since the key privacy trust \(T_{\text{KP}}(A_i, A_m, A_k)\) is true, no such agent \(A_m\) can obtain \(key_x\) indirectly. For each \(A_m\), the trust against masquerading is true, hence no other agent except \(A_k\)
sends messages using $key_x$ and the authenticity condition of channel($A_i, A_k$) is satisfied. For each $A_m$, the message privacy trust is true, and hence no other agent except $A_k, A_j$ and $A_m$ obtain a secret message sent by $A_i$ using $key_x$. But $A_i$ knows the identities of $A_k, A_j$ and $A_m$, and hence the privacy condition of channel($A_i, A_k$) is satisfied. Thus, the trusts are sufficient for channel($A_i, A_k$) to be established.

The proof that the authenticity trust is necessary is the same as that in Theorem 3.2.

We now show that each assignment to the variables $key_x, msg_x, A_m_1, A_m_2$ and $A_m$ that falsifies either the forwarding trust or the key privacy trust or the trust against masquerading or the message privacy trust can also falsify one of the two conditions of channel($A_i, A_k$).

Consider the forwarding trust. Its negation can be written as:

$$(B_i B_j B_k owner (key_x, A_k) \land send (A_j, key_x) \land receive (A_{m_2}, key_x) \land$$

$$(\neg B_i B_j receive (A_{m_2}, key_x)) \lor (B_i B_j B_k owner (key_x, A_k) \land send (A_k, key_x) \land$$

$$(receive (A_{m_1}, key_x) \land \neg B_j B_k receive (A_{m_1}, key_x) \lor (B_i B_j B_k owner (key_x, A_k) \land$$

$$(B_j B_k receive (A_{m_1}, key_x) \land \neg B_i B_j B_k receive (A_{m_1}, key_x)))) \tag{3.3}$$

In the above expression, if the first disjunct is satisfied, $A_i$ uses $key_x$ as the key of channel($A_i, A_k$), $A_{m_2}$ receives $key_x$, but $A_i$ does not receive the identity of $A_{m_2}$. Thus, the second channel condition is not satisfied. If either the second or the third disjunct is satisfied, $A_i$ uses $key_x$ as the key of channel($A_i, A_k$), $A_{m_1}$ receives $key_x$, but $A_i$ does not receive the identity of $A_{m_1}$. Thus, again, the second channel condition is not satisfied. Hence the forwarding trust is necessary.

The negation of the message privacy trust can be written as follows:

$$send (A_i, msg_x) \land B_i receive (A_k, msg_x) \land receive (A_m, msg_x) \land$$

$$send (A_m, msg_x) \tag{3.4}$$

If expression (3.4) is satisfied, $A_i$ sends a secret message $msg_x$ to $A_k$, $A_m$ is able to decrypt $msg_x$, and $A_m$ sends $msg_x$ to some other agent. In this instance, the privacy condition of channel($A_i, A_k$) is not satisfied.

The negation of the key privacy trust can be written as follows:

$$B_i B_k owner (key_x, A_k) \land send (A_m, key_x) \tag{3.5}$$

If expression (3.5) is satisfied, $A_i$ uses $key_x$ as the key of channel($A_i, A_k$), and $A_m$ sends $key_x$ to some other agent whose identity $A_i$ may not know. Thus, the privacy condition of channel($A_i, A_k$) is not satisfied.

The negation of the trust against masquerading can be expressed as:

$$receive (A_i, msg_x) \land B_i send (A_k, msg_x) \land send (A_m, msg_x) \tag{3.6}$$

If expression (3.6) is satisfied, $A_i$ receives a message $msg_x$, $A_i$ determines that the sender of $msg_x$ is $A_k$, but the true sender of $msg_x$ is $A_m$. Thus, the authenticity condition of channel($A_i, A_k$) is not satisfied.
All of the above trusts, i.e., the authenticity trust, the forwarding trust, the key privacy trust, the trust against masquerading and the message privacy trust are necessary. This concludes the proof of Theorem 3.3.

The key and message privacy trusts exhibit an interesting property. In SKE-based composition of channel \((A_i, A_k)\) from channels \((A_i, A_j)\) and \((A_j, A_k)\), let the key of channel \((A_j, A_i)\) be \(key_y\). For channel \((A_j, A_i)\) to exist, for each agent \(A_n\) that possesses \(key_y\), message privacy trust \(T_{MP}(A_j, A_n, A_i)\) must be true. When \(A_j\) sends \(A_k\)'s key, \(key_x\), on channel \((A_j, A_i)\), agent \(A_n\) can receive \(key_x\). However, \(T_{MP}(A_j, A_n, A_i)\) requires that \(A_n\) not reveal a message sent on channel \((A_j, A_i)\) and hence not reveal \(key_x\). The condition that \(A_n\) not reveal \(key_x\) is exactly the requirement of the key privacy trust, \(T_{KP}(A_i, A_n, A_j)\). Thus, the validity of the key privacy trust, \(T_{KP}(A_j, A_n, A_i)\), follows from the validity of the message privacy trust, \(T_{MP}(A_j, A_n, A_i)\). This is summarized by the following theorem.

**Theorem 3.4 (Privacy Trust Theorem):** \(\forall A_i, A_j, A_k, A_n, key_x,\)

\[(B_i B_j B_k \text{owner}(A_k, key_x) \land T_{MP}(A_j, A_n, A_i)) \Rightarrow T_{KP}(A_i, A_n, A_k).\]

### 3.6. Composition of Two Dependent Channels

Suppose that channel \((A_i, A_k)\) is to be composed from channels \((A_i, A_j)\) and \((A_j, A_k)\). Let channels \((A_i, A_j)\) and \((A_j, A_k)\) be dependent channels with channel \((A_i, A_j)\) having been composed earlier from channels \((A_i, A_1), (A_1, A_2), ..., (A_1, A_j)\), and with channel \((A_j, A_k)\) having been composed earlier from channels \((A_k, A_f), (A_f, A_f), ..., (A_f, A_j)\) (see Figure 3.7). Being dependent channels, channels \((A_i, A_j)\) and \((A_j, A_k)\) have some trust requirements, and these trust requirements get carried over to channel \((A_i, A_k)\). In contrast, component channels in independent channel composition have no trust requirements.

![Figure 3.7: Dependent channel composition](image-url)
The channel composition mechanism is identical in independent and dependent channel composition, but the trust requirements are different. To see why, let $A_k$ send its key, $key_x$, to $A_j$ in a message $msg_{kj}$, and let $A_j$ forward $key_x$ to $A_i$ in a message $msg_{ji}$ to $A_i$. $Key_x$ is a public key in the PKE scheme and a single key in the SKE scheme. In the SKE scheme $msg_{kj}$ and $msg_{ji}$ are both secret messages encrypted using the algorithms of channels $(A_k, A_j)$ and $(A_j, A_i)$ respectively. In both SKE and PKE schemes, when $A_j$ receives $msg_{kj}$, $A_j$ authenticates $msg_{kj}$ using the algorithm of channel$(A_j, A_k)$, and when $A_i$ receives $msg_{ji}$, $A_i$ authenticates $msg_{ji}$ using the algorithm of channel$(A_i, A_j)$. However, the validities of these algorithms are contingent upon the satisfaction of the trust requirements of the respective channels with which the algorithms are associated. Thus the validities of messages $msg_{kj}$ and $msg_{ji}$, and the validity of $key_x$ transmitted in these messages, are dependent upon the trust requirements of the component channels.

Let us compute exactly the effects of the trust requirements of the component channels on the validity of $key_x$. The trust requirements of the component channels can in general be expressed as trust predicates, which are boolean combinations of the trusts such as the authenticity trust, the forwarding trust, etc. Given the truth or falsity of the various trusts in these boolean combinations, trust predicates can be evaluated to true or false. The satisfaction of a trust predicate associated with a channel is necessary and sufficient for the satisfaction of the authenticity and privacy conditions (given in Section 3.2) of the channel. Let $pred_{kj}$, $pred_{jk}$, $pred_{ji}$ and $pred_{ij}$ denote the trust requirements of channels $(A_k, A_j)$, $(A_j, A_k)$, $(A_j, A_i)$ and $(A_i, A_j)$ respectively. The effects of these trust predicates on channel$(A_i, A_k)$ are computed in four steps:

1. $A_k$ sends its key, $key_x$ to $A_j$ in $msg_{kj}$: In the PKE scheme, there are no trust-related computations at this step of the mechanism. In the SKE scheme, $msg_{kj}$ is sent as a secret message, and $A_k$ has to compute the set $set_{m_1}$ of all agents $A_{m_1}$ that can decrypt $msg_{kj}$. For each $A_{m_1}$, a key user-possessor trust involving $A_i$, $A_{m_1}$ and $A_k$ is required. Let $T_{KUP}(A_i, A_{m_1}, A_k) = \{ T_{KUP}(A_i, A_{m_1}, A_k) | A_{m_1} \in set_{m_1} \}$. The key user-possessor trust requirement is expressed as follows:

   \[ T_{KUP}(A_i, set_{m_1}, A_k) \Rightarrow channel(A_i, A_k) \tag{3.7} \]

   However, the set of agents that can decrypt $msg_{kj}$ depends on the encryption algorithm of channel$(A_k, A_j)$, which depends on $pred_{kj}$. This dependency is expressed by modifying formula (3.7) to:

   \[ pred_{kj} \Rightarrow (T_{KUP}(A_i, A_{m_1}, A_k) \Rightarrow channel(A_i, A_k)) \tag{3.8} \]

2. $A_j$ receives $msg_{kj}$: In both PKE and SKE schemes, $A_j$ authenticates $msg_{kj}$ using the algorithm of channel$(A_j, A_k)$, which is dependent on $pred_{jk}$. This dependency in the PKE scheme is expressed as:

\[ \text{If channel}(A_k, A_j), \text{channel}(A_j, A_k), \text{channel}(A_j, A_i) \text{ and channel}(A_i, A_j) \text{ had been independent channels, none of the trust predicates } pred_{kj}, pred_{jk}, pred_{ji} \text{ and } pred_{ij} \text{ would have been necessary, which is equivalent to saying that the trust predicates would have been equal to the boolean constant "TRUE".} \]

\[ \text{By trust-related computation at an agent we mean an encryption or decryption operation involving the key of another agent.} \]
pred_jk => channel(A_i, A_k)  

(3.9)

In the SKE scheme, this dependency is expressed by modifying formula (3.8) to:

pred_jk => (pred_jk => (T_KUP(A_i, set_m_1, A_k) => channel(A_i, A_k)))

(3.10)

(3) A_j forwards key_x in msg_{ji} to A_i: In the PKE scheme there are no trust-related computations at this step of the mechanism. In the SKE scheme, the key user-possessor trust is required in all agents A_m_2 that can decrypt the secret message msg_{ji}. This requirement is expressed by modifying formula (3.10) to:

T_KUP(A_i, set_m_2, A_k) => (pred_jk => (pred_jk => (T_KUP(A_i, set_m_1, A_k) => channel(A_i, A_k))))

(3.11)

However, the set of agents that can decrypt msg_{ji} depends on the encryption algorithm of channel(A_i, A_i), which depends on pred_ji. Thus, formula (3.11) gets modified to:

pred_ji => (T_KUP(A_i, set_m_2, A_k) => (pred_jk => (pred_jk => (T_KUP(A_i, set_m_1, A_k) => channel(A_i, A_k))))

(3.12)

(4) A_i receives msg_{ji}: In the PKE scheme, the authenticity trust is required of A_j. This requirement is expressed by modifying formula (3.9) to:

T_A(A_i, A_j, A_k) => (pred_jk => channel(A_i, A_k))

(3.13)

However, the validity of the determination of A_j as the sender of msg_{ji} is contingent upon the validity of the algorithm of channel(A_i, A_i) that A_i uses to authenticate msg_{ji}. This final dependency in the PKE scheme is expressed by modifying formula (3.13) to:

pred_ji => (T_A(A_i, A_j, A_k) => (pred_jk => channel(A_i, A_k)))

(3.14)

In the SKE scheme, three trusts involving A_i, A_j and A_k are required, namely, the authenticity trust, the forwarding trust and the key user-possessor trust. These trust requirements are expressed by modifying formula (3.12) to:

(T_A(A_i, A_j, A_k) \land T_F(A_i, A_j, A_k) \land T_KUP(A_i, A_j, A_k)) => (pred_ji =>

(T_KUP(A_i, set_m_2, A_k) => (pred_jk => (pred_jk => (T_KUP(A_i, set_m_1, A_k) =>

channel(A_i, A_k)))))

(3.15)

However, the identity of A_j used in the above expression is authentic only if the algorithm of channel(A_i, A_j) used by A_i to authenticate the received message msg_{ji}, is valid. Thus, we obtain the following final expression for the SKE scheme:

pred_ji => (T_A(A_i, A_j, A_k) \land T_F(A_i, A_j, A_k) \land T_KUP(A_i, A_j, A_k)) => (pred_ji =>

(T_KUP(A_i, set_m_2, A_k) => (pred_jk => (pred_jk => (T_KUP(A_i, set_m_1, A_k) =>

channel(A_i, A_k)))))

(3.16)
The following two theorems summarize all the above results regarding trust requirements in PKE- and SKE-based dependent channel composition.

**Theorem 3.5:** Let dependent channels \((A_i, A_j)\) and \((A_j, A_k)\) be composed to form channel \((A_i, A_k)\) using the PKE scheme. Let the trust predicates associated with the two component channels be \(\text{pred}_{ij}\) and \(\text{pred}_{jk}\), respectively. The trust requirements of channel \((A_i, A_k)\) are expressed as follows:

\[
\text{pred}_{ij} \Rightarrow (T_A(A_i, A_j, A_k) \Rightarrow (\text{pred}_{jk} \Rightarrow \text{channel}(A_i, A_k)))
\]

\[\square\]

**Theorem 3.6:** Let dependent channels \((A_i, A_j)\), \((A_j, A_k)\), \((A_j, A_k)\) and \((A_k, A_j)\) be composed to form \((A_i, A_k)\) using the SKE scheme. Let the trust predicates associated with the four component channels be \(\text{pred}_{ij}\), \(\text{pred}_{ji}\), \(\text{pred}_{jk}\) and \(\text{pred}_{kj}\), respectively. Further, let the sets of agents that can decrypt secret messages on channel \((A_k, A_j)\) and channel \((A_j, A_i)\) be \(\text{set}_{m1}\) and \(\text{set}_{m2}\), respectively. The trust requirements of channel \((A_i, A_k)\) are expressed as follows:

\[
\text{pred}_{ij} \Rightarrow ((T_A(A_i, A_j, A_k) \land T_F(A_i, A_j, A_k) \land T_{KUP}(A_i, A_j, A_k)) \Rightarrow (\text{pred}_{ji} \Rightarrow (T_{KUP}(A_i, \text{set}_{m2}, A_k) \Rightarrow (\text{pred}_{kj} \Rightarrow (\text{pred}_{kj} \Rightarrow (T_{KUP}(A_i, \text{set}_{m1}, A_k) \Rightarrow \text{channel}(A_i, A_k)))))))
\]

\[\square\]

Notice that, since an independent channel is a special case of a dependent channel, in which the trust predicate associated with the channel is the constant "TRUE", a composition involving an independent channel and a dependent channel is a special case of dependent channel composition, and hence does not require a separate analysis.

We conclude this section with the observation that, as channels are composed to form newer channels, the trust requirements propagate. With each channel composition, the number of trusts required for the composed channel increases by a factor of two for the PKE scheme and by a factor of four for the SKE scheme.

### 3.7. Composition of More Than Two Channels

A distributed system provides independent channels at the time of system configuration. By Theorem 3.1, any other channel in the system must be composed from independent or dependent channels using a sequence of two-channel compositions. Some of these two-channel compositions will involve only independent channels, and some will involve dependent channels that are results of earlier two-channel compositions in the sequence. We now illustrate the analysis of trust requirements in a sequence of two-channel compositions. Notice that, after a channel has been established by composing a sequence of existing channels, the messages on the channel do not have to follow the same route as the channel establishment messages. However, any agent that has the channel key can obtain a message on the channel by eavesdropping on whatever route the message takes. Thus, the route taken by the messages does not affect the trust requirements of the channel.

Figure 3.8 shows a sample distributed system, in which there are agents \(A_i, A_j, A_j, A_j, A_j\) and \(A_k\), and there are existing independent channels \((A_i, A_j), (A_j, A_j), (A_j, A_j), (A_j, A_j)\) and \((A_j, A_j)\). Figure 3.8 can be thought of as part of the system's hierarchical name space [Lu86, TPR84], in which \(A_j, A_j, A_j, A_j\) and \(A_j, A_j\) are name servers and there are independent channels between each node and its parent.
Suppose a new channel, \((A_i, A_k)\), is to be established. Several sequences of two-channel compositions can be used. One such sequence is:

1. channels \((A_{j2}, A_{j3})\) and \((A_{j3}, A_k)\) are composed to form channel\((A_{j2}, A_k)\),
2. channels \((A_{j1}, A_{j2})\) and \((A_{j2}, A_k)\) are composed to form channel\((A_{j1}, A_k)\), and
3. channels \((A_i, A_{j1})\) and \((A_{j1}, A_k)\) are composed to form channel\((A_i, A_k)\).

The intermediate channels, channel\((A_{j2}, A_k)\) and channel\((A_{j1}, A_k)\), are both dependent channels. In the following two sub-sections, we analyze the trust requirements in the above sequence of channel compositions using PKE and SKE schemes.

### 3.7.1. Sequence of PKE-based Channel Compositions

In the PKE scheme, each of the three compositions in the sequence requires an authenticity trust, and hence channel\((A_i, A_k)\) requires a conjunction of all the three authenticity trusts:

\[
T_A(A_{j2}, A_{j3}, A_k) \land T_A(A_{j1}, A_{j2}, A_k) \land T_A(A_i, A_{j1}, A_k) \tag{3.17}
\]

### 3.7.2. Sequence of SKE-based Channel Compositions

SKE-based two-channel composition requires bidirectional component channels, hence, all the independent channels must be bidirectional, and all the intermediate dependent channels must be established in both directions. Thus, establishing channel\((A_i, A_k)\) in the system of Figure 3.8 consists of a sequence of five compositions. Thus, five authenticity trusts are required:

\[
T_A(A_{j2}, A_{j3}, A_k) \land T_A(A_k, A_{j3}, A_{j2}) \land T_A(A_{j1}, A_{j2}, A_k) \land T_A(A_k, A_{j2}, A_{j1}) \land T_A(A_i, A_{j1}, A_k) \tag{3.18}
\]

Each of the five channel compositions also gives rise to a forwarding trust, resulting in the following predicate of forwarding trusts:

\[
T_F(A_{j2}, A_{j3}, A_k) \land T_F(A_k, A_{j3}, A_{j2}) \land T_F(A_{j1}, A_{j2}, A_k) \land T_F(A_k, A_{j2}, A_{j1}) \land
\]
Evaluating the key user-possessor trust requirements consists of evaluating the identities of agents other than \( A_i \) and \( A_k \) who may have obtained the keys of the five newly established channels when they participated in the sequence of channel compositions leading to the establishment of the new channels. Let us assume that the key of each independent channel is known only to the two ends of the independent channel. In the first two compositions, the keys of newly established channels \((A_{j2}, A_k)\) and \((A_k, A_{j2})\) are known only to \( A_{j3} \) in addition to \( A_{j2} \) and \( A_k \). Thus, the first two compositions require two key user-possessor trusts, one for each composition:

\[
T_{KUP}(A_{j2}, A_{j3}, A_k) \land T_{KUP}(A_k, A_{j3}, A_{j2})
\]

In the next two compositions, channels \((A_{j1}, A_k)\) and \((A_k, A_{j1})\) are established from channels \((A_{j1}, A_{j2})\) and \((A_{j2}, A_k)\). In these compositions, not only \( A_{j2} \), but also \( A_{j3} \) can obtain the keys of channel\((A_{j1}, A_k)\) and channel\((A_k, A_{j1})\). Thus, the key user-possessor trust requirements proliferate, and the key user-possessor trust requirements for the third and the fourth compositions are:

\[
T_{KUP}(A_{j1}, A_{j2}, A_k) \land T_{KUP}(A_{j1}, A_{j3}, A_k) \land T_{KUP}(A_k, A_{j2}, A_{j1}) \land
T_{KUP}(A_k, A_{j3}, A_{j1})
\]

In the final composition, channel\((A_i, A_k)\) is established from channels \((A_i, A_{j1})\) and \((A_{j1}, A_k)\). Agents \( A_{j1}, A_{j2} \) and \( A_{j3} \) can all obtain the key of channel\((A_i, A_k)\). Thus the key user-possessor trust requirements for the final composition are:

\[
T_{KUP}(A_i, A_{j1}, A_k) \land T_{KUP}(A_i, A_{j2}, A_k) \land T_{KUP}(A_i, A_{j3}, A_k)
\]

The key user-possessor trust requirement for the entire sequence of compositions is a conjunction of the three formulae (3.20), (3.21), and (3.22).

It is interesting to observe that the trust expressions are not symmetric in \( A_{j1}, A_{j2} \) and \( A_{j3} \). Hence, different channel composition sequences, even when they use the same set of independent channels and establish the same final channel, may require different trust relationships.

### 3.8. Differences between PKE and SKE Schemes

It is clear from the previous sections that channel compositions using PKE and SKE schemes require different trust relationships. In fact, the PKE scheme requires only a small subset of the trusts required by the SKE scheme.

Using informal arguments, Popek and Kline [KLP79] claim that PKE and SKE schemes have identical trust requirements. Because of their informal approach to trust, Popek and Kline were not able to see differences in trust requirements between the PKE and SKE schemes, and hence a number of advantages of the PKE scheme were not identified. To compare their approach with ours, we briefly describe Popek and Kline's approach to determining trust requirements in PKE and SKE-based channel composition mechanisms.

Consider the composition of channels \((A_i, A_j)\) and \((A_j, A_k)\) to form channel\((A_i, A_k)\). In the PKE scheme, each channel requires a (public-key, private-key) pair. In particular, \( A_j \) must possess its private key and \( A_k \)’s public key. \( A_j \) must keep its private key secret. For channel\((A_i, A_k)\) to be established, \( A_j \) forwards \( A_k \)’s public key to \( A_i \). In the SKE scheme, each channel requires a single key. Channels \((A_i, A_j)\) and \((A_j, A_k)\) require \( A_j \) to possess their single
keys. During the establishment of channel(Ai, Ak), A1 may come to know the single key of that channel (since A1 is involved in forwarding it). A1 has to keep all the single keys in its possession secret. However, notice that A1 can encrypt all the single keys in its possession with another key keyoverall and keep keyoverall secret. Since in the PKE scheme A1 had to keep its own private key secret anyway, one can conclude that in both PKE and SKE schemes A1 is trusted to keep one key secret. Thus, the trust requirements in both the PKE and SKE schemes are the same.

In the approach described above, Popek and Kline are using the notion of trust to represent the secret-keeping behavior of an agent, and ignore all other aspects of agent behavior. This lack of consideration of all the inherent assumptions in PKE and SKE-based channel composition mechanisms is a major drawback of their approach, and is a consequence of its informality. Assumptions about keeping secrets are just one kind of assumptions necessary in channel composition mechanisms. For instance, in the SKE-based channel composition scenario, A1 is assumed not only to keep the single key of channel(Ai, Ak) secret, but also not to use that single key either for masquerading as Ak or for decrypting a secret message on channel(Ai, Ak). It should be noted that, in our formal approach to trust, we capture all these assumptions.

Even with regard to assumptions about keeping secrets, there are three major differences between the two approaches. The differences are best explained with reference to the composition of channels (Ai, Aj) and (Aj, Ak).

1. In our approach (unlike Popek and Kline's), there is no trust requirement involving an agent Aj if Aj has to keep its own private key secret. Trust requirements involving Aj arise only when Aj has to keep the key of some other agent or channel secret. To illustrate this difference, notice that, in the PKE scheme, if Aj's private key gets compromised after channel(Ai, Ak) has been established, only A1 needs to change its private key, and Ai and Ak are unaffected. In the SKE scheme, A1 has to keep the key of channel (Ai, Ak) secret, and if this key gets compromised, Ai and Ak have to re-establish channel (Ai, Ak) whereas A1 is unaffected. Popek and Kline's approach does not yield these distinctions.

2. Unlike Popek and Kline's approach, in our approach two trusts are different if the security losses due to their being ill posed are different. To see this point, notice that, in the PKE scheme, if Aj's private key gets compromised after channel(Ai, Ak) has been established, security losses may involve resources owned by Aj. In the SKE scheme, suppose Aj encrypts all the single keys in its possession with a key keyoverall, and keeps keyoverall secret. Security losses resulting from a leak of keyoverall may involve resources owned by Ai, Aj, and Ak. Thus, the security losses due to a leak of Aj's private storage are different in the PKE and SKE schemes, and hence, in our approach, their trust requirements are vastly different. Popek and Kline's approach does not make these distinctions, as is clear from the fact that their approach equates the trust concerned with keeping Aj's private key secret with that with keeping keyoverall secret.

3. Popek and Kline's approach, unlike ours, captures only the steady state assumptions with regard to keeping secrets. Thus, in SKE-based channel composition, even though A1 can store all the single keys encrypted with another key keyoverall, there is a finite time period preceding the encryption by keyoverall during which more than one single key has to be kept secret.

To further see how our formal approach yields differences between PKE and SKE schemes, notice that, in our approach, the SKE scheme requires four more trusts than the PKE scheme, namely, the forwarding trust, the message privacy trust, the trust against masquerading, and the key privacy trust. As we will now show, each of these additional trusts gives rise to important
differences between PKE and SKE schemes. The usual channel composition scenario involving
$A_i, A_j$ and $A_k$ is used throughout.

3.8.1. Differences Arising from the Forwarding Trust

Consider the SKE channel composition scenario depicted in Figure 3.6. Suppose that $A_{m1}$ obtained $A_k$'s key by decrypting $msg_{kj}$. A security attack on $A_{m1}$ can make the key available to the attacker, who can then compromise the security of channel($A_i, A_k$). Thus, $A_i$ must constantly monitor the security situation at $A_{m1}$, and $A_i$ must invalidate channel($A_i, A_k$) the moment it detects a security attack on $A_{m1}$. For $A_i$ to take these actions, $A_i$ must know the identity of $A_{m1}$, which is what is exactly abstracted by the forwarding trust. In contrast, in the PKE scheme, $A_{m1}$ can only obtain the public key of $A_k$, and an attack on $A_{m1}$ may reveal $A_k$'s public key, but that does not pose any security danger to channel($A_i, A_k$).

3.8.2. Differences Arising from the Message Privacy Trust

In the SKE scheme, $A_j$ can decrypt secret messages sent by $A_i$ on channel($A_i, A_k$). Suppose there is a security attack on $A_j$ after channel($A_i, A_k$) has been established. Any secret message on channel($A_i, A_k$) that $A_j$ might possess becomes available to the attacker. In the event of a similar attack in the PKE scheme, $A_j$, since it does not possess $A_k$'s private key, cannot decrypt secret messages on channel($A_i, A_k$), and hence secret messages on channel($A_i, A_k$) remain unavailable to $A_j$'s attacker.

3.8.3. Differences Arising from the Trust Against Masquerading

In the SKE scheme, $A_j$ may have the key of channel($A_i, A_k$) in its possession. Thus, $A_j$ can send messages masquerading as $A_k$ on channel($A_i, A_k$). In the PKE scheme, $A_j$, since it does not possess $A_k$'s private key, cannot masquerade as $A_k$ on channel($A_i, A_k$).

3.8.4. Differences Due to the Key Privacy Trust

In the SKE, $A_j$ has the single key of channel($A_i, A_k$). Suppose there is a security attack on $A_j$ after channel($A_i, A_k$) has been established. Furthermore, suppose the attacker has kept track of all encrypted messages exchanged on channel($A_i, A_k$). Once the attacker obtains the key from $A_j$, all past, present and future secret messages on channel($A_i, A_k$) may become available to the attacker. In the event of a similar attack in the PKE case, since $A_j$ does not possess the private key of $A_k$, past, present and future secret messages on the channel remain unavailable to the attacker. This difference has a significant impact on key caching and can be illustrated as follows. Consider $A_k$'s key which is used as the key of channel($A_i, A_k$). When key caching is used, at different moments different agents may cache $A_k$'s key. Some of these agents may not have obtained $A_k$'s key for the purpose of encrypting messages to $A_k$, but may have obtained it together with some other keys from the name server for the sole purpose of caching them. In the SKE scheme, since any agent that may have cached $A_k$'s key can decrypt all past, present, and future messages on channel($A_i, A_k$), $A_i$ must place key user-possessor trust in every agent at which $A_k$'s key may have been cached at some instant in time, even when channel($A_i, A_k$) is no longer in use. Such enormous trust requirements make caching highly unattractive. In contrast, in the PKE scheme, caching of public keys can be used without limitations. Note that, in distributed systems, caching in general is highly desirable from the viewpoint of performance [Ter].
3.8.5. Differences with Regard to Replication

We will now illustrate that replication makes trusts easier to satisfy in the PKE scheme, but makes trusts harder to satisfy in the SKE scheme.

Consider the scenario shown in Figure 3.9, in which there are independent channels (ibm, ibm-j), (ibm-j, jap), (ibm, sony-us), and (sony-us, jap). Suppose a new channel called channel\(_A\) is established between ibm and jap by composing (ibm, ibm-j) and (ibm-j, jap). In PKE-based composition, ibm obtains a key, key\(_A\), which is guaranteed to be the public key of jap if the trust requirement (obtained using Theorem 3.3) \(T_A(\text{ibm}, \text{ibm-j}, \text{jap})\) for channel\(_A\) is satisfied. In SKE-based composition, ibm obtains a key, key\(_A\), which is guaranteed to be the single key of jap if the trust requirement (obtained using Theorem 3.4) \(T_A(\text{ibm}, \text{ibm-j}, \text{jap}) \land T_F(\text{ibm}, \text{ibm-j}, \text{jap}) \land T_{KUP}(\text{ibm}, \text{ibm-j}, \text{jap})\) for channel\(_A\) is satisfied.

Suppose a second channel, channel\(_B\), is established between ibm and jap by composing (ibm, sony-us) and (sony-us, jap). In PKE-based composition, ibm obtains a key, key\(_B\), which is guaranteed to be the public key of jap if the trust requirement \(T_A(\text{ibm}, \text{sony-us}, \text{jap})\) for channel\(_B\) is satisfied. In SKE-based composition, ibm obtains a key, key\(_B\), which is guaranteed to be the single key of jap if the trust requirement \(T_A(\text{ibm}, \text{sony-us}, \text{jap}) \land T_F(\text{ibm}, \text{sony-us}, \text{jap}) \land T_{KUP}(\text{ibm}, \text{sony-us}, \text{jap})\) for channel\(_B\) is satisfied.

Suppose ibm compares the keys, key\(_A\) and key\(_B\), and uses either one of them for channel(\text{ibm}, \text{jap}) only if both are identical. In the PKE scheme, if at least one of \text{ibm-j} or \text{sony-us} returns the valid public key of jap, the security of channel(\text{ibm}, \text{jap}) is guaranteed. Thus, the trust relationship required for channel(\text{ibm}, \text{jap}) is a disjunction of the trust relationships for channel\(_A\) and channel\(_B\) (also see [VeA87]):

![Diagram of channel replication](image-url)
Thus, the trust requirements decrease with replication in the PKE scheme.

In the SKE scheme, even though it is necessary that at least one of \texttt{ibm}_j or \texttt{sony}_us return the valid single key of \texttt{jap}, both \texttt{ibm}_j and \texttt{sony}_us may possess the single key of \texttt{jap}, and the forwarding and key user-possessor trusts are required of both agents. The forwarding and key user-possessor trusts requirements are:

\[
T_F(\texttt{ibm}, \texttt{ibm}_j, \texttt{jap}) \land T_{KUP}(\texttt{ibm}, \texttt{ibm}_j, \texttt{jap}) \land T_F(\texttt{ibm}, \texttt{sony}_us, \texttt{jap}) \land T_{KUP}(\texttt{ibm}, \texttt{sony}_us, \texttt{jap})
\]

This shows that the forwarding and key user-possessor trusts increase with replication in SKE scheme. Thus, from the viewpoint of security, replication is advantageous in PKE systems but disadvantageous in SKE systems.

In conclusion, our formal analysis has revealed several differences between PKE and SKE schemes, and these differences make PKE schemes much more attractive than SKE schemes in large distributed systems.

3.9. Conclusion

Trust arises primarily in establishing channels for secure communication. The only way to establish a new channel is by composing a sequence of existing adjacent channels. There are two kinds of channels: independent channels, which have no trust requirements and are provided by the system at configuration time, and dependent channels, which are composed from independent channels and have trust requirements. Channel composition mechanisms are commonly based on either public key encryption (PKE) or single key encryption (SKE). PKE-based channel composition requires what we have called \textit{authenticity trusts}, which are functions of three agents. SKE-based channel composition has much larger trust requirements than PKE-based channel composition. The differences in trust requirements of PKE and SKE-based channel compositions translate to significant advantages of PKE over SKE-based channel composition with respect to replication, caching, permanence of trust requirements, and so on. Different sequences of compositions, even though they use the same set of independent channels and establish the same final channel, have different trust requirements. Thus, our analyses provide insight into the basic structure and limitations of mechanisms with regard to their trust requirements.
We show that it is desirable to have a tree of independent channels in a distributed system, and that this tree represents the global name space of the system. To establish a channel between two agents in a name space, there are two alternatives for the order in which the independent channels in the path between the two agents can be composed, and they are called \textit{iterative} and \textit{recursive}. These two channel composition orders have different trust requirements and exhibit interesting duality properties. We develop algorithms for synthesizing name spaces so that, given a channel composition order and the actual trusts of all agents, channel composition between any two agents requires only a subset of the given set of trusts. The given trusts are in general functions of three agents, but they can also be functions of two agents, in which case the algorithms are simpler. We derive some NP-completeness results with respect to putting bounds on the size of the database of encryption keys stored at each node in a name space. Sample runs of the algorithms show that small differences in trust relationships can cause substantial differences in the resulting name spaces.

4.1. Introduction

Agents sharing a distributed system have trust relationships among themselves. One of the most important applications of a formal theory of trust consists of synthesizing a distributed system that satisfies the trust relationships of the agents in the system. The synthesis of a system from trust specifications was in fact the eventual goal with which we began our formal study of trust.

In this chapter, we show how a distributed system's name space determines the trust requirements needed for channel composition between every pair of agents, and we develop algorithms for synthesizing a name space so as to satisfy a given set of trust specifications of agents. Section 4.2 shows the association between a name space and a tree of independent channels. In Section 4.3, we show that there are two alternatives for the order in which the channels between two agents in a name space can be composed, which are called \textit{iterative} and \textit{recursive}, and examine their trust properties. In Sections 4.4-4.9, we develop polynomial-time algorithms for synthesizing name spaces given actual trusts of all agents. Each node in the name space stores a database with the encryption keys of all its children, and it is desirable to put bounds on the size of this database. We derive some NP-completeness results in this regard. The polynomial-time name space synthesis algorithms to be described in Sections 4.4-4.9 have been implemented and experimented with. Section 4.10 presents some interesting sample runs of these algorithms, and finally, Section 4.11 concludes the chapter.

4.2. Necessity of Fast Channel Establishment Procedures in a VLDS

Any two agents in a distributed system must be able to communicate securely (see Figure 4.1). One way to achieve this is to have an independent channel between each pair of agents in the system (see Figure 4.2). In such a system, no new channels need be established, and hence there are no trust requirements for communication. However, there are several disadvantages in having independent channels between all pairs of agents. Since independent channels have to be established using external mechanisms, there would be \(O(n^2)\) channels that need to be
established using such mechanisms. External mechanisms, e.g., trusted couriers, are expensive, extremely slow and cumbersome. Since having an independent channel to an agent requires storing the agent's encryption keys, each agent would have to store the entire database of encryption keys of all other agents.

Such a scheme has numerous performance disadvantages in a large distributed system. For example, when an agent changes its keys, the agent has to inform every other agent of the change through external mechanisms. When a new agent joins a distributed system, the new agent has to choose \( n \) different channel keys and exchange its keys with every other agent in the system through external mechanisms. In summary, in a large distributed system, it is not desirable to have a independent channel between each pair of agents. The number of independent channels must be minimized.

Given an initial set of independent channels in a distributed system, any two agents must be able to establish a channel by composing the independent channels and any existing dependent channels between them. Therefore, if we represent the agents as nodes and independent channels as edges in a graph, the initial graph of independent channels must be connected.

Let us suppose that we have a connected graph of independent channels. It was shown in the previous chapter that the trust relationships required in establishing a channel between two agents is determined by both the independent channels between the two agents and the order of compositions of these independent channels. Thus, to establish a channel to an agent Ibaraki, an agent such as Alice would have to keep a database of its trust relationships, and find a path to Ibaraki in the graph of independent channels such that the trust relationships in composing the independent channels in the path in some order are present in the database. The channel establishment procedure would involve trying out a sequence of independent channels, backtracking if either no path is found beyond a node in the graph or a path is found but the path requires

---

![Figure 4.1: Agents in a sample VLDS](image-url)

---
trust relationships not present in its database, and so on (see Figure 4.3). The worst-case performance of such a channel establishment procedure would be intolerable.

The goal of this chapter is to investigate whether, given the trust relationships of all agents at system configuration time, we can synthesize a graph of independent channels so that, between any two agents there is a path in the graph, and composition of the independent channels in the path in a pre-specified order requires only a subset of the given set of trust relationships. Thus, an agent would not have to keep a database of its trust relationships, and at channel establishment time the agent would not have to check either the existence of a path or the satisfiability of a path's trust requirements. Consequently, the channel establishment procedure would be much faster.

What kind of a graph of independent channels should we synthesize? It is desirable to minimize the number of independent channels. A connected graph with a minimum number of edges is a tree. Thus, our goal is to synthesize a tree of independent channels. We will later examine the kinds of trees that are more preferable than others.

In a tree of independent channels connecting $n$ nodes, there are exactly $n-1$ independent channels yielding an average of one independent channel per agent, and there is a unique path between each pair of agents. An agent in the tree must be able to determine the unique path
between itself and any other agent without searching the entire tree. For instance, in Figure 4.3, to establish a channel to *ibaraki*, it should not be necessary for *alice* to establish a channel to *ibm*, backtrack after finding that there is no further path from *ibm* towards *ibaraki*, then establish a path to *ibm*–*j*, again backtrack since there is no further path from *ibm*–*j* to *ibaraki*, and then establish a path to *ibaraki* through *jap*. In other words, given the names of two agents, it should be possible to write down the independent channels between them without having to traverse the entire tree.

To accomplish a fast translation from the names of two agents to the independent channels between them, we encode the independent channels into the agents’ names. Specifically, one of the agents in the tree is designated as the root of the tree. This agent acts as the reference point for naming the nodes of the tree. The name of an agent in the tree, referred to as its *pathname*, encodes the independent channels from the root to the agent. Given the pathnames of two agents, one can easily write down the sequence of independent channels in the path between them. Figure 4.4 illustrates this mechanism. In the tree shown in the figure, the agent named *world* is designated as the root. Each pathname begins with a “/”, which denotes the root node (*world*). The pathname of *Alice* is /usa/ibm/alm/alice and that of *Ibaraki* is /jap/ibm-jiibaraki. Given the pathnames /usa/ibm/alm/alice and /jap/ibm-jiibaraki, the independent channels between *Alice* and *Ibaraki* are (alice, alm), (alm, ibm), (ibm, usa), (usa, root), (root, jap), (jap, ibm_j), and (ibm_j, ibaraki). Since the tree of independent channels determines the...
pathnames of agents, the tree is referred to as a name space of the distributed system. Each node in the name space has independent channels to its children and to its parent. If independent channels are PKE-based, each node keeps a database of public keys of its children and its parent.

In practice, there are several factors other than security, such as administrative and geographical factors, that must be considered in designing a name space for a distributed system. A name space that is optimal from the viewpoint of security may not be so from the viewpoint of the other factors. As explained in Section 1.2, having two separate name spaces, one for security purposes and a second one for other purposes, has significant performance drawbacks. Thus, in practice, it is desirable to design a single name space, and the design must be carried out as a compromise among several objectives, both security and non-security oriented.

It should be noted that a tree-structured name space does not imply a hierarchical trust pattern. To see why, notice that a tree is hierarchical with respect to a property $P$ if in the tree, whenever $P$ holds for a node, $P$ also holds for the node's parent. In a tree-structured name space, the trust relationships of a node need not form a subset of the trust relationships of the node's parent. The number of trust relationships of a node can be much larger than that of the node's parent.

Figure 4.5 illustrates the non-hierarchical nature of a name space tree. Let $A$ be a child of $B$, and $M$ be a node that is not a descendent of $B$. $B$ is on the path from $A$ to each of $A$'s non-descendents. Thus, $A$ requires a trust relationship involving $B$ for establishing a channel to each of $A$'s non-descendents. $C$ is on the path from $A$ to each of $B$'s non-descendents. Thus, $A$ requires a trust relationship involving $C$ for establishing a channel to each of $B$'s non-descendents. The set of nodes that are non-descendents of $B$ is a subset of the set of $A$'s non-descendents. Thus, there are more trust relationships involving $A$ and $B$ than those involving $A$
and $C$. Since $A$ is a child of $B$, and the number of $B$'s children can be arbitrarily large, there can be more trust relationships involving $B$ than $C$, even though $C$ is $B$'s parent in the tree. None of these trust relationships involve $A$'s descendents, and, using similar arguments, it can be shown that a node in the tree need not fully trust all its descendents. Thus, there is no hierarchical pattern with respect to trust relationships in a tree-structured name space. This can also be inferred from the observation that the selection of the root, which determines the parent-child relationships in the tree, can be arbitrary: any node can be designated as the root of the tree. The root just acts as the reference node for naming all other nodes.

4.3. Channel Composition Algorithms

Suppose that a distributed system has a name space such as the one shown in Figure 4.4. Given the names of two agents, say Alice and Ibaraki, we can write down the independent channels in the path between them. There are two possibilities for the order in which these independent channels can be composed to form the required channel between Alice and Ibaraki, called iterative and recursive channel composition, respectively. These two channel composition orders are illustrated next.

4.3.1. Iterative Channel Composition

Consider the case of channel establishment from Alice to Ibaraki in the name space of Figure 4.4. The iterative channel composition algorithm composes channels beginning from Alice (see Figure 4.6) and consists of the following steps (see Figure 4.7):

* Alice makes a remote invocation $F_1$ to the node next in the path from Alice to Ibaraki, namely alm, requesting the encryption key of ibm. alm returns ibm's key in the return message $R_1$. This in effect composes the channels $alice$-$alm$ and $alm$-$ibm$ to form the channel $alice$-$ibm$.
Alice makes a remote invocation $F_2$ to ibm requesting the key of usa. The return message $R_2$ containing usa's key results in the composition of channels alice-ibm and ibm-usa to from the channel alice-usa.

Alice continues to make such remote invocations, and compositions of channels continue until the channel alice-ibaraki is established. Since alice repeatedly makes remote invocations in this algorithm, the algorithm is called iterative channel composition.

Figure 4.8 illustrates the trust relationships in iterative channel composition. C is B's parent node, N is any descendent of B, and M is any non-descendent of B (i.e., any node not in the subtree rooted at B). To establish channel $M-N$ using iterative channel composition, M successively establishes channels to nodes in the path from M to N. At some step M establishes a channel to C, and at the successive step it establishes a channel to $B$ by composing channels $M-C$ and $C-B$. Using the results of Section 3.4, $T_A(M, C, B)$ is true. Similarly, to establish channel $N-M$, N successively establishes channels to nodes in the path from N to M. At some step N establishes a channel to B, and at the successive step it establishes a channel to C by composing channels $N-B$ and $B-C$. Hence, $T_A(N, B, C)$ is true. The next theorem summarizes these results. We will use the term "node_1 trusts node_2 for node_3" to mean that $T_A(node_1, node_2, node_3)$ is true.

Theorem 4.1: In iterative channel composition, all non-descendents of a node trust the node's parent for the node, and all descendents of a node trust the node for the node's parent.

4.3.2. Recursive Channel Composition

Consider the case of channel establishment from Alice to Ibaraki in the name space of Figure 4.4. The recursive channel composition algorithm composes channels beginning from Ibaraki ¹ (see Figure 4.9) and consists of the following steps (see Figure 4.10):

Alice makes a remote invocation $F_1$ to alm, which is the next node in the path from alice to ibaraki, requesting the encryption key of ibaraki.

When alm receives $F_1$ request, alm makes a remote invocation $F_2$ to ibm, requesting the public key of ibaraki.

This sequence of remote invocations continues till finally jap makes a remote invocation $F_6$ to ibm_j which is the node just ahead of ibaraki in the path from alice to ibaraki, requesting the key of ibaraki.

Ibm_j, which has ibaraki's key, sends the key in a return message $R_1$. This in effect composes the channels jap-ibm_j and ibm_j-ibaraki to form the channel jap-ibaraki.

This sequence of return messages and channel compositions continues, and towards the end the return message $R_5$ results in the composition of channels alm-ibm and ibm-ibaraki to form the channel alm-ibaraki.

Finally, the return message $R_6$ results in the composition of channels alice-alm and alm-ibaraki to form the required channel alice-ibaraki.

¹ It should however be noted that, even though channel composition begins from Ibaraki, the initiative always starts from Alice.
Figure 4.6: An instance of iterative channel composition. In channel establishment from Alice to Ibaraki, channel composition starts from Alice.

Figure 4.7: Remote invocations in iterative channel composition. In channel establishment from Alice to Ibaraki, Alice successively makes remote invocations to nodes in its path to Ibaraki.
Figure 4.8: Trust relationships in iterative channel composition
Figure 4.9: An instance of recursive channel composition. In channel establishment from Alice to Ibaraki, channel composition starts from Ibaraki.

Figure 4.10: Remote invocations in recursive channel composition. In channel establishment from Alice to Ibaraki, Alice makes a recursive remote invocation to its parent.
In this algorithm, *alice* makes a remote invocation $F_1$ within which there is a remote invocation $F_5$, and so on. Thus *alice* makes a recursive remote invocation, and the algorithm is called *recursive* channel composition.

Figure 4.11 illustrates the trust relationships in recursive channel composition. $C$ is $B$'s parent node, $N$ is any descendant of $B$, and $M$ is any non-descendant of $B$. To establish channel $N-M$ using recursive channel composition, channels are successively established to $M$ from nodes in the path $M-N$ in the name space. At some step $C$ establishes a channel to $M$, and at the successive step $B$ establishes a channel to $M$ by composing channels $B-C$ and $C-M$. Using the results in Section 3.4, $T_A(B, C, M)$ is true. Similarly, to establish channel $M-N$, channels are successively established to $N$ from nodes in the path $N-M$ in the name space. At some step $B$ establishes a channel to $N$, and at the successive step $C$ establishes a channel to $N$ by composing channels $C-B$ and $B-N$. Hence $T_A(C, B, N)$ is true. Theorem 4.2 summarizes these results.

**Theorem 4.2:** In recursive channel composition, a node trusts its parent for the node's non-descendants, and the parent of the node trusts the node for the node's descendants.

Having obtained the trusts required by iterative and recursive channel composition in Theorems 4.1 and 4.2 above respectively, we are now ready to tackle the design problem mentioned at the beginning of this chapter: Assuming that the trust relationships of all agents and a channel composition order are given, a tree-structured name space is to be synthesized so that only a subset of the given set of trust relationships is necessary for establishing a channel between any pair of agents in the distributed system. The next section discusses the nature of trust relationships that might be specified in such a design problem.

![Figure 4.11: Trust relationships in recursive name resolution](image-url)
4.4. Trust Specifications

In Section 3.7, it was shown that the trust relationships required by PKE-based channel composition protocols form a proper subset of the trust relationships required by SKE-based channel composition protocols. However, current hardware implementations of SKE are several orders of magnitude faster than those of PKE. In the next chapter, we will present protocols that establish channels using PKE but switch-over to SKE once a channel has been established. These protocols have the smaller trust requirements of PKE-based channel composition but have the performance of SKE. Thus, in synthesizing name spaces we assume that the trust requirements are those of PKE-based channel composition. By Section 3.4, we know that PKE-based channel composition requires authenticity trusts, which are (boolean) functions of three agents. Thus, we assume that trust specifications for synthesizing name spaces are in general 3-agent authenticity trust predicates. Nevertheless, the synthesis algorithms that we develop in the following sections can also be used for designing SKE-based name spaces, if, wherever we check for the satisfaction of the authenticity trust involving three agents, we also check for the satisfaction of the forwarding trust and of the key user-possessor trust involving the same three agents.

However, trust specifications can also be functions of two agents. The next section investigates such trust specifications.

4.5. Two-Agent Trust Specifications

When an agent specifies its trust relationships, it is common for the trust relationships to take one of the following two forms:

1. An agent trusts another agent,
2. An agent is trusted for another agent.

Such trust relationships involve two agents, and can be thought of as simplifications of the general 3-agent trust predicates required in channel composition. The first of the 2-agent trust relationships above can be thought of as resulting from the elimination of the last argument of a 3-agent trust relationship of the form, "an agent A trusts an agent B for every other agent". We will denote such a 2-agent trust relationship by $T_A(A, B, *)$. The second 2-agent trust relationship can be thought of as resulting from the elimination of the first argument of a 3-agent trust relationship of the form, "every agent trusts an agent B for an agent C". We will denote such a 2-agent trust relationship by $T_A(*, B, C)$.

The permutation of the two types of 2-agent trust relationships with iterative and recursive channel composition results in four combinations. These four combinations have some interesting properties, which we investigate in next.

4.5.1. Iterative and Recursive Channel Composition

Suppose that trust specifications are of the form $T_A(node_2, node_1, *)$, and that the channel composition order is iterative. Consider any node $B$ in the name space. Let $N$ be any descendant of $B$, and $C$ be the parent of $B$ (see Figure 4.12). By Theorem 4.1, $T_A(N, B, C)$ is true. Since trust specifications are all of the form $T_A(node_2, node_1, *)$, satisfying $T_A(N, B, C)$ requires that $T_A(N, B, *)$ be true. Thus we have the following lemma:

**Lemma 4.1**: Suppose that trust specifications are of the form $T_A(node_2, node_1, *)$ and that the channel composition order is iterative. In any name space, all the descendents of each node trust the node.
Considering the same figure (Figure 4.12), let $M$ be any non-descendent of $B$ and $A$ be any child of $B$. By Theorem 4.1, $T_A(M, B, A)$ is true. Since trust specifications are of the form $T_A(node_2, node_1, *)$, satisfying $T_A(M, B, A)$ requires that $T_A(M, B, *)$ be true. Thus we have the following lemma:

**Lemma 4.2:** Suppose that trust specifications are of the form $T_A(node_2, node_1, *)$ and that the channel composition order is iterative. In any name space, all the non-descendents of each node trust the node.

Lemmas 4.1 and 4.2, together with the observation that every node is either a descendent or a non-descendent of a node, yield the following theorem:

**Theorem 4.3:** If trust relationships are of the form $T_A(node_2, node_1, *)$ and channel composition is iterative, all the nodes in any name space are globally trusted.

Now consider the situation when trust specifications are of the form $T_A(*, node_1, node_2)$ and the channel composition is recursive. Let $B$ be any node, $A$ be any child of $B$ and $M$ be any non-descendent of $B$ (see Figure 4.13). By Theorem 4.2, $T_A(A, B, M)$ is true. Since trust specifications are all of the form $T_A(*, node_1, node_2)$, satisfying $T_A(A, B, M)$ requires that $T_A(*, B, M)$ be true. Thus, we have the following lemma:
Lemma 4.3: When trust specifications are of the form $T_A(*, node_1, node_2)$ and the channel composition is recursive, in any name space, each node is trusted for all its non-descendents.

Considering the same figure (Figure 4.13), let $C$ be the parent of $B$ and $N$ be any descendant of $B$. By Theorem 4.2, $T_A(C, B, N)$ is true. Since trust specifications are all of the form $T_A(*, node_1, node_2)$, satisfying $T_A(C, B, N)$ requires that $T_A(*, B, N)$ be true. Thus, we have the lemma:

**Lemma 4.4:** When trust specifications are of the form $T_A(*, node_1, node_2)$ and the channel composition is recursive, in any name space, each node is trusted for all its descendants.

Lemmas 4.3 and 4.4, together with the observation that every node is either a descendant or a non-descendent of a node, yield the following theorem:

**Theorem 4.4:** If trust relationships are of the form $T_A(*, node_1, node_2)$ and channel composition is recursive, all the nodes in any name space are globally trusted.

Since our primary goal is the elimination of global trust requirements, by Theorems 4.3 and 4.4, it is clear that the combinations $T_A(node_2, node_1, *)$ plus iterative channel composition and $T_A(*, node_1, node_2)$ plus recursive channel composition are not interesting.

Let us now consider the remaining two combinations. We model the combination $T_A(*, node_1, node_2)$ plus iterative channel composition by a directed graph called $IT$-graph,
in which there is an edge \textit{node}_1 node_2 if and only if \textit{node}_1 is trusted for \textit{node}_2, i.e., if $T_A(*, \textit{node}_1, \textit{node}_2)$ is true (see Figure 4.14(a)). In the name space, if $C$ is the parent of a node $B$, $M$ is any non-descendent of $B$, and $N$ is any descendent of $B$, by Theorem 4.1 $T_A(M, C, B)$ and $T_A(N, B, C)$ are true (see Figure 4.14(b)). If $B$ is a leaf, then $B$ does not have any descendent $N$ and only $T_A(M, C, B)$ is true. Since trust specifications are of the form $T_A(*, \textit{node}_1, \textit{node}_2)$, satisfying $T_A(M, C, B)$ and $T_A(N, B, C)$ requires that $T_A(*, C, B)$ and $T_A(*, B, C)$ be true. If $B$ is a leaf, only $T_A(*, C, B)$ need be true. This result is summarized in the following theorem:

**Theorem 4.5:** Suppose that the trust specifications are of the form $T_A(*, \textit{node}_1, \textit{node}_2)$, and that the channel composition is iterative. A name space satisfies a given set of trust specifications if and only if the following two conditions are satisfied:

1. If there is a link $C$-\textit{B} in the name space and \textit{B} is not a leaf, then $T_A(*, C, B)$ and $T_A(*, B, C)$ must be true, i.e., there must be edges $CB$ and $BC$ in the IT-graph for the trust specifications.

2. If there is a link $C$-\textit{B} in the name space and \textit{B} is a leaf, $T_A(*, C, B)$ must be true, i.e., there must be an edge $CB$ in the IT-graph for the trust specifications.

The fourth and the final combination is of the form $T_A(\textit{node}_2, \textit{node}_1, *)$ with recursive channel composition. We model this combination by a directed graph called \textit{RT-graph}, in which there is an edge \textit{node}_1 node_2 if and only if \textit{node}_2 trusts \textit{node}_1, i.e., $T_A(\textit{node}_2, \textit{node}_1, *)$ (see Figure 4.15(a)). In the name space, if $C$ is the parent of a node $B$, $M$ is any non-descendent of $B$, and $N$ is any descendent of $B$, by Theorem 4.2 $T_A(B, C, M)$ and $T_A(C, B, N)$ are true (see Figure 4.15(b)). If $B$ is a leaf, then $B$ does not have any descendent

---

**Figure 4.14:** IT-graph representation of trust relationships of the form $T_A(*, \textit{node}_1, \textit{node}_2)$ when channel composition is iterative. (a) There is a directed edge \textit{node}_1 node_2 in the IT-graph if and only if $T_A(*, \textit{node}_1, \textit{node}_2)$ is true. (b) If there is an independent channel $C$-\textit{B} in the name space and \textit{B} is not a leaf, then there must be edges $CB$ and $BC$ in the IT-graph.
and only $T_A(B, C, M)$ is true. Since trust specifications are of the form $T_A(node_2, node_1, *)$, satisfying $T_A(B, C, M)$ and $T_A(C, B, N)$ requires that $T_A(B, C, *)$ and $T_A(C, B, *)$ be true. If $B$ is a leaf, only $T_A(B, C, *)$ need be true. The following theorem summarizes these results:

**Theorem 4.6:** Suppose that the trust specifications are of the form $T_A(node_2, node_1, *)$, and that the channel composition is recursive. A name space satisfies a given set of trust specifications if and only if the following two conditions are satisfied:

1. If there is a link $C - B$ in the name space and $B$ is not a leaf, then $T_A(B, C, *)$ and $T_A(C, B, *)$ must be true, i.e., there must be edges $CB$ and $BC$ in the RT-graph for the trust specifications.

2. If there is a link $C - B$ in the name space and $B$ is a leaf, $T_A(B, C, *)$ must be true, i.e., there must be an edge $CB$ in the RT-graph for the trust specifications.

4.5.2. Duality

It may be observed that Theorems 4.5 and 4.6 are identical except for the fact that the last and the first arguments to the trust predicates are interchanged. In other words, the combination $T_A(node_2, node_1, *)$ coupled with recursive channel composition is a dual of $T_A(*, node_1, node_2)$ coupled with iterative channel composition, with the first trust argument replacing the role of the last trust argument. This is summarized in the following theorem:

**Theorem 4.7 (Duality Theorem):** The algorithms for synthesizing name spaces for the combination $T_A(*, node_1, node_2)$ plus iterative channel composition become the algorithms for synthesizing name spaces for the combination $T_A(node_2, node_1, *)$ plus recursive channel composition, if the last and first arguments of the trust specifications are interchanged. The converse

---

**Figure 4.15:** Trust relationships are of the form $T_A(node_2, node_1, *)$ and channel composition is recursive. (a) There is a directed edge $node_1 node_2$ in the RT-graph if and only if $T_A(node_2, node_1, *)$ is true. (b) If there is an independent channel $C - B$ in the name space, and $B$ is not a leaf, then there must be edges $CB$ and $BC$ in the RT-graph.
of this statement is also true.

A consequence of Theorem 4.7 is that developing name space synthesis algorithms only for the combination \( T_A(*, node_1, node_2) \) plus iterative channel composition is sufficient.

The arguments to trust specifications are agents, which represent organizations and individuals sharing a distributed system. These agents occur as nodes in a name space for the system. There are two kinds of nodes in a name space, namely, internal nodes and leaf nodes. The internal nodes serve as managers of databases of keys and have agents as their owners, and hence they are referred to as name servers. The leaf nodes represent the agents themselves. A single agent may own several name servers, but each agent is represented by a unique leaf node in a name space. Hence, in synthesizing name spaces, we assume that more than one internal node can correspond to the same agent, but one and only one leaf node must correspond to each agent, and every agent appears as a leaf node.

4.6. Name Space Synthesis Given Two-Agent Trusts and Iterative Composition

In this section we develop an algorithm for synthesizing name spaces for the combination \( T_A(*, node_1, node_2) \) plus iterative channel composition. The input to the algorithm is a set of trust specifications of the form \( T_A(*, node_1, node_2) \).

4.6.1. The Synthesis Algorithm

The first step of the algorithm is to synthesize the IT-graph for the given trust specifications (Figure 4.16-a). The IT-graph is then transformed into an undirected graph by replacing all bidirectional edges with undirected edges, and by deleting all unidirectional edges (Figure 4.16-b). The resulting undirected graph is converted into a spanning forest (Figure 4.16-c). If there is any isolated node in the spanning forest such that in the original IT-graph there is an edge towards it from another node, an undirected edge is added between these two nodes in the spanning forest (Figure 4.16-d).
Figure 4.16: Name space synthesis from trust functions of the form $T_A(\ast, node_1, node_2)$ and iterative channel composition. (a) A sample IT-graph. (b) Bidirectional edges replaced by undirected edges, and unidirectional edges are removed. (c) Transformation of the undirected graph to a spanning forest. (d) An isolated node such as $D$ is re-attached if there is an edge towards it in the IT-graph.
If the resulting spanning forest contains more than one tree, then no tree-structured secure name space is possible. On the other hand, if there is only one tree in the spanning forest, one of the nodes is selected as the root, and all the nodes in the tree are given their pathnames as names. The resulting name space satisfies the given trust specifications when iterative channel composition order is used. The exact algorithm is as follows:

Algorithm 4.1

construct IT-graph for the given trust specifications;
for (each edge \( AB \) in the IT-graph) do {
    if (there is no edge \( BA \)) then {
        delete \( AB \) from the IT-graph;
    } else {
        delete \( AB \) and \( BA \);
        add undirected edge \( AB \);
    } fi
} for (each connected component in IT-graph) do {
    transform to a spanning tree;
} for (each node \( A \) in the spanning forest) do {
    if (\( A \) is an isolated node) then {
        if (there is an edge \( NA \) in the original IT-graph) then {
            add an undirected edge \( NA \);
            mark \( A \) as a leaf;
        } fi
    } fi
} if (there is more than one tree in the forest) {
    print(no name space exists);
    return;
} else {
    select any node \( R \) that has not been marked as a leaf;
    mark \( R \) as the root of the tree;
    for (each node) do {
        label it with its complete pathname;
    }
}
return the constructed name space;

4.6.2. Correctness and Complexity of Algorithm 4.1

In the following theorem, we show the correctness and derive the computational complexity of Algorithm 4.1.

Theorem 4.8: If Algorithm 4.1 constructs a name space, the constructed name space satisfies the given set of trust specifications when iterative channel composition is used. If a name space
exists, Algorithm 4.1 constructs a tree-structured name space. The algorithm's worst-case time-complexity is $O(\text{max}(\text{number of trusts}, \text{number of agents}))$.

Proof: We first prove the first part of the theorem, i.e., that, if Algorithm 4.1 constructs a name space, the constructed name space satisfies the given set of trust specifications when iterative channel composition is used. Suppose the algorithm constructs a name space $NS$. Consider each edge $BC$ in $NS$. Let $C$ be the parent of $B$. If $B$ is not a leaf, $BC$ was an outcome of the second for-loop of Algorithm 4.1, hence $BC$ was an undirected edge in a connected component at the beginning of that for-loop, and therefore $BC$ was an undirected edge in a connected component at the end of the first for-loop. Thus, there are edges $BC$ and $CB$ in the IT-graph. If $B$ is a leaf, then it is an outcome of the third for-loop, and hence there is an edge $CB$ in the IT-graph. Applying Theorem 4.5, we obtain that $NS$ satisfies the given trust specifications. This completes the proof of the first part of the Theorem.

Suppose that a name space $NS$ exists for the given trust specifications. We now show that the algorithm will construct a name space. Applying Theorem 4.5, the IT-graph must contain at least (1) bidirectional edges corresponding to the links between two internal nodes in $NS$, and (2) unidirectional edges corresponding to links between an internal node and a leaf node in $NS$. Thus, at the end of the first for-loop of Algorithm 4.1, there will be a connected graph in which all the internal nodes of $NS$ are present, and at the end of the second for-loop, this connected graph will get transformed into a tree. All the nodes not in this tree will get added as leaves to the tree in the third for-loop. Thus, Algorithm 4.1 yields a tree-structured name space if one exists.

Let us now derive the worst-case time complexity of Algorithm 4.1. The number of edges is of the order of the number of trust relationships. Suppose that this number is $e$. Suppose that the number of nodes, which is the number of agents, is $n$. The complexity of the first for-loop is $O(e)$, the complexity of the second for-loop is the complexity of constructing a spanning tree, which is $O(\text{max}(e, n))$, and the complexity of the last for-loop is $O(n)$. Thus, the complexity of Algorithm 4.1 is $O(\text{max}(e, n))$.

This completes the proof of Theorem 4.8.

Algorithm 4.1 gives one possible name space satisfying the given trust relationships. There may be more than one name space satisfying the same given set of trust relationships. In a name space, each node has to store a database of encryption keys of all its children. In a name space in which the root is the parent of all other nodes (see Figure 4.17-a), the root has to store the entire database containing the keys of all other agents in the system, resulting in a centralized name server. At the other end of the spectrum is a name space corresponding to a hamiltonian path (Figure 4.17-b), in which each node has to store just one key (that of its sole child). Both of these extreme name space configurations are undesirable. Therefore, it is desirable to put bounds on the number of children of each node in a name space. However, the next section derives some NP-completeness results with respect to these bounded-children name space design problems [AHU74, GaJ79].

4.6.3. NP-Completeness Results

The next theorem shows that the problem of putting an upper bound on the number of children of each node in a name space is NP-complete. But before we prove the theorem, we need two graph-theoretic lemmas, which we prove next. The next lemma shows an equivalence between hamiltonian paths and bounded-children spanning trees in graphs.
Lemma 4.5: Let $G$ be a graph for which the existence of a hamiltonian path is to be determined. Suppose that we construct a graph $G'$ by adding new edges and new nodes as follows (see Figure 4.18): to each node $V$ in $G$, we add $u-2$ new edges, each edge connecting $V$ and a new node. A hamiltonian path for $G$ is a spanning tree for $G'$, and in the spanning tree, each node has at most $u$ children. The converse of this statement is also true.

Proof: Suppose that there is a hamiltonian path $H$ for $G$. Let the set of new nodes and the set of new edges added to form $G'$ be denoted by $N_{G'}$ and $E_{G'}$. When we add an edge of $E_{G'}$ to $H$, a new node gets added to $H$, and $H$ remains a tree (i.e., no cycles are created). Thus, when we add all the edges of $E_{G'}$ to $H$, we obtain a tree $T'$ for $G'$. In $T'$, each node has at most two edges of $H$ and at most $u-2$ edges of $E_{G'}$ incident on it. Thus, there is an upper bound of $u$ on the number of children of each node in $T'$.

Suppose that there is a spanning tree $T'$ for $G'$ with an upper bound of $u$ on the number of children of each node. Since each edge of $E_{G'}$ connects a new node of $N_{G'}$ to $T'$, and all nodes of $N_{G'}$ must be present in a spanning tree, all the edges of $E_{G'}$ must be present in $T'$. Each node of $N_{G'}$ is a terminal node in $T'$. Thus, when we remove the nodes of $N_{G'}$ from $T'$, we obtain a tree $T$ in which all the nodes except those of $N_{G'}$, i.e., all the nodes of $G$, are present. Hence, $T$ is a spanning tree for $G$. Each node of $T$ must have had $u-2$ more edges incident on it in $T'$ (since in $T'$, each of these nodes was connected to $u-2$ nodes of $N_{G'}$). But in $T'$, each node has at most $u$ edges incident on it. Thus, each node in $T$ can have at most 2 edges incident on it. A tree in which each node has at most 2 edges incident on it is a hamiltonian path. Hence, $T$ is a hamiltonian path for $G$. This completes the proof of Lemma 4.5.
The next lemma proves equivalence between any spanning tree of a graph and the IT-graph for the graph. This equivalence is used in both Theorem 4.8 and Theorem 4.9.

**Lemma 4.6**: Given a graph $G$, suppose that we construct an IT-graph $ITG$ by replacing each edge in $G$ with a bidirectional edge. Any name space synthesized for $ITG$ is a spanning tree $^2$ of $G$, and any spanning tree for $G$ is a name space for $ITG$.

**Proof**: Using Theorem 4.5 and the fact that $ITG$ contains only bidirectional edges, we obtain that for each edge in $NS$ there is a bidirectional edge in $ITG$. Since for each bidirectional edge in $ITG$ there is an undirected edge in $G$, for each edge in $NS$ there is an edge in $G$. Thus, $NS$ is a subgraph of $G$.

Consider any spanning tree $T$ for $G$. For each edge in $T$, there is a bidirectional edge in $ITG$, and hence, by Theorem 4.5, $T$ is a name space for $ITG$. Thus, any spanning tree for $G$ is a name space for $ITG$. This completes the proof of Lemma 4.6.

---

---

**Figure 4.18**: Construction of a graph $G'$ from a graph $G$ in Lemma 4.5 when $u=5$. (a) A sample graph $G$. (b) $G'$ obtained from $G$ by adding $u-2 = 3$ new edges to each node $V$ in $G$, each new edge connecting $V$ and a new node.

$^2$When we say a name space is a spanning tree for a graph $G$, we mean that all the nodes of $G$ are present in the name space and there are no cycles in the name space.
Theorem 4.9: For the combination $T_A(*, \text{node}_1, \text{node}_2)$ plus iterative channel composition, synthesizing a name space with a given upper bound on the number of children of each node is NP-complete.

Proof: To prove that the problem is in NP, we first show that guessing a solution and checking the validity of the guessed solution has polynomial time complexity [AHU74, GaJ79]. Let the number of agents be $V$, and the upper bound be $u$. To build a name space, we have to enumerate $V-1$ edges, and designate one of the nodes as the root. Checking the validity of the guessed name space consists of the following steps:

1. The name space must be connected and tree-structured (i.e., there must not be any cycles in the name space).
2. The name space must satisfy the given trust specifications. Thus for each edge between internal nodes in the name space, there must be bidirectional edges in the IT-graph for the given trust specifications. For each edge between an internal node and a leaf node in the name space, there must be an edge from the internal node to the leaf node in the IT-graph. In all, there are $V-1$ edges in the name space.
3. The number of children of each node must not exceed $u$.

Each of these steps can be carried out in $O(V)$ time. Thus the problem is in NP.

It is known that determining the existence of a hamiltonian path in a graph is an NP-complete problem [AHU74, GaJ79]. We reduce this NP-complete problem to our problem of putting an upper bound on the number of children of each node in a name space. To achieve this reduction, given a graph $G$ for which the existence of a hamiltonian path is to be determined, we construct a graph $G'$ as follows (see Figure 4.18): to each node $V$ in $G$, we add $u-2$ new edges, each edge connecting $V$ and a new node. By Lemma 4.5, to determine the existence of a hamiltonian path for $G$, it suffices if we determine the existence of a spanning tree for $G'$ with each node having at most $u$ children.

We now construct an IT-graph $ITG'$ by replacing each edge in $G'$ with a bidirectional edge. By Lemma 4.6, a spanning tree with an upper bound of $u$ on the children of each node exists for $G'$ if and only if there is a name space for $ITG'$ with the same upper bound. Constructing $G'$ from $G$ is an $O(n \times u)$ operation, and if $e$ is the number of edges in $G$, constructing $ITG'$ from $G'$ is an $O(e)$ operation. However, the upper bound $u$ cannot exceed the number of nodes $n$, and the number of edges cannot exceed the square of the number of nodes. Therefore, the complexity of the two constructions together is $O(n^2)$. Thus we have reduced the problem of determining the existence of a hamiltonian path for $G$ to the bounded children name space problem in polynomial time. Since the hamiltonian path problem is NP-complete, the bounded children name space problem is NP-complete.

Now consider the other extreme case (Figure 4.17(b)). We consider two approaches to avoiding such an extreme case. In the first approach, we assume that an agent can occur either as a leaf node or as an internal node in a name space, and try to put a lower bound on the number of leaves in a name space. In the second approach, we try to put a lower bound on the number of children of each node in a name space.

Theorem 4.10: Consider the combination $T_A(*, \text{node}_1, \text{node}_2)$ plus iterative channel composition. If an agent can occur either as a leaf node or as an internal node in a name space, synthesizing a name space in which there is a given lower bound on the number of leaf nodes is NP-complete.
Proof: Proving that the problem is in NP is identical to the proof we gave in Theorem 4.9, except that, while checking the validity of a guessed name space, we have to check for the minimum, rather than the maximum, number of children of a node in the name space.

It is known that the following problem is an NP-complete problem [AHU74, GaJ79]: Given a graph $G$ and a positive integer $K < n$ (where $n$ is as usual the number of vertices in the graph), does the graph have a spanning tree such that the number of leaves in the tree is at least $K$? A straight-forward application of Lemma 4.6 yields the reduction from this NP-complete problem to the bounded leaf name space problem.

Now consider the problem of putting a lower bound on the number of children of each node in a name space. The following algorithm gives a reduction from the bounded leaf problem to bounded children name space problem. Let the given graph in the bounded leaf problem be $G$ having $n$ vertices.

Algorithm 4.2

for (each set $vert$ of $n-K$ vertices in the graph $G$) do {
    construct a graph $G'$ from $G$ by:
        coalescing the subgraph formed by $vert$ into a single node $x$;
    construct an IT-graph $ITG'$ by:
        replacing each edge in $G'$ by a bidirectional edge;
    if ($ITG'$ has a name space with children of each node bounded below by $K$) {
        print($G$ has a spanning tree with number of leaves bounded below by $K$);
        return (success);
    }
    print("$G$ does not have a spanning tree with number of leaves $\geq K");
    return (failure);
}

Figure 4.19 illustrates the coalescing step of the algorithm. The subgraph formed by the vertices B, C1 and C2 in the graph of Figure 4.19-a are coalesced to form the graph of Figure 4.19-b. To coalesce the subgraph $S$ formed by a set of vertices $verts$ in a graph $G$, we carry out the following steps:

1. A single new node $x$ replaces the nodes in the set $verts$.
2. All edges between nodes within $verts$ are deleted.
3. Each edge between a node $w$ outside $verts$ and a node within $verts$ is replaced by an edge between $w$ and $x$.

The subgraph $S$ consists of the vertices in $verts$ and the edges between nodes within $verts$.
All other nodes and edges are retained.

We now show that Algorithm 4.2 returns success if and only if there is a solution for the bounded leaf problem. Suppose that Algorithm 4.2 gives an affirmative answer in the if-step. By Lemma 4.5, a name space for $ITG'$ is a spanning tree for $G'$. Thus, $G'$ has a spanning tree $T_{G'}$ with the number of children of each node at least $K$. Since the number of leaves in a graph is no less than the smallest number of children of a node in a tree, the number of leaves of $T_{G'}$ is at least $K$. To obtain a spanning tree $T_G$ for the original graph $G$, we de-coalesce the coalesced node $n$ in $T_{G'}$ into its original subgraph $S$, and transform the subgraph $S$ into a spanning subtree. All the leaves except possibly $x$ of $T_{G'}$ remain as leaves of $T_G$. If $x$ is a leaf in $T_{G'}$, when $x$ is de-coalesced to obtain $T_G$, $x$ fans-out into at least one leaf. Thus $T_G$ has a number of leaves at least equal to that of $T_{G'}$, and hence $T_G$ has its number of leaves at least $K$. Thus, if Algorithm 4.2 gives an affirmative answer, $G$ has a tree in which the number of leaves is at least $K$. Such a tree is a solution for the bounded leaf problem. Thus, if Algorithm 4.2 gives an affirmative answer, there is a solution for the bounded leaf problem.

We now show that, if there is a solution to the bounded leaf problem, Algorithm 4.2 gives an affirmative answer. Suppose that there is a solution to the bounded leaf problem. Thus, there is a spanning tree $T_G$ for $G$ whose number of leaves is at least $K$. If we coalesce the interior nodes of tree $T_G$, we obtain a tree $T_{G'}$ which has only one internal node, and all the leaves of $T_G$ are children of this internal node. Thus, the number of children of the only node in $T_{G'}$ is at least $K$. By Lemma 4.5, $T_{G'}$ is also a name space for $ITG'$. Thus, Algorithm 4.2, when it coalesces the nodes of $G$ that occur as interior nodes of $T_G$, will yield an affirmative answer.

In summary, Algorithm 4.2 yields an affirmative answer if and only if the bounded leaf problem has a solution. Thus, Algorithm 4.2 is a reduction from the bounded leaf problem to our bounded children name space problem.

The complexity of Algorithm 4.2 is as follows. There are $C(n, n-K) = O(n^K)$ possibilities for a set of $n-K$ vertices. Coalescing $G$ to $G'$ involves looking at each of the vertices and

Figure 4.19: Reduction of "bounded leaf" problem to "bounded children name space design" problem by coalescing internal nodes. The internal nodes $B$, $C_1$ and $C_2$ in the graph shown in (a) are coalesced to get the graph shown in (b).
edges in $G$, and hence is of complexity $O(\max(n,e))$. Transforming $G'$ to $ITG'$ is an $O(e)$ operation. Thus, the complexity of Algorithm 4.2 is of the order of,

$$n^k \times (\max(n,e) + \text{complexity of bounded children name space problem}).$$

4.7. 3-Agent Trust Specifications

In this section we develop algorithms for synthesizing name spaces given 3-agent trust specifications. Directed graphs are again used to represent the trust specifications. However, a directed edge by itself can only represent a binary relationship between its two vertices. Thus, in representing 3-agent trust relationships by a directed graph, two of the agents in the trust relationship become vertices, and the third agent becomes a label of the directed edge between the vertices. Thus, a labeled directed edge is used to represent a 3-agent trust relationship.

4.7.1. Iterative and Recursive Composition

As explained at the beginning of this chapter, a name space is specific to a channel composition order. When the channel composition order is iterative, we model the given trust relationships by a labeled directed graph called LIT-graph. A LIT-graph contains an edge $BC$ labeled with a set of agents $set_{BC}$ if and only if for all agents $A$ in $set_{BC}$ we have $T_A(A,B,C) = \text{true}$ in the given set of trust specifications (see Figure 4.20). Consider a name space that satisfies the given set of trust specifications. If $B$ is any node in the name space, and $C$ is its parent, the link $B-C$ divides the set of all agents in the name space into two disjoint subsets containing, (1) the descendents of $B$, e.g., $N$, and (2) the non-descendents of $C$, e.g., $M$, respectively. By Theorem 4.1, for all descendents $N$ of $B$, $T_A(N,B,C) = \text{true}$, and hence $N$ belongs to $set_{BC}$, and for all non-descendents $M$ of $C$, $T_A(M,C,B) = \text{true}$, and hence $M$ belongs to $set_{CB}$. Since every node except $B$ and $C$ is either a descendent of $B$ or a non-descendent of $C$, and since $B$ and $C$ trivially belong to $set_{BC}$ and $set_{CB}$, $set_{BC} \cup set_{CB} = \text{the universal set}^5$.

Thus, if in the LIT-graph for a given set of trust specifications $set_{BC} \cup set_{CB}$ does not equal the universal set, then $C$ cannot be the parent of $B$.

Interchanging the roles of $C$ and $B$, we obtain that, if $set_{CB} \cup set_{BC}$ does not equal the universal set, $B$ cannot be the parent of $C$.

But, since $set_{BC} \cup set_{CB} = set_{CB} \cup set_{BC}$, if $set_{CB} \cup set_{BC}$ does not equal the universal set, the link $B-C$ cannot exist in any name space synthesized for the given trust specifications, and hence the edges $BC$ and $CB$ can be removed from the LIT-graph. These results are summarized in the following theorem:

**Theorem 4.11:** Given 3-agent trust specifications, if iterative channel composition is used, a link $B-C$ can exist in the name space only if, in the LIT-graph constructed from the trust specifications, the union of the labels of edges $BC$ and $CB$ equals the universal set. Moreover, if $C$ is the parent of $B$, an agent $N$ that is a descendent of $B$ in the name space must belong to $set_{BC}$, and an agent $M$ that is a non-descendent of $B$ in the name space must belong to $set_{CB}$.

---

4 *LIT-graph* stands for *Labeled Iterative Trust graph*.

5 The symbol "\( \cup \)" is used to mean the union of sets.
When the channel composition order is recursive, we model the given trust relationships by a labeled directed graph called LRT-graph. A LRT-graph contains an edge $BC$ labeled with a set of agents $set_{BC}$ if and only if, for all agents $A$ in $set_{BC}$, $T_A(C, B, A) = true$ in the given set of trust specifications (see Figure 4.21). Consider a name space that satisfies the given set of trust specifications. If $B$ is any node in the name space, and $C$ its parent, the link $B - C$ divides the set of all agents in the name space into two disjoint subsets containing, (1) the descendents of $B$, e.g., $N$, and (2) the non-descendents of $B$, e.g., $M$, respectively. By Theorem 4.2, for all descendents $N$ of $B$, we have $T_A(C, B, N) = true$, and hence $N$ belongs to $set_{BC}$; also, for all non-descendents $M$ of $B$, we have $T_A(B, C, M) = true$, and hence $M$ belongs to $set_{CB}$. Since every node is either a descendent or a non-descendent of a node, $set_{BC} \cup set_{CB} = the \ universal \ set$.\footnote{6}

Thus, if, in the LRT-graph for a given set of trust specifications, $set_{BC} \cup set_{CB}$ does not equal the universal set, $C$ cannot be the parent of $B$.

Interchanging the roles of $C$ and $B$, we obtain that, if $set_{CB} \cup set_{BC}$ does not equal the universal set, $B$ cannot be the parent of $C$.

But, since $set_{BC} \cup set_{CB} = set_{CB} \cup set_{BC}$, if $set_{CB} \cup set_{BC}$ does not equal the universal set, the link $B - C$ cannot exist in any name space synthesized for the given trust specifications, and hence the edges $BC$ and $CB$ can be removed from the LRT-graph. These results are summarized in the following theorem:

**Theorem 4.12:** Given 3-agent trust specifications, if recursive channel composition is used, a link $B - C$ can exist in the name space only if, in the LRT-graph constructed from the trust specifications, the union of the labels of edges $BC$ and $CB$ equals the universal set. Moreover, if $C$ is the parent of $B$, an agent $N$ that is a descendent of $B$ in the name space must belong to

---

**Figure 4.20:** LIT-graph representation for the combination consisting of 3-agent trust specifications plus iterative channel composition. (a) The LIT-graph contains an edge $BC$ labeled with $set_{BC}$, where $set_{BC}$ is the set of all agents $A$ such that $T_A(A, B, C)$ is true. (b) The union of $set_{BC}$ and $set_{CB}$ must be the universal set.
set_{BC}, and an agent M that is a non-descendent of B in the name space must belong to set_{CB}.

4.7.2. Duality

It may be observed that Theorems 4.11 and 4.12 are identical except that the sets in Theorem 4.11 are the labels in a LIT-graph whereas the sets in Theorem 4.12 are the labels in a LRT-graph. The difference in the computations of labels in a LIT-graph and those in a LRT-graph is that, in a LIT-graph, a label set_{BC} contains the agents A such that \( T_A(A, B, C) = true \). In a LRT-graph, a label set_{BC} contains the agents A such that \( T_A(C, B, A) = true \). Thus, the conditions for synthesizing name spaces for the cases of iterative and recursive channel composition are identical except for the fact that the last and the first arguments to the trust predicates are interchanged. These results are summarized in the following theorem:

Theorem 4.13 (General Duality Theorem): Any algorithm that synthesizes name spaces from LIT-graphs is also an algorithm for synthesizing name spaces from LRT-graphs, and vice versa. Any algorithm that synthesizes name spaces for 3-agent trust specifications combined with iterative channel composition becomes an algorithm for synthesizing name spaces for 3-agent trust specifications combined with recursive channel composition if the roles of the first and the last arguments to the trust specifications are interchanged, and vice versa.

Figure 4.21: LRT-graph representation for 3-agent trust specifications combined with recursive channel composition. (a) The LRT-graph contains an edge BC labeled with set_{BC}, where set_{BC} is the set of all agents A such that \( T_A(C, B, A) \) is true. (b) The union of set_{BC} and set_{CB} must be the universal set.
As a consequence of Theorem 4.13, we can limit ourselves to developing name space synthesis algorithms only for 3-agent trust specifications plus iterative channel composition.

4.8. Name Space Synthesis Given Three-Agent Trusts and Iterative Composition

This section gives an algorithm for synthesizing name spaces for 3-agent trust specifications plus iterative channel composition. The algorithm takes a LIT-graph as input, and outputs a name space satisfying the trust specifications represented by the LIT-graph, if such a name space exists.

4.8.1. The Synthesis Algorithm

The algorithm employs a dynamic programming technique to synthesize the name space. Since the algorithm is quite involved, for ease of understanding, we will describe it in a bottom-up fashion with the help of an example.

Algorithm 4.3

Let us assume that there are $n$ agents $l_1, l_2, ..., l_n$ to be named in the name space. $l_1, l_2, ..., l_n$ must occur as leaf nodes in the name space. Let us assume that, out of these $n$ agents, there are $m$ agents $i_1, i_2, ..., i_m$ that can serve as name servers, i.e., as internal nodes in the name space (see Figure 4.22). Whether an agent can serve as a name server or not is the choice of the agent; hence $m$ is determined by the number of agents willing to serve as name servers.

There are at most $n$ steps in the algorithm. At each step there are $m$ trees, each tree rooted at a different internal node.

Step 1: In the first step, $m$ trees are constructed. Each tree has a different internal node as its root and as many leaf nodes as possible as the children of the root. A leaf node has no descendents, and all nodes are its non-descendents. By Theorem 4.11, a leaf node $l_g$ can become a child of an internal node $i_c$ if and only if the edge $CB$ in the input LIT-graph is labeled by the universal set (see Figure 4.23).

In the example, let $l_1, l_2$ and $l_3$ become the children of $i_1$, $l_4, l_5$ and $l_6$ become the children of $i_2$, $l_7, l_8$ and $l_9$ become the children of $i_3$, and $l_{10}, l_{11}$ and $l_{12}$ become the children of $i_4$ (see Figure 4.24).

This concludes step 1.

Let us assume we have carried out $k$ steps, at the end of which there are $m$ trees, $t_1^k, t_2^k, ..., t_m^k$ rooted at internal nodes $i_1, i_2, ..., i_m$, respectively (see Figure 4.25(a)). Let us further assume that the algorithm does not meet any of the termination conditions to be presented later.
Set \( CB \) must be the universal set

\[ \text{Figure 4.23: Node } l_B \text{ becomes a child of } i_C \text{ if and only if } set_{CB} \text{ is the universal set} \]

\[ \text{Figure 4.24: Trees at the end of step 1 of Algorithm 4.3 for the example} \]

in this section. (For clarity and ease of understanding, the termination conditions are presented at the end of the description of the \( k+1 \)th step.) The \( (k+1) \)th step of the algorithm is carried out as follows.

**Step k+1:** We construct \( m \) trees \( t_1^{k+1}, t_2^{k+1}, \ldots, t_m^{k+1} \) in \( m \) substeps \( \text{substep}(t_1^{k+1}), \text{substep}(t_2^{k+1}), \ldots, \text{substep}(t_m^{k+1}) \), respectively (see Figure 4.25(b)). Each of these substeps starts from the end of step \( k \), and hence they can all be carried out in parallel. We will describe \( \text{substep}(t_1^{k+1}) \), which consists of constructing \( t_1^{k+1} \); the other substeps are very similar.

**Substep(\( t_1^{k+1} \)):** Tree \( t_1^{(k+1)} \) rooted at \( i_1 \) is constructed by attaching each of \( t_2^k, t_3^k, \ldots, t_m^k \) as a subtree of \( t_1^k \), directly under \( t_1^k \)'s root \( i_1 \). Effectively, we are proposing the \( m-1 \) new independent channels \( i_1-i_2, i_1-i_3, \ldots, i_1-i_m \) (see Figure 4.25(b-1)). The procedure for attaching \( t_2^k \) to \( t_1^k \) as a subtree is called \( attachment(t_1^k, t_2^k) \), and is described next. The procedures \( attachment(t_1^k, t_3^k), \ldots, attachment(t_1^k, t_m^k) \) for attaching \( t_3^k, \ldots, t_m^k \) respectively to \( t_1^k \) are very similar, and can be obtained by modifying in the obvious way the one to be described.

**Attachment(\( t_1^k, t_2^k \)):** When \( t_2^k \) is attached as a subtree of \( t_1^k \), root \( i_2 \) of \( t_2^k \) becomes a child of root \( i_1 \) of \( t_1^k \). The nodes in \( t_2^k \) and \( t_1^k \) must now satisfy the trust relationships represented by the input LIT-graph. The following tests check if the nodes satisfy the required trust relationships, and eliminate the nodes that do not (see Figure 4.26):

\( \text{(T1): Each node in } t_2^k \text{ is a descendent of } i_2. \) By Theorem 4.11, each node in \( t_2^k \) must belong to \( set_{21} \) in the input LIT-graph. Nodes that do not satisfy this condition must be
Figure 4.25: Illustration of step \( k+1 \) of the name space design algorithm. (a) Trees at the end of step \( k \). (b) Step \( k+1 \) consists of substeps \( \text{substep} (t_1^{k+1}), \text{substep} (t_2^{k+1}), \ldots, \text{substep} (t_m^{k+1}) \). Each of these substeps starts from (a), and hence they can all be carried out in parallel. (b-1) Substep \( (t_1^{k+1}) \), consisting of attachments \( \text{attachment} (t_1^k, t_2^k), \ldots, \text{attachment} (t_1^k, t_m^k) \) that attach all other trees to \( t_1^k \). (b-2) Substep \( (t_2^{k+1}) \), consisting of attachments \( \text{attachment} (t_2^k, t_1^k), \ldots, \text{attachment} (t_2^k, t_m^k) \) that attach all other trees to \( t_2^k \). (b-m) The deletion of a node from a tree is described by Algorithm 4.4 in the
next section, and may not always be possible. Thus, if a node in $t_2^k$ that does not belong to set$_{21}$ cannot be deleted from $t_2^k$, $t_2^k$ cannot be attached as a subtree of $t_1^k$, tests T2 and T3 are skipped, and attachment $(t_1^k, t_2^k)$ terminates.

(T2): After test T1, each node that is not in $t_2^k$ will be a non-descendent of $i_2$ in the final tree, and, by Theorem 4.11, such a node must belong to set$_{12}$. If this is not satisfied, $t_2^k$ cannot be attached as a subtree of $t_1^k$, test T3 is skipped, and attachment $(t_1^k, t_2^k)$ terminates.

(T3): After tests T1 and T2, if a leaf node $l_D$ occurs in both $t_1^k$ and $t_2^k$, one of the two duplicates of $l_D$ must be eliminated (see Figure 4.26). If $l_D \in$ set$_{21}$, deletion of $l_D$ from $t_1^k$ becomes permissible if the node deletion algorithm, Algorithm 4.4 returns success. If $l_D \in$ set$_{12}$, deletion of $l_D$ from $t_2^k$ becomes permissible if Algorithm 4.4 returns success. If deletion of $l_D$ from either $t_1^k$ or $t_2^k$ becomes permissible, the choice of the tree from which it is to be deleted can be made arbitrarily. That this choice will have no effect on subsequent attachments is shown in Section 4.8.3. If, on the other hand, the deletion of $l_D$ from either $t_1^k$ or $t_2^k$ is not permissible, $t_2^k$ cannot be attached as a subtree of $t_1^k$, and attachment $(t_1^k, t_2^k)$ terminates.

After attachment $(t_1^k, t_2^k)$, the succeeding attachments are, in order:

attachment$(t_1^k, t_3^k)$, ..., attachment$(t_1^k, t_m^k)$.

The sequence:

{attachment$(t_1^k, t_2^k)$, ..., attachment$(t_1^k, t_m^k)$}

forms substep$(t_1^{k+1})$ (see subfigure b-1 of Figure 4.25).

The sequence:

\[\text{attachment}(t_1^k, t_2^k),...,\text{attachment}(t_1^k, t_m^k)\]

forms substep$(t_1^{k+1})$ (see subfigure b-1 of Figure 4.25).
forms step $k+1$ (see Figure 4.25).

The algorithm terminates at the end of the $k+1^{th}$ step if any of the following conditions are satisfied:

1. At least one of $t_1^{k+1}, t_2^{k+1}, ..., t_m^{k+1}$ contains all the leaf nodes $l_1, l_2, ..., l_n$. One of $t_1^{k+1}, t_2^{k+1}, ..., t_m^{k+1}$ that has all the leaf nodes is output as the name space.

2. $t_1^{k+1}, t_2^{k+1}, ..., t_m^{k+1}$ are identical to $t_1^k, t_2^k, ..., t_m^k$ respectively. In this case, further steps will not yield any new trees, and no name space exists.

This concludes step $k+1$ of Algorithm 4.3. If none of the termination conditions are satisfied, the algorithm proceeds to step $k+2$, without regard for the value of $k$. However, in Theorem 4.16, we will show that there can be at most $m$ steps in the Algorithm.

4.8.2. Leaf Node Deletion Algorithm

Nodes that do not satisfy tests T1, T2 or T3 during an attachment must be deleted from their respective trees. In this section, we present the algorithm for deleting a node from a tree. Note that the deletion of a node from a tree is not always possible.

Algorithm 4.4

Consider an attachment such as attachment $(t^k_1, t^k_2)$, in which a leaf node such as $l_D$ is to be deleted from $t^k_2$ (see Figure 4.26). When $l_D$ is in $t^k_2$, $l_D$ is a descendent of all the nodes that are in the path from the root $i_2$ of $t^k_2$ to $l_D$ and a non-descendent of all other nodes in $t^k_2$. For each link $C \rightarrow B$ in the path from $i_2$ to $l_D$, $l_D$ is a descendent of $B$, and hence, by Theorem 4.11, $l_D \in \text{set}_{BC}$. When $l_D$ is deleted from $t^k_2$, $l_D$ becomes a non-descendent of all nodes in $t^k_2$; $l_D$ is then a non-descendent of $B$ (see Figure 4.27), and hence by Theorem 4.11, $l_D \in \text{set}_{CB}$. Thus, it is permissible to delete $l_D$ from $t^k_2$ only if, for all links $C \rightarrow B$ in the path from the root $i_2$ to $l_D$, $l_D \in \text{set}_{CB}$.

This completes the description of Algorithm 4.4.

4.8.3. Independence Properties of Duplicate Elimination in Algorithm 4.3

Test T3 in an attachment performs deletion of duplicate nodes. If the deletion of a duplicate node from either of the trees involved in an attachment is permissible, a question may arise as to whether the choice will have any effect on subsequent attachments. Suppose that the choice does have an effect. In order to find a name space if one exists, Algorithm 4.3 would have to exhaust the entire choice space, and hence would have to backtrack if it fails to find a name space tree. Consequently, Algorithm 4.3 would become very cumbersome.
Figure 4.27: Effect of deleting node \( l_D \) from tree \( t_5^{k+1} \). For each link \( C \rightarrow B \) in the path from the root \( i_2 \) to \( l_D \), (a) prior to deletion when \( l_D \) is in \( t_5^{k+1} \), \( l_D \) is a descendent of \( i_B \), and (b) after deletion, \( l_D \) is a non-descendent of \( i_B \).

Fortunately, the following theorem proves that the choice of the duplicate for deletion will have no effect whatsoever on subsequent attachments, and thus Algorithm 4.3 does not have to backtrack.

Theorem 4.14: Suppose that in an attachment, say attachment \( (t^k_f, t^k_g) \), \( l_D \) occurs as a duplicate, and the deletion of \( l_D \) from either \( t^k_f \) or \( t^k_g \) is permissible, choosing either \( t^k_f \) or \( t^k_g \) for deleting \( l_D \) will have no effect on subsequent attachments and hence the choice between them can be arbitrary.

Proof: To maintain uniformity of description with Algorithms 4.3 and 4.4, we will consider attachment \( _1 = \text{attachment}(t^k_f, t^k_g) \) for the purposes of the proof. The same proof holds for any other attachment.

Let choice 1 denote the deletion of \( l_D \) from \( t^k_f \), and choice 2 denote the deletion of \( l_D \) from \( t^k_g \). Choice 1 requires that the deletion of \( l_D \) from \( t^k_f \) be permissible in Algorithm 4.4. Hence choice 1 requires that the following condition be satisfied (see Figure 4.28):

(S1): For each link \( F \rightarrow E \) in the path from \( i_1 \) to \( l_D \), \( l_D \in \text{set}_{FE} \)

Choice 2 requires that the deletion of \( l_D \) from \( t^k_g \) be permissible in Algorithm 4.4. Hence choice 2 requires that the following condition be satisfied (see Figure 4.29):

(S2): For each link \( C \rightarrow B \) in the path from \( i_2 \) to \( l_D \), \( l_D \in \text{set}_{CB} \)

Consider the subsequent attachment, namely, attachment \( _2 = \text{attachment}(t^{k+1}_f, t^{k+1}_g) \). Notice that, if \( l_D \) does not occur in \( t^{k+1}_g \), the deletion of \( l_D \) in attachment \( _1 \) has no effect on attachment \( _2 \). Hence making choice 1 or choice 2 has no effect on attachment \( _2 \).

Suppose that \( l_D \) does occur in \( t^{k+1}_g \). \( l_D \) must be deleted during attachment \( _2 \). The only effect that choice 1 or choice 2 could possibly have on attachment \( _2 \) is on the deletion of \( l_D \) during attachment \( _2 \). There are two possibilities for the deletion of \( l_D \) during attachment \( _2 \):
Figure 4.28: *Choice 1*: During attachment of $t_2^k$ to $t_1^k$, duplicate $l_D$ is deleted from $t_2^k$.

Figure 4.29: *Choice 2*: During attachment of $t_2^k$ to $t_1^k$, duplicate $l_D$ is deleted from $t_1^k$.

(P1): Deletion of $l_D$ from $t_3^k$, or
(P2): Deletion of \( l_D \) from \( t_2^k \) or \( t_1^k \) depending on whether \( attachment_2 \) was preceded by \( choice_1 \) or \( choice_2 \), respectively.

Consider the possibility P1. If \( choice_1 \) was taken during \( attachment_1 \), deletion of \( l_D \) from \( t_3^k \) during \( attachment_2 \) is permissible in Algorithm 4.4 if (see Figure 4.30):

(S3): For each link \( H - G \) in the path from \( i_3 \) to \( l_D \), \( l_D \in \text{set}_{HG} \)

If \( choice_2 \) had been taken during \( attachment_1 \), deletion of \( l_D \) from \( t_3^k \) during \( attachment_2 \) is permissible in Algorithm 4.4 if (see Figure 4.31):

(S4): For each link \( H - G \) in the path from \( i_3 \) to \( l_D \), \( l_D \in \text{set}_{HG} \)

Conditions S3 and S4 are identical. Thus, for possibility P1 during \( attachment_2 \), making \( choice_1 \) or \( choice_2 \) during \( attachment_1 \) has no effect on \( attachment_2 \).

Now consider possibility P2. Suppose that \( choice_1 \) was taken during \( attachment_1 \). Deletion of \( l_D \) from \( t_2^k \) during \( attachment_2 \) is permissible in Algorithm 4.4 if (see Figure 4.30):

(S5): For each link \( C - B \) in the path from \( i_2 \) to \( l_D \), \( l_D \in \text{set}_{CB} \)

Suppose that \( choice_2 \) was taken during \( attachment_1 \). Deletion of \( l_D \) from \( t_1^k \) during \( attachment_2 \) is permissible in Algorithm 4.4 if (see Figure 4.31):

(S6): For each link \( F - E \) in the path from \( i_1 \) to \( l_D \), \( l_D \in \text{set}_{FE} \)

There is a difference in conditions S5 and S6. Suppose that S5 is satisfied. \( Choice_1 \) must have been taken during \( attachment_1 \), and hence S1 must be satisfied. But S1 is identical to S6, and hence S6 is satisfied. Thus, \( S5 \Rightarrow S6 \).

Suppose that S6 is satisfied. \( Choice_2 \) must have been taken during \( attachment_1 \), and

---

Figure 4.30: Attachment_2 after \( choice_1 \): \( l_D \) occurs as duplicates in \( t_2^k \) and \( t_3^k \).
hence S2 must be satisfied. But S2 is identical to S5, and hence S5 is satisfied. Thus, S6 => S5.

In conclusion, S5 <=> S6. Hence, making \textit{choice}_1 or \textit{choice}_2 does not have any effect in the case of possibility P2.

This completes the proof that making \textit{choice}_1 or \textit{choice}_2 has no effect on subsequent attachments, and hence can be arbitrary. Thus, in any attachment, if the deletion of either of the duplicates is possible, the choice of the duplicate to be deleted can be arbitrary.

\[\square\]

4.8.4. Correctness of Algorithm 4.3

The following theorem proves the correctness of Algorithm 4.3.

\textbf{Theorem 4.15}: If the channel composition order is iterative, a name space synthesized by Algorithm 4.3 satisfies the given trust specifications, and Algorithm 4.3 synthesizes a name space if one exists when node deletions are independent.

\textbf{Proof}: The proof is by induction on the number of steps in the algorithm. The induction hypothesis is as follows:

\textbf{Induction Hypothesis}: At step \( k \), for all \( x \) from 1 through \( m \), \( t^k_x \) satisfies the given trust relationships, and \( t^k_x \) contains all the leaf nodes that can possibly be at a distance \( k \) or less from its root.

While deriving the complexity of Algorithm 4.3 in Theorem 4.16, we show that, if the number of internal nodes is \( m \), there can be at most \( m \) steps in the Algorithm. Thus, when \( k = m \), the induction hypothesis becomes the theorem itself. Hence, to prove the theorem, it suffices to prove the induction hypothesis.
Base step: Let \( k = 1 \) in the induction hypothesis. For each \( x \) and \( y \), Algorithm 4.3 adds a leaf node \( l_x \) as a child of \( i_x \) only if \( \text{set}_x \) is the universal set. Since \( l_y \) is a leaf node and all other nodes are its non-descendents, by Theorem 4.11 the link \( i_x - l_y \) satisfies the given trust relationships. As this is true for all \( x \) and \( y \), all the trees satisfy the given trust relationships. Since Algorithm 4.3 tries to add each leaf node to each internal node, each tree contains all the leaf nodes that can possibly be at a distance 1 from the root of the tree. This completes the proof of the base step.

Induction step: Let us assume that the induction hypothesis is true for \( k \). We have to show that the hypothesis is true for \( k+1 \). We will give the proof for \( x = 1 \) in the hypothesis. The same proof holds for all other values of \( x \).

Consider \( t_1^k \), the tree rooted at \( i_1 \) at the end of step \( k \). In each of the attachments \( (t_1^k, t_2^k), (t_1^k, t_3^k), \ldots, (t_1^k, t_m^k) \), tests T1 and T2 are direct applications of Theorem 4.11. Hence, they eliminate all the nodes that do not satisfy the required trust relationships, and \( t_1^{k+1} \) satisfies the given trust relationships. This proves the first part of the induction hypothesis.

Since the induction hypothesis is assumed to be true for \( k \), \( t_1^k \) contains all the leaf nodes that can possibly be at a distance not exceeding \( k \) from \( i_1 \). Now consider a leaf node \( l_x \) that can possibly be at a distance not exceeding \( k+1 \) in a tree rooted at \( i_1 \). \( l_x \) has to be either at a distance not exceeding \( k \) or at a distance of \( k+1 \). If \( l_x \) is at a distance not exceeding \( k \), it must be present in \( t_1^k \), and hence it will be present in \( t_1^{k+1} \). If \( l_x \) is at a distance of \( k+1 \) from \( i_1 \), it must be at a distance \( k \) from some child \( i_2 \) of \( i_1 \). By the induction hypothesis the tree \( t_2^k \) at step \( k \) contains all nodes that can possibly be at a distance not exceeding \( k \), and hence contains \( l_x \). During attachment\( (t_1^k, t_2^k) \) of step \( k+1 \), Algorithm 4.3 attaches \( t_2^k \) as a subtree of \( t_1^k \), and, when node deletions are independent, \( l_x \) becomes part of \( t_1^{k+1} \) at a distance of \( k+1 \) from the root. Thus, \( t_1^{k+1} \) contains all the leaf nodes that can possibly be at a distance \( k+1 \) or less from \( i_1 \). This completes the proof of the induction step and of Theorem 4.15.

In a name space, the path from the root to a leaf node cannot contain duplicate internal nodes. This is because, if a path contains duplicate internal nodes, the part of the path between the duplicate nodes together with one of the duplicate nodes can be removed from the path. For example, suppose that a path contains the sequence \( i_1, i_2, \ldots, i_z \), and \( i_1 \), in which there is a duplication of \( i_1 \). The nodes \( i_1, i_2, \ldots, i_z \) can be deleted from the path, eliminating the duplication. Thus, in a name space, the path from the root to any leaf can contain at most all the \( m \) internal nodes (including the root). Therefore, each leaf node must be at a distance \( m \) or less from the root. But notice that, when \( k = m \), the induction hypothesis that we proved above becomes the theorem itself, and hence each tree at step \( m \) contains all the leaf nodes that can possibly be at a distance \( m \) or less from the root. Thus, if a name space exists, there will be a tree containing all agents as leaf nodes at the end of step \( m \) of Algorithm 4.3. Hence, Algorithm 4.3 synthesizes a name space if one exists.

This completes the proof of Theorem 4.15.

\[ \square \]

4.8.5. Complexity of Algorithm 4.3

At step \( S \) of Algorithm 4.3, to each tree all the nodes that can possibly be at distance not exceeding \( S \) from the root of the tree are added to the tree. Since there are at most \( m \) internal nodes, a leaf node can at most be at a distance of \( m \) from the root of the name space. Thus, Algorithm 4.3 synthesizes a name space if one exists in at most \( m \) steps, and hence there are at
most $m$ steps in Algorithm 4.3.

Each step of the algorithm consists of $m$ substeps. Each substep consists of $m-1$ attachments. Thus, at each step there are $O(m^2)$ attachments, and there are at most $O(m^3)$ attachments in the algorithm.

In each attachment, tests T1, T2 and T3 are executed once for each leaf node in the tree being attached, and hence they are executed at most $n$ times. T1 and T3 each consist of node deletion, and each deletion involves testing $O(n)$ trust relationships and hence takes $O(n)$ time. T2 consists of checking one trust relationship per leaf node and hence takes $O(n)$ time. Thus each attachment takes $O(n^2)$ time at worst. These results are summarized in the following theorem:

**Theorem 4.16:** If $m$ is the number of internal nodes and $n$ the number of leaf nodes, there can be at most $m$ steps in Algorithm 4.3, and the worst case complexity of Algorithm 4.3 is $O(m^3n^2)$.

When $O(m) = O(n)$, by Theorem 4.16, the worst case complexity of Algorithm 4.3 is $O(n^5)$.

Theorem 4.9, together with the observation that the problem of designing a name space given 3-agent trust specifications is the general case of which the problems of designing a name space given 2-agent trust specifications are special cases, yields that putting an upper bound on the number of children of each node in the name space for 3-agent trust specifications is NP-complete. This is summarized in the following theorem:

**Theorem 4.17:** Given 3-agent trust specifications and either iterative or recursive channel composition, the problem of designing a name space with an upper bound on the number of children of each node is NP-complete.

4.9. An Example

The name space design algorithms described in this chapter have been implemented and experimented with. Figure 4.32 shows the name space synthesized by Algorithm 4.3 from a sample set of trust specifications. The trust specifications are enumerated in Appendix 1. Only the internal nodes of the name space are shown: each node is assumed to have leaf nodes corresponding to the employees of the organization labeling the node.

The trust specifications were then changed and the name space was reconstructed using Algorithm 1. The changes in trust specifications were that, the trust relationships in which agents trust *ibm* for *ibm-j* and vice-versa were replaced by trust relationships in which agents trust *ibm* for *sony-usa* and vice-versa. Figure 4.33 shows the reconstructed name space.

The two name spaces are quite different, even though the difference in their trust specifications is not significant. Thus, small changes in trust relationships can cause substantial differences in the name space configuration. This shows that trust relationships can have significant effects on the structure of a name space. Name space design that takes into account such non-trivial effects is too complicated to be carried out by manual trial-and-error methods for a large distributed system. Thus, the algorithms described in this chapter are useful for designing real distributed systems.
4.10. Conclusion

In a distributed system, it is desirable to have a tree of independent channels. A tree of independent channels also represents a global name space. There are two channel composition orders, namely, *iterative* and *recursive*. Iterative and recursive channel compositions require different trusts and are duals of each other. As one of the most important applications of a formal theory of trust, we have developed polynomial-time algorithms for synthesizing name spaces so that, given a channel composition order and the trust relationships of agents, channel composition between any two agents requires only a subset of the given set of trust relationships. The trust relationships are in general functions of three agents, but can also be functions of two agents, in which case the algorithms are simpler. Each node in the name space has to store the database of public keys of its children, and hence it is desirable to put an upper bound
on the size of this database. However, this problem is NP-complete. Sample runs of the name space design algorithms show that small differences in trust relationships can cause substantial differences in name spaces, thus demonstrating that trust relationships can have a significant effect on the structure of a name space. Design of a name space that takes into account the non-trivial effects of trust relationships is too complicated to be done by manual trial-and-error methods for a real distributed system (especially a VLDS), from which we can infer the practical utility of the algorithms described in this chapter.
CHAPTER 5

TRADING TRUST REQUIREMENTS FOR PERFORMANCE

No synthesis is complete without performance considerations. Under some conditions, to improve performance of channel establishment mechanisms, we may accept to increase their trust requirements. If channel composition is PKE-based, slightly increasing the trust requirements allows agent-to-agent channels to be built on top of host-to-host channels. Only the host-to-host channels need be established over the network, and this approach can greatly increase the performance of agent-to-agent secure communication with respect to that of establishing agent-to-agent channels directly. The accompanying increase in trust requirements is still always a subset of the set of trust specifications from which the system’s name space has been synthesized. However, if channel composition is SKE-based, this approach requires global trusts, which may not be satisfied in the name space. The protocol for establishing host-to-host channels can be handled at the subtransport level of the network protocol hierarchy. A prototype of the subtransport-level channel establishment protocol has been implemented on Sun 3/50 workstations connected by a 10 Mb/s Ethernet. Experimental measurements confirm that both the average latency of messages and the maximum throughput improve substantially when, instead of establishing agent-to-agent channels directly, host-to-host channels are established, and agent-to-agent channels are built on top of host-to-host channels. These improvements are primarily due both to the sharp decrease in the number of channel establishment operations across the network and to piggybacking of messages from several agent-to-agent channels on to a single host-to-host channel message.

5.1. Introduction

In Chapter 3, protocols for PKE- and SKE-based channel composition were analyzed for their trust requirements, and it was shown that PKE-based channel composition has much smaller trust requirements. In the previous chapter, we showed how to construct PKE-based name spaces given the trust relationships of all the agents sharing the distributed system. When a new channel, say channel($A_i, A_k$), is established using a PKE-based name space, $A_i$ obtains the public key of $A_k$. Authentication of a message from $A_k$ to $A_i$ on channel($A_i, A_k$) is provided by encrypting the message with $A_k$’s private key. Privacy of a message from $A_i$ to $A_k$ on channel($A_i, A_k$) is provided by encrypting the message with $A_k$’s public key.

However, public key encryption is expensive [Koc,NBS77]. An alternative way would be for $A_i$ and $A_k$ to agree upon a single key $S_{ik}$ using the first few messages on channel($A_i, A_k$), and then use $S_{ik}$ as the key of channel($A_i, A_k$) [Dif82,PoK79]. Authentication of a message from $A_k$ to $A_i$ on channel($A_i, A_k$) would in this case be provided by a cryptographic checksum computed using $S_{ik}$. Privacy of a message from $A_i$ to $A_k$ on channel($A_i, A_k$) is provided by encrypting the message with $S_{ik}$. This bootstrapped PKE-based channel composition protocol combines the advantages of PKE and SKE schemes, i.e., it has the efficiency of the SKE scheme while having the smaller trust requirements of the PKE scheme.

The bootstrapped PKE-based protocol, even though it is much more efficient than a pure PKE-based protocol, has significant performance disadvantages with respect to protocols without security mechanisms. Suppose that there are agents $A_{i1}$ and $A_{i2}$ on a host $H_A$, and there
are agents $A_{k1}$ and $A_{k2}$ on a second host $H_B$. If each agent on $H_A$ communicates with each agent on $H_B$ and vice versa, eight channels (four bidirectional channels) have to be established. Thus, the number of channels to be established in the worst case grows quadratically with the number of agents on the two hosts (see Figure 5.1). Each channel establishment involves agreeing upon a single key, and each agreement requires a three-way handshake protocol. The cost of such a channel establishment mechanism can become sufficiently prohibitive so as to dissuade agents from using secure communication channels entirely.

The goal of this chapter is to investigate whether trust relationships can be traded for performance; in particular we want to design a protocol with substantially improved performance but with slightly higher trust requirements. We will show how, in a PKE-based name space, by only slightly increasing the trust requirements, the performance can be greatly improved, while the increased trust requirements still form a subset of the set of trust specifications from which the name space has been synthesized. We will also show how, in an SKE-based name space, the requisite increase in trust requirements is so unacceptable as not to permit any practical increase in performance.

The performance disadvantages of pure/bootstrapped PKE-based channel establishment protocols stem from the fact that, for each pair of communicating agents on two hosts, a

---

**Figure 5.1: Multiple user-to-user channels**

---

1 When we say an agent is on a host, we mean that a process belonging to the agent is on the host. Similarly, an agent-to-agent channel between two agents means a process-to-process channel between two processes belonging to the two agents.
separate channel must be established across the network. To increase the performance, we must reduce the number of channels established across the network. In this chapter we present a channel establishment protocol called Authenticated Datagram Protocol (ADP) \(^2\), which establishes just one host-to-host channel across the network between any two hosts, and builds agent-to-agent channels on top of these host-to-host channels.

The reduction in the number of channels that are established across the network in ADP comes with an accompanying increase in trust requirements. This increase in trust requirements consists of trust relationships involving hosts on which agents have processes. In order effectively to describe these trust relationships involving hosts and agents, we present a high level model of process execution in Section 5.2. In Section 5.3 we describe ADP, and in Section 5.4 we derive ADP's trust requirements, and show how they are noticeably higher than those of a pure PKE-based channel establishment protocol. We then introduce some modifications to ADP that substantially reduce its trust requirements without affecting its performance. With these modifications, ADP's trust requirements become only slightly higher than those of a pure PKE-based channel establishment protocol. Moreover, the increased trust requirements still form a subset of the set of trust specifications from which the name space has been synthesized. In Section 5.5, we show that, if an SKE-based name space is used, building agent-to-agent channels upon host-to-host channels (as in ADP) results in global trust requirements which may not be satisfied in the name space. Section 5.6 defines the concept of a trust domain, and shows how it can be used further to increase the performance of ADP. Section 5.7 details the advantages of ADP over directly establishing agent-to-agent channels across the network, and Section 5.8 presents results of experimental measurements of a prototype of ADP that confirm its expected performance benefits. Finally, Section 5.9 concludes the chapter.

5.2. A Model of Process Execution on Hosts

Each host has a kernel running on it. At any point in time, each host has an agent that is the host's owner. Host ownership is established at boot time, before network communication takes place; it might be done manually or from a ROM. Usually, the agent who boots the kernel on the host becomes the host owner. Host ownership may change over time, e.g., as different people boot a public workstation. Each host has a (public-key, private-key) pair associated with it, which is the (public-key, private-key) pair of the host owner. The trust relationships of a host are those of its owner. A crash-free period under a single host owner is called a kernel session. In the sequel, we shall use the terms "host" and "kernel" interchangeably. We shall also use the terms "agent", "user" and "owner" synonymously.

The host may support multiple user processes, each of which has an agent as its owner, perhaps different from the host owner. Processes communicate with each other through messages. Each message has a message sender field containing the name of the owner of the sending process, and a message receiver field containing destination information. A kernel has access to the private keys of the host owner and of the owners of all the user processes it has executed or is executing.

A kernel must satisfy some correctness requirements with regard to security. There has been substantial formal work in the area of kernel security correctness [CGH81,Lan81,Sal74]. Without going into formal descriptions of secure kernels, for the purposes of discussing how

\(^2\) The protocol is so named because it provides message authenticity for all messages, and assumes the existence of an underlying network protocol that provides at least a host-to-host datagram service.
ADP might fit inside a secure kernel, we will give an intuitive set of conditions that a secure kernel must satisfy.

Figure 5.2 represents our model for the organization of the kernel. It consists of modules of code and private data. The passing of messages between modules is handled by a special kernel module called the message passing module. The channel establishment functions are handled by a module called the security module. Kernel modules which handle either an outgoing message before it is passed to the security module or an incoming message after it has been processed by the security module are called type-1 modules. Protocol modules above the layer at which channel establishment mechanisms are handled are examples of type-1 modules. Kernel modules which handle either an outgoing message after it has been processed by the security module or an incoming message before it has been processed by the security module are called type-2 modules. In a protocol architecture where channel establishment mechanisms are handled above the data link layer, a network driver is an example of a type-2 module. Kernel modules other than the security module, the message passing module, the type-1 modules, and the type-2 modules are called type-3 modules. The private keys are part of the private data storage of the security module. The security module, the message passing module, the type-1 modules, and the type-2 modules are together called critical modules.

A kernel is security-correct if the following conditions hold:

1. The only way for a user process to communicate with the kernel is through messages.
2. When a user process sends a message to a kernel module, the message passing module sets the message sender field to be the owner of that process. Thereafter, the message passing module and the type-1 modules do not change (a) the message sender field, or (b) the message receiver field, (c) the data part of the message.

![Figure 5.2: Kernel module structure](image)
The message passing module does not deliver a message to a user process if the owner of that user process is different from that indicated in the message receiver field.

The private data storage of the security module is read or written by no other module.

The security module executes its algorithms (to be given later) correctly.

Type-2 modules do not directly communicate with type-1 or type-3 modules or user processes.

These conditions were obtained from intuitive notions of what a non-malicious kernel should provide with respect to secure communication between user processes.

When an agent \( A_i \) executes a process on a security-correct host \( H_A \), there are some implications for \( A_i \)'s trust relationships. Since \( A_i \)'s private key is accessible to \( H_A \), \( H_A \) is assumed not to use the private key to masquerade as \( A_i \) or reveal secret messages sent to \( A_i \), and is assumed not to reveal the private key. Thus, a trust relationship that guarantees the following three conditions is required between \( A_i \) and the owner of \( H_A \):

For each agent \( A_x \) (\( x \neq i \)),

1. When \( A_x \) receives a message encrypted with \( A_i \)'s private key, the message was not sent by \( H_A \) masquerading as \( A_i \).
2. When \( A_x \) sends a secret message encrypted with \( A_i \)'s public key, host \( H_A \), which can decrypt this message with \( A_i \)'s private key, does not reveal the secret message.
3. \( H_A \) does not reveal \( A_i \)'s private key to \( A_x \).

If \( A_j \) denotes the owner of \( H_A \), the above trust relationship between \( A_i \) and \( A_j \) can be formally expressed using the key user-possessor trust defined in Chapter 3. The first aspect of the trust is expressed by message privacy trust, the second by trust against masquerading and the third by key privacy trust. The three together form key user-possessor trust, which with universal quantification over \( A_x \) defines the Universal Trust:

\[ T_U(A_i, A_j) \text{ (Universal Trust): } T_U(A_i, A_j) \text{ is true if and only if } \forall A_x \ x \neq i, \ T_{KUP}(A_x, A_j, A_i) = true. \]

\( T_U(A_i, A_j) \) is required to be true whenever a host owned by \( A_j \) has access to \( A_i \)'s private key, and vice versa.

Universal trust is transitive, as shown by the following theorem:

**Theorem 5.1 (Transitivity Theorem):** For all agents \( A_i, A_j \) and \( A_k \), if \( T_U(A_i, A_j) = true \) and \( T_U(A_j, A_k) = true \), then \( T_U(A_i, A_k) = true \).

**Proof:** The proof is fairly simple.

Since \( T_U(A_i, A_j) = true \), a host \( H_j \) owned by \( A_j \) has access to \( A_i \)'s private key in \( H_j \)'s storage. \( \quad (5.1) \)

Since \( T_U(A_j, A_k) = true \), a host \( H_k \) owned by \( A_k \) has access to \( A_j \)'s private key. With \( A_j \)'s private key in its possession, \( H_k \) has access to all of \( H_j \)'s storage. \( \quad (5.2) \)

\[ T_{KUP}(A_x, A_j, A_i) \text{ is trivially true for } x=j. \]
By (5.1) and (5.2), $H_k$ has access to $A_i$'s private key. Thus $T_U(A_i, A_k)$ must be true.

If we view the universal trust as a binary relation on agents, we can define a set of agents, denoted by $\text{key-closure}$, as the following union of transitive closures of the universal trust.

$\text{key-closure}(A_i): \text{Union of the transitive closures of } T_U(A_i, A_y) \text{ for all } A_y.$

$\text{Key-closure}(A_i)$ will contain the owners of all hosts that have access to $A_i$'s private key.

5.3. The Authenticated Datagram Protocol

ADP [AnV87] is a host-to-host channel establishment protocol, and hence is handled at the subtransport level of a network protocol hierarchy [AFV87c]. ADP has been designed and implemented [AFV87b] as part of DASH, an experimental distributed operating system [AFV87a] being designed at the University of California at Berkeley.

DASH is an open system in which many transport-level protocols [Tan81, Tan88], both stream-oriented and request/reply, may exist. The clients of ADP are kernel-level transport protocol modules, and ADP in turn is a client of multiple network-level services that provide at least a host-to-host datagram service [Tan81, Tan88] (see Figure 5.3).

ADP maintains two kinds of channels, host-to-host channels and agent-to-agent channels, with the latter being built on top of the former. Host-to-host channels are also called ADP channels. A summary description of ADP's operation, as it would be carried out between two hosts $H_A$ and $H_B$, is as follows. The two hosts establish an ADP channel using a bootstrapped PKE-based channel composition protocol. No agent-specific channels are established across the network - all user messages between $H_A$ and $H_B$ are sent on the ADP channel between them. This reduces the worst case channel establishment overhead from being a quadratic function of the number of users to being a constant factor. $H_A$ and $H_B$ build agent-to-agent channels upon their ADP channel by sending to each other the PKE-based certificates [Akl83, Den84b]

---

Figure 5.3: Position of ADP in the ISO/OSI model of network architecture
91

(described below) of their respective users. The two hosts cache the PKE-based certificates sent and received, thereby reducing the overhead of building agent-to-agent channels over their ADP channel.

5.3.1. ADP Channels

ADP establishes an ADP channel between two hosts $H_A$ and $H_B$ when they communicate for the first time, and thereafter the channel continues to exist until one of the hosts fails. The protocol for channel establishment consists of the following steps (see Figures 5.4 and 5.5):

1. $H_A$ sends an ADP channel request message to $H_B$. This message contains two random strings $S$ and $T$. $S$ is encrypted with $H_B$'s public key for privacy. $T$ may be sent in cleartext. The entire message is cryptographically checksummed with $H_A$'s private key for authenticity [Akl, Den84a]. $S$ will be used as the single key of the ADP channel between $H_A$ and $H_B$, and $T$ will be used for certificates from $H_B$ to $H_A$.

2. $H_B$ sends an ADP channel acknowledgement message containing a random string $R$ to be used for certificates from $H_A$ to $H_B$. The first certificate sent from $H_A$ to $H_B$ serves to complete a three-way handshake for the ADP channel establishment. If both $H_A$ and $H_B$ simultaneously try to establish an ADP channel to each other, the host with the lexicographically greater name determines the channel key $S$.

In Section 5.4, we derive the trust requirements necessary if this protocol is to result in the establishment of a host-to-host channel $H_A - H_B$, i.e., channel($H_A, H_B$) and channel($H_B, H_A$).

5.3.2. Sending Certificates of Agents

If $A_k$ is an agent, we say that $A_k$ is certified from $H_B$ to $H_A$ if $H_B$ sends the string $\{H_B, R\}$, which is a concatenation of the name $H_B$ and the random number $R$ specified by $H_A$, encrypted with $A_k$'s private key on the ADP channel between $H_A$ and $H_B$ (see Figure 5.6). When $H_A$ receives the certificate, $H_A$ decrypts it with $A_k$'s public key, and compares the result with $\{H_B, R\}$. In Section 5.4, we derive the trust requirements necessary if the certification of $A_k$ from $H_B$ to $H_A$ is to result in the establishment of channel($A_x, A_k$), for all agents $A_x$ having processes on $H_A$.

The sending of $A_k$'s certificate from $H_B$ to $H_A$ is normally done only once on an ADP channel. $H_A$ and $H_B$ both maintain identical tables of agents that have been certified from $H_B$ to $H_A$, and separate tables for agents certified in the reverse direction (see Figure 5.7). This caching of certificates means that expensive PKE-based encryption is done only once per agent per host per ADP channel.

5.3.3. Messages on an ADP Channel

Three levels of messages must be distinguished (Figure 5.2):

Client messages are the messages read or written by clients of ADP.

ADP messages are a logical unit of exchange between ADP instances on different hosts. An ADP message consists of a header followed by one or more items, each of which may be 1) a client message; 2) an agent's certificate or a request for an agent's certificate; 3) a request to establish a ADP channel, or the acknowledgement of such of a request; 4) a request to change the key of a ADP channel.

Network messages: the network facility underlying ADP is assumed to provide an insecure and unreliable datagram service. If the size of an ADP message exceeds the maximum size allowed by the network layer beneath ADP, ADP divides the ADP message into multiple network
Host A

\[\text{user} \quad \text{Ai} \quad \text{kernel} \quad \text{ADP}\]

Host B

\[\text{Ak} \quad \text{user} \quad \text{kernel} \quad \text{ADP}\]

Figure 5.4: Hosts \(H_A\) and \(H_B\) just before establishing an ADP channel

---

**Figure 5.5:** Establishment of an ADP channel between \(H_A\) and \(H_B\). \(H_A\) sends the channel request \(msg_1\) consisting of the doubly encrypted channel key \(S\) (encrypted first with \(H_A\)'s private key and then with \(H_B\)'s public key) and a random number \(T\). \(H_B\) sends the channel reply \(msg_2\), consisting of a random number \(R\).
messages.

5.3.4. The ADP Client Interface

Message addressing is done on the basis of network-dependent host addresses and, on a particular host, multiple ports. Ports have identifiers (port ID’s) that are guaranteed to be unique on a given host between crashes. ADP clients inform ADP that messages can be delivered to the given port using

```c
register_port(
    port_ID port, // the port being registered
    char *local_agent, // agent associated with the port
);```

where local_agent is an agent whose private key is known to the host. ADP may then deliver messages to the port. Each message is prepended with the name of its sender. Clients of ADP can send messages using
Figure 5.7: Certification of agent $A_i$ from $H_A$ to $H_B$. $H_A$ sends a certificate $msg_4$ consisting of the string $(H_A, R)$ encrypted with $A_i$'s private key.

```
ADP_send(
    MESSAGE *msg, // the message being sent
    char *local_agent, // name of sender
    char *remote_host, // destination host name
    char *remote_port, // destination port ID
    char *remote_agent, // name of recipient
    BOOLEAN privacy, // whether message is private
    int max_delay // maximum local queueing delay
)
```

The `remote_agent` argument is used only if `privacy` is true; in this case ADP will obtain a certificate of the agent on the remote host before sending the message. The `max_delay` parameter is a time interval (in microseconds) for which this message can be queued locally (see Section 5.3.6).

5.3.5. Transmission of Client Messages

ADP's handling of client messages depends on the nature of the intervening network. A physical broadcast network (PBN) is one in which there is a single transmission medium. In the absence of packet loss due to buffer overrun, if any node on a PBN receives a packet in its entirety, then the node to which it is addressed also does so. A single Ethernet, for example, is a PBN. Token rings and bridged Ethertnets, though they may support logical broadcast, are not
PBN's.

ADP messages to a destination on the same PBN as the sender are transmitted as a sequence of network packets (fragments), each of which ends with a security trailer \(^4\) containing a sequence number encrypted with the channel key. When the node to which a fragment is addressed receives a packet, the node decrypts the security trailer to obtain the packet's sequence number. If the sequence number is greater than that of the previously received packet, this is the first packet on the network with this sequence number. Since the sequence number is encrypted with the channel key, and the host at the other end of the ADP channel must have sent the packet, the packet is authentic and is accepted by the destination node. Thus, security trailers provide message authentication on PBN's. The destination ADP module handles reassembly. Strictly increasing sequence numbers are used; when the space of sequence numbers is exhausted, a new channel key is negotiated. Client messages for which privacy was requested are sent encrypted with the channel key; others are sent in cleartext.

If the destination host is not on a common PBN, ADP uses a lower-level Internet Protocol [81b] module to handle routing and fragmentation. ADP delivers complete ADP messages to IP with a security header that includes a cryptographic checksum [Akl, Den84a] of the entire message, encrypted with the channel key. The IP module at the destination host reassembles the ADP message and delivers it to the ADP module, which recomputes the cryptographic checksum and verifies that it matches the encrypted version. As before, private client messages are encrypted.

5.3.6. Piggybacking

In some cases, system performance can be increased by piggybacking multiple client messages into a single ADP message (see Figure 5.8). This is made possible by allowing ADP clients to specify a maximum queueing delay for each message. Many types of messages, such as retransmissions, asynchronous write operations, and some types of acknowledgements, can be delayed a small amount (a fraction of a second) with no loss in system performance or functionality. These "non-urgent" messages can therefore be queued in the sender for this period and merged with other client messages on the same channel. In this case, one ADP message may include several client messages. If the client messages do not require secrecy, then in general less encryption is required, since a single encrypted sequence number or checksum will serve to authenticate multiple client messages. Piggybacking may also reduce CPU overhead, since the per-message costs of piggybacking (queueing and timers) are likely to be lower than those of network packet transmission.

The maximum delay of a client message is supplied by the process sending it (usually a transport protocol). If the delay is zero (i.e., for "urgent" messages) ADP will send the message as soon as possible. If the delay is nonzero the message may be queued for a period not exceeding the delay. The queueing period will be less if the queue exceeds the maximum ADP message size, or if there is a shorter-delay message in the queue. This maximum ADP message size will depend on the amount of buffer space available, and may be limited by the maximum size of the messages that can be accepted by the lower protocol layers.

\(^4\) Either a security trailer or a security header can be used.
5.4. Trust Requirements of ADP

The establishment of an ADP channel between two hosts $H_A$ and $H_B$ results in the logical establishment of two channels $(H_A, H_B)$ and $(H_B, H_A)$, if some trust requirements are satisfied. Let us investigate these trust requirements.

In the ADP channel establishment protocol, let $H_A$ send the ADP channel request. In order to send the request, $H_A$ must obtain $H_B$'s public key using name resolution in the name space. But notice that obtaining $H_B$'s public key results in the establishment of a PKE-based channel $(H_A, H_B)$ (see Section 3.4). Using results of Section 3.7, security of this channel requires the satisfaction of a trust predicate, which we denote by $\text{pred}_{AB}$.

When $H_B$ receives the ADP channel request from $H_A$, $H_B$ must obtain $H_A$’s public key to decrypt the request. Obtaining $H_A$’s public key results in the establishment of a PKE-based channel $(H_B, H_A)$. Let the trust predicate that must be satisfied for the security of this channel be denoted by $\text{pred}_{BA}$.

If both $\text{pred}_{AB}$ and $\text{pred}_{BA}$ are true, $H_A$ and $H_B$ correctly possess each others’ public keys, and the single key $S$ sent by $H_A$ in the ADP channel request is known only to agents hôsts that might possess either $H_A$’s private key or $H_B$’s private key, i.e., the agents hôsts in key-

---

5 The trust predicate is determined by the sequence of channel compositions (in the name space) used in establishing the channel.
closure($H_A$) or in key-closure($H_B$). Since the agents in key-closure($H_A$) and key-closure($H_B$) possess the private keys of $H_A$ and $H_B$ respectively, we obtain that the following must be true (recall the meaning of $T_U$ from Section 5.2):

$$T_U(H_A, \text{key-closure}(H_A)) \land T_U(H_B, \text{key-closure}(H_B))$$  \hfill (5.3)

If $H_A$ and $H_B$ were to use $S$ as their channel key, $H_A$ and $H_B$ must have key-user possessor trust in all the agents that might possess $S$, so as to ensure that these agents will not use $S$ to compromise the security of the ADP channel between $H_A$ and $H_B$. Since the agents that might possess $S$ are those in key-closure($H_A$) or key-closure($H_B$), we obtain that the following must be true:

$$T_{\text{KUP}}(H_A, \text{key-closure}(H_A), H_B) \land T_{\text{KUP}}(H_A, \text{key-closure}(H_B), H_B) \land$$

$$T_{\text{KUP}}(H_B, \text{key-closure}(H_A), H_A) \land T_{\text{KUP}}(H_B, \text{key-closure}(H_B), H_A)$$  \hfill (5.4)

But notice that, using the definition of the universal trust (see Section 5.2),

$$T_{\text{KUP}}(H_A, \text{key-closure}(H_B), H_B)$$

follows from:

$$T_U(H_B, \text{key-closure}(H_B)),$$

and

$$T_{\text{KUP}}(H_B, \text{key-closure}(H_A), H_A)$$

follows from:

$$T_U(H_A, \text{key-closure}(H_A))$$

Hence, in eq (5.4),

$$T_{\text{KUP}}(H_A, \text{key-closure}(H_B), H_B) \land T_{\text{KUP}}(H_B, \text{key-closure}(H_A), H_A)$$

follows from eq (5.3).

Summarizing these results, we obtain the following theorem:

**Theorem 5.2**: Establishment of an ADP channel between two hosts $H_A$ and $H_B$ results in the logical establishment of channels $(H_A, H_B)$ and $(H_B, H_A)$ if and only if:

1. The trust requirements for establishing channels $(H_A, H_B)$ and $(H_B, H_A)$ using channel compositions in a PKE-based name space are satisfied,
2. $H_A$ has universal trust in all agents in key-closure($H_A$) and $H_B$ has universal trust in all agents in key-closure($H_B$), and
3. $T_{\text{KUP}}(H_A, \text{key-closure}(H_A), H_B)$ and $T_{\text{KUP}}(H_B, \text{key-closure}(H_B), H_A)$ are satisfied.

---

6 The following short-hand notations are used throughout: A host's name mentioned in a place where an agent's name would be expected refers to the host's owner, an agent's name mentioned in a place where a host's name would be expected refers to all the hosts owned by the agent, and a set mentioned in a place where a member of the set would be expected refers to all the members in the set.
Now consider agent certification. When $H_A$ receives a certificate of an agent $A_k$ from $H_B$, for all agents $A_x$ that have processes on $H_A$, channel($A_x, A_k$) becomes established if some additional trust requirements are satisfied. Let us investigate these trust requirements.

When $H_A$ receives $A_k$'s certificate, $H_A$ must obtain $A_k$'s public key. Obtaining $A_k$'s public key results in the establishment of a PKE-based channel($H_A, A_k$). Let the trust predicate that must be satisfied for the security of this channel be denoted by $pred_{Ak}$.

If $pred_{Ak}$ is satisfied, $H_A$ correctly possesses $A_k$'s public key, and hence some agent that has $A_k$'s private key sent $A_k$'s certificate to $H_A$. If $T_U(A_k, key\text{-}closure(A_k))$ is satisfied, $A_k$ must have sent the certificate. Since the certificate contains the name of $H_B$, $A_k$ on $H_B$ must have sent the certificate. Since the certificate contains the random number $T$, $A_k$ on $H_B$ must have sent the certificate during the kernel session of the current ADP channel. Thus, $H_B$ has a process owned by $A_k$ during the current kernel session.

Now consider a message $msg_{kr}$ sent from $H_B$ to $H_A$ with the message sender field equal to $A_k$, and the message recipient field equal to $A_x$. Since $H_B$ has a process owned by $A_k$ during the current kernel session, $H_B$ possesses $A_k$'s private key, $H_B$ belongs to key-closure($A_k$), and hence $T_U(A_k, H_B)$ is satisfied. Thus, $H_B$ does not masquerade as $A_k$, and hence $msg_{kr}$ must have been sent by $A_k$. If the trust requirements of Theorem 5.2 are satisfied, the authenticity of $msg_{kr}$ remains intact between $H_B$ and $H_A$. If $A_x$ has a process on $H_A$ and $H_A$ is security-correct, the authenticity of $msg_{kr}$ remains intact from $H_A$ to $A_x$.

A similar derivation can be carried out for the privacy of a secret message sent from $A_x$ to $A_k$.

The following theorem summarizes the above derivations for the trust requirements of channel($A_x, A_k$).

**Theorem 5.3:** Suppose $A_k$ is an agent having processes on a host $H_B$. For all agents $A_x$ that have processes on a host $H_A$, channel($A_x, A_k$) is established when $H_A$ receives $A_k$'s certificate on its ADP channel to $H_B$ if and only if $H_A$ is security-correct and the following trust requirements are satisfied:

1. the trust requirements specified by Theorem 5.2 for establishing channels ($H_A, H_B$) and ($H_B, H_A$),
2. the trust requirements for establishing channel($H_A, A_k$) using compositions in a PKE-based name space, and
3. $T_U(A_k, key\text{-}closure(A_k))$.

In the trust requirements given by Theorem 5.3 for channel($A_x, A_k$), the trust predicates $pred_{Ak}, pred_{AB}$ and $pred_{BA}$ arise from channel compositions in the name space. Since the name space is designed so as to satisfy the trust relationships in any channel composition carried out through it, $pred_{Ak}, pred_{AB}$ and $pred_{BA}$ are automatically satisfied. But the universal and key user-possessor trust requirements which involve the various key-closures can potentially greatly increase the trust requirements, and hence it is desirable to eliminate these trust requirements.

Figure 5.9 illustrates an effect of a universal trust requirement involving a key-closure. Alice has a process on host $H_A$ owned by Riccardo, and Bob has a process on host $H_B$ owned by Stuart. The channel request $CR_{AB}$ of the ADP channel between $H_A$ and $H_B$ was sent by $H_A$, encrypted with Riccardo's private key and Stuart's public key. Stuart himself has a process on
a host $H_C$ owned by Peter. Since a host has access to its users' storage, Peter has access to Stuart's private key on $H_C$. Thus Peter can obtain the single key $SK_{AB}$ of the ADP channel between $H_A$ and $H_B$ by decrypting $CR_{AB}$ with Stuart's private key and Riccardo's public key. Since messages from Bob to Alice are encrypted using $SK_{AB}$, Peter can compromise the security of communication between Bob and Alice. In other words, since Bob has placed universal trust in Stuart, and Stuart has placed universal trust in Peter, Bob has to place universal trust in Peter. In fact, Bob must place universal trust in owners $O_1$ of all hosts on which Peter might have user processes, in owners $O_2$ of all hosts on which owners $O_1$ might have user processes, and so on. Thus, the requirement of universal trust in key-closures can be a serious disadvantage.

Let us see if we can eliminate the universal and key user-possessor trust requirements involving the various key-closures. The primary cause for these trust requirements in the above example is the accessibility of Stuart's private key by Peter. Suppose Stuart's host uses a (public-key, private-key) pair that is different from the (public-key, private-key) pair used by Stuart's user-level process on Peter's host. $H_C$ no longer has access to the private key used by $H_A$. Peter can no longer decrypt the channel request from $H_A$ to $H_B$, and hence Peter can no longer obtain the key of the ADP channel between $H_A$ and $H_B$. Consequently, the trust requirements involving key-closures vanish.

Let $ADP_{\text{modified}}$ denote ADP with the modification that each agent has two (public-key, private-key) pairs, one of which is used by the agent's hosts and the other is used by the agent's user-level processes. Key-closure($H_A$) and key-closure($H_B$) become empty. Key-closure($A_k$) becomes the set of hosts, hosts ($A_k$), on which $A_k$ has processes. Using these substitutions in Theorems 5.2 and 5.3, we obtain the next two theorems for $ADP_{\text{modified}}$:

Theorem 5.4: Establishment of an $ADP_{\text{modified}}$ channel between two hosts $H_A$ and $H_B$ results in the establishment of channels $(H_A, H_B)$ and $(H_B, H_A)$ if and only if the trust requirements for establishing channels $(H_A, H_B)$ and $(H_B, H_A)$ using a PKE-based name space are satisfied.

---

**Figure 5.9: Effect of key-closure trust requirements**
Theorem 5.5: Suppose $A_k$ is an agent having processes on a host $H_B$. For all agents $A_x$ that have processes on a host $H_A$, channel($A_x, A_k$) is established when $H_A$ receives $A_k$'s certificate on its $ADP_{\text{modified}}$ channel to $H_B$, if and only if $H_A$ is security-correct and the following trust requirements are satisfied:

1. The trust requirements specified by Theorem 5.4 for establishing channels ($H_A, H_B$) and ($H_B, H_A$).
2. The trust requirements for establishing channel($H_A, A_k$) using channel composition in a PKE-based name space, and
3. $T_U(A_k, \text{hosts}(A_k))$.

Notice that the first two trust requirements of $ADP_{\text{modified}}$ (as given by Theorem 5.5) are automatically satisfied in the name space. The third trust requirement involves only those hosts on which $A_k$ has/had processes, and, given that a host has access to all the data of an agent having processes on the host, this trust requirement cannot be eliminated by any channel establishment protocol.

If agent-to-agent channels are established directly, establishing channel($A_i, A_k$) requires that a trust predicate $\text{pred}_k$, which is determined by the sequence of channel compositions between $A_i$ and $A_k$ in the name space, be satisfied. In comparison, the trust requirements of ADP (as given by Theorem 5.5) are more numerous. In Section 5.7, we justify the increased trust requirements of ADP by its significant performance advantages over protocols that establish agent-to-agent channels directly.

5.5. Trust Requirements of ADP When Name Space is SKE-based

We now show that if the name space is SKE-based as in [BLN86], ADP requires global trust. Consider a system consisting of hosts $H_A$ and $H_B$, agents $A_i, A_k1, A_k2$, and an SKE-based name space in which $H_A, H_B, A_i, A_k1, A_k2$ are leaf nodes (see Figure 5.10). Let $A_i$ have processes on $H_A$, and $A_k1, A_k2$ have processes on $H_B$. Let hosts $H_A$ and $H_B$ establish an ADP channel with a channel key $S_{AB}$. As pointed out earlier, establishing the ADP channel involves the establishment of SKE-based channel($H_A, H_B$) and SKE-based channel($H_B, H_A$) using compositions in the name space. Let the set of name space nodes between $H_A$ and $H_B$ be denoted by $NS_{A-B}$. The nodes in $NS_{A-B}$ are involved in establishing channel $H_A-H_B$. If the name space is SKE-based, as shown in Chapter 3, the nodes in $NS_{A-B}$ might possess channel key $S_{AB}$ of channel $H_A-H_B$.

When $H_A$ receives a certificate of $A_k1$, SKE-based channel($H_A, A_k1$) must be established using compositions in the name space. The key of channel($H_A, A_k1$) becomes known to all name space nodes in the path between $H_A$ and $A_k1$. But ADP uses the same single key $S_{AB}$ for channel($H_A, A_k1$). Thus $S_{AB}$ is now known to agents in any of $NS_{A-B}$ or $NS_{A-k1}$.

---

7 In the sequel, ADP will be used to mean $ADP_{\text{modified}}$. 
Next, when $H_A$ receives a certificate of $A_{k_2}$, channel($H_A, A_{k_2}$) is established using compositions in the name space. Using similar arguments as above, we obtain that $S_{AB}$ is now known to agents in any of $NS_{A-B}$, $NS_{A-k_1}$ or $NS_{A-k_2}$.

When $A_i$ on $H_A$ sends a secret message to either $A_{k_1}$ or $A_{k_2}$, $S_{AB}$ is used for encryption. Since $S_{AB}$ is known to agents in any of $NS_{A-B}$, $NS_{A-k_1}$ or $NS_{A-k_2}$, the following key user-possessor trusts are required:

$$T_{KUP}(A_i, \{NS_{A-B}, NS_{A-k_1}, NS_{A-k_2}\}, A_{k_1}) = true \quad (5.4)$$

$$T_{KUP}(A_i, \{NS_{A-B}, NS_{A-k_1}, NS_{A-k_2}\}, A_{k_2}) = true \quad (5.5)$$

Continuing these arguments it can be shown that, if an agent having processes on $H_B$ can reside at any position in the name space, each agent on $H_A$ will be required to have key user-possessor trust in all the name space nodes (the middle argument to $T_{KUP}$ above becomes the universal set), and hence will be required to have global key user-possessor trust. This is summarized by the following theorem:

**Theorem 5.6:** ADP, if used in a system whose name space is SKE-based, requires global key user-possessor trust.

As a result of Theorem 5.6, ADP plus an SKE-based name space is an unacceptable combination in distributed systems without global trust.
5.6. Trust Domains

The set of hosts in many distributed computing environments may contain subsets with the following property: Within a subset, the hosts and the communication channels between them are physically secure, and agents with access to the hosts all place universal trust in one another (Figure 5.11). By Theorem 5.1, universal trust is transitive, and hence the same host cannot belong to two domains. Across subsets, the communication links may not be physically secure, and agents in one subset may not place universal trust in hosts in the other subset. We call such subsets trust domains. Suppose also that all communication across a subset boundary is routed through one or more hosts called domain gateways. Within a trust domain, no channel establishment mechanisms are necessary. Between two domains, the two domain gateways can establish a channel, and multiplex messages from, and demultiplex messages to, agents on various hosts within each domain. A special ADP module on the domain gateway performs these functions. This has the following advantages:

- **Efficiency**: Communication within the domain has no encryption overhead. Only the domain gateway does encryption, so only it need to have encryption hardware.
- **Flexibility**: The channel establishment mechanism between domain gateways can be changed at any time. Intra-domain communication will not see any changes.

5.7. ADP versus Direct Establishment of Agent-to-Agent Channels

Channel establishment mechanisms must be introduced into some levels of the network protocol architecture [Tan81, Tan88]. ADP being a host-to-host channel establishment protocol, these mechanisms can be introduced at the subtransport level. Protocols that establish agent-to-agent channels directly across the network must be introduced at transport or higher levels, because the lowest level at which processes (belonging to agents) rather than hosts can be communicating entities, is the transport level [SRC84, VoK83]. If agent-to-agent channels are established directly, establishing channel($A_i$, $A_k$) requires that a trust predicate $pred_{ik}$, which is determined by the sequence of channel compositions between $A_i$ and $A_k$ in the name space, be satisfied. In comparison, the trust requirements of ADP (as given by Theorem 5.5) are more numerous.

We will now justify the increased trust requirements of ADP (as compared to those of protocols that establish agent-to-agent channels directly) by the performance advantages of subtransport level channel establishment over channel establishment at transport or higher level protocols. The advantages can be grouped as follows:

1) general advantages of subtransport level channel establishment,
2) specific advantages relative to transport level channel establishment, and
3) specific advantages relative to putting channel establishment above the transport level.

---

8 Suppose a host $H_A$ belongs to two domains. The hosts of the first domain place universal trust in $H_A$, but $H_A$ places universal trust in hosts of the second domain. By transitivity of universal trust, hosts of the first domain place universal trust in those of the second. Similarly, we can show that hosts of the second domain place universal trust in those of the first. Consequently, the two domains can in this case be coalesced into a single domain.

9 Recall that a host can only belong to a single domain. Thus, two domains must have two different domain gateways.
5.7.1. General Advantages of Subtransport Level Channel Establishment

Putting channel establishment at the subtransport level has several advantages relative to putting it at higher protocol levels:

- It simplifies transport level protocols. When a host crashes, its channels are destroyed. Thus, remote host crashes can be detected at the host-to-host level at the time of channel establishment, and transport level protocols do not have to employ elaborate timer mechanisms to detect them [Che86, 81b]. If transport protocols above ADP employ a sequence number that is monotonically increasing within a channel, message duplicates and replays may be eliminated. Since a host crash initiates a new channel, duplicates across crashes are eliminated. Thus, 3-way handshakes are not required in transport protocols for the purpose of duplicate elimination. This also means that 3-way handshakes can often be eliminated from transport-level protocols. A short transaction then requires just two messages in the best case, as opposed to at least six in TCP [Dif85, Ken77] and four in secure RPC [BiN84, Bir85].

- More than one protocol may exist at higher layers, and different protocols may require different channel establishment mechanisms. Thus, unlike in the subtransport layer, channel establishment mechanisms may have to be duplicated in higher layers.

- There are two public-key operations per agent per remote host per kernel session. Often these operations can be done at boot time or during idle periods. There are no per-process or per-operation public-key operations, resulting in a substantial performance gain.

- Since messages from all client processes and higher level protocols pass through the subtransport layer, a number of these messages destined to a common remote host can all be
combined into a single datagram and sent as a single ADP message. This can reduce the number of single key operations.

- In the presence case of trust domains, handling channel establishment at the subtransport level drastically reduces the number of transport level connections necessary for communication between two processes in two different domains (see Figure 5.12).

5.7.2. Disadvantages of Transport Level Channel Establishment

Transport level protocols are used to implement a variety of communication paradigms. Request/reply (RPC) [BiN84] and full duplex byte streams [81a] are two of the popular communication paradigms. We examine secure RPC [Bir85] as an instance of channel establishment in an RPC protocol, and secure TCP [Dif85, Ken77] as an instance of channel establishment in a full duplex byte stream protocol. Both secure RPC and secure TCP are transport level protocols.

---

**Figure 5.12:** Comparison of ADP (Figure (a)) and transport level channel establishment (Figure (b)) with regard to trust domains. In ADP, the number of transport level connections required for inter-domain channels $A_{i1} - A_{k1}$ and $A_{i2} - A_{k2}$ is much smaller.
5.7.2.1. Secure RPC

When a client issues its first RPC request to a remote server, the RPC mechanism establishes a channel between the two processes. This consists of agreeing on a channel key to be used for encrypting RPC requests and replies. There are several disadvantages of such a scheme:

- For each channel, the RPC system must maintain long-term state information consisting of a channel key and sequence numbers of requests within the channel. This converts simple stateless RPC into one with long-term state information.

- A three-way handshake is necessary to agree upon the channel key. There are $O(n^2)$ encryption keys to be agreed upon. The cost of this three-way handshake is small if it is amortized over many RPC's. If, however, there are lots of short-lived processes making just one or two remote procedure calls, the performance penalty due to a three-way handshake is substantial. This can reduce the efficiency of RPC for short transactions.

- There are four public key operations for each channel. If the channel is established for just a single RPC, the relative cost is substantial.

- There is a single key encryption and a decryption for each RPC request, reply, and acknowledgement. Since messages from different processes use different channel keys, it is not possible to reduce the encryption cost by piggybacking messages from different processes that are all destined to the same host.

5.7.2.2. Secure TCP

TCP is a DARPA Internet transport protocol [81a] providing full duplex byte stream connections between processes on different hosts. Secure TCP [Dif85] requires an initial agreement upon a single key to be used during the lifetime of the TCP connection after the end points are authenticated to each other. In addition to those mentioned in Section 5.7.1, there are two more disadvantages associated with this scheme:

- Four public key operations are performed for each TCP connection.

- Encryption cost reduction by piggybacking is impossible since keys are not per host-pair.

5.7.3. Disadvantages of Having Channel Establishment above the Transport Level

There are several disadvantages in placing channel establishment mechanisms above the transport level:

- Transport level protocols like TCP do connection establishment using three-way handshakes. If channel establishment mechanisms are above the transport layer, they require their own handshake to agree upon keys after the transport level has established a connection. This duplication of handshaking entails higher message overhead.

- Transport level protocols do error detection using (insecure) checksums. Channel establishment mechanisms above the transport layer must do their own cryptographic checksumming. This is an unnecessary duplication of effort as error detection at higher layers can be avoided if checksumming is done at the subtransport or transport level.

- Transport level protocols employ sequencing to eliminate duplicates and out-of-sequence messages. Since an intruder could change the transport level headers and hence the transport level sequence numbers, channel establishment mechanisms above the transport layer must also do sequencing to detect such an intrusion, again resulting in an unnecessary duplication of effort.
If an intruder sends a false message with the correct transport level sequence number, the transport level protocol will accept it as the next message and reject the true message which may arrive later. The channel establishment mechanisms above will reject the false message correctly, but will never get the true message. False acknowledgements at lower levels can disrupt the sequencing. The only way to recover from such situation is to re-establish the connection at both the transport and the secure communication levels. This has the potential for much unnecessary tearing down of connections and the associated performance overhead.

Unauthenticated messages are detected only at the level where channel establishment mechanisms are. These messages are unnecessarily processed at all lower levels of the protocol hierarchy. Thus, if the channel establishment mechanisms are at a high level, the amount of this unnecessary work can be large (but this should be a rare occurrence).

Public key operations are more numerous than those required by ADP, and single key operations cannot be reduced by piggybacking.

5.8. Experimental Verification

No design is complete without performance evaluation [Fer78]. It is quite clear from what has been said in the previous sections that the performance advantages of ADP can be expected to be primarily due to:

(a) the reduced total overhead for channel establishment: in ADP, expensive public-key encryption and three-way message handshake overhead are needed only for setting up a host-to-host channel and once per agent per host (for sending certificates), rather than for every agent-to-agent connection or session; ADP channels will be many fewer in number and much longer lived; and

(b) the much higher likelihood that the benefits of piggybacking will be felt, as the traffic intensity on a host-to-host channel is never lower than that on an agent-to-agent channel involving the same two hosts, and is often much higher; one can easily speculate that the effectiveness of piggybacking grows with the traffic intensity, as more client messages can be shipped within one ADP message; also, with channel establishment mechanisms at a higher level, each channel will have its own channel key, and client messages traveling on two different channels between the same two hosts cannot be bundled together in the same ADP message, as their encrypted portions will require two different keys.

However, given that ADP is better than direct agent-to-agent channel establishment, an important question is: How much better is it?

To give this question an empirical answer, we measured the performance of a prototype of ADP implemented as part of the DASH Project at the University of California at Berkeley [AFV87d]. The implementation is written in C++ and runs on Sun 3/50 workstations connected by 10 Mb/s Ethernet.

Since we had not yet built any transport protocols on top of ADP, we could not implement agent-to-agent channel establishment mechanisms in them. We therefore transformed ADP for some of the experiments into an agent-to-agent channel establishment protocol, i.e., we had ADP establish a channel (using public key encryption) every time a process started communicating with another process on another host.

Because of the influences on ADP performance of client message sizes and arrival times, we could not use a synthetic input such as those of some previous studies, where all messages have the same size and arrive at regular intervals. Thus, it was decided to run trace-driven
measurement experiments. Figure 5.13 shows the experimental setup.

Figure 5.13: ADP experimental setup
Since a complete DASH system incorporating ADP does not exist yet, a real ADP input trace cannot be measured. The assumption was made that the message traffic on a local-area network interconnecting a variety of machines, including diskless workstations and file server, would represent a reasonable approximation to the type of traffic that an ADP module will experience in a DASH system.

A trace of all packets transmitted on a 10 Mb/s Ethernet among 96 machines of various types, 49 of which were diskless Sun workstations and 6 were Sun file servers, was converted into the corresponding client message trace, and also decomposed into traces containing only the messages generated by a given transport level protocol. In particular, in the experiments whose results are summarized below, we used the following three traces:

- **ALL**: a trace including all client message types;
- **TCP**: a trace containing only TCP messages (TCP was taken as an example of a transport level agent-to-agent full-duplex byte stream protocol).
- **NFS**: a trace containing only SUN NFS messages (NFS was taken as an example of a transport level agent-to-agent request/response protocol).

The primary performance indices we measured were:

- **Latency \( L \)**: the average delay incurred by a message between the instant it is given to the sending ADP module for transmission and the instant it is delivered by the receiving ADP module to the destination process on the destination host. To compute \( L \), we averaged the delays of the messages in a given finite sequence.

- **Throughput \( T \)**: the maximum rate at which information can be transmitted by ADP on the sending host and received by ADP on the destination host.

In a throughput experiment, the client messages in the trace arrive at such a high rate that the input queue of ADP is never empty. In the agent-to-agent channel establishment case, when the arrival rate of a message trace is increased with respect to the measured one, a decision must be made about whether and how the process creation rate should be modified. There are two extreme cases: (1) The process creation rate is kept constant while the rate of message production by the existing processes is increased. This is one end of the spectrum and represents the best possible case for an agent-to-agent channel establishment protocol. The throughput for this scenario will be denoted by \( T_1 \). (2) The mean message production rate of processes is fixed. Thus, when the message transmission rate is increased to its maximum value, the process creation rate must be increased linearly. This represents the worst case for agent-to-agent channel establishment. The throughput for this scenario will be denoted by \( T_2 \). These two cases are of interest only for agent-to-agent channel establishment.

Figure 5.14 shows the latency and throughput values for the following cases: (1) ADP (2) direct establishment of agent-to-agent channels in the TCP trace, and (3) direct establishment of agent-to-agent channels in the NFS trace. From the figure, we conclude that the performance gains of ADP over both instances of agent-to-agent channel establishment are substantial. The inferior performance of agent-to-agent channel establishment approaches in the table is to be attributed entirely to the much higher rate of PKE-based channel establishment operations that these approaches require with respect to that caused by ADP, since even the ADP experiments were performed without piggybacking. PKE-based channel establishment is quite time-consuming: we have measured in our system an average establishment time of 1.75 seconds.

In the agent-to-agent channel establishment experiments, the TCP trace had 76 new connection establishments, and the NFS trace had 41 new RPC transactions from new processes, both in a sequence of 10,000 messages. Corrections introduced to remove edge effects reduced
<table>
<thead>
<tr>
<th>Latency (ms)</th>
<th>Throughput T1 (kB/s)</th>
<th>Throughput T2 (kB/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCP</td>
<td>NFS</td>
<td>TCP</td>
</tr>
<tr>
<td>Subtransport</td>
<td>6 8</td>
<td>90 325</td>
</tr>
<tr>
<td>Transport</td>
<td>30 42</td>
<td>76 305</td>
</tr>
</tbody>
</table>

Figure 5.14: Table and histogram showing latency and throughput values for TCP and NFS traces with subtransport and transport approaches to channel establishment. T1 is the throughput with constant process creation rate. T2 is the throughput with a process creation rate linearly increasing with the arrival rate.
the values of the numbers of new connections slightly in the latency and T2 experiments. In the T1 experiments, there were 7 new connections in the TCP trace and 4 new RPC transactions in the NFS trace.

Figure 5.15 confirms the conjectures made at the beginning of this section about the effect of piggybacking in ADP. The ALL input trace was used. Among the many experiments we performed, an interesting one was that intended to determine the variation of the latency as the arrival rate of messages in the input trace was progressively increased. Figure 5.15 shows that the effect of piggybacking is insignificant at low arrival rates. Without piggybacking, an increase in the message arrival rate causes a rapid increase in the latency, whereas with piggybacking the latency starts increasing for much higher arrival rates. Figure 5.16 shows the latencies, throughputs and CPU utilizations for an unmodified client message trace (ALL) with an average message arrival rate of 250 messages/s, for the following cases: (1) without any channel establishment mechanisms, (2) without channel establishment mechanisms but with message piggybacking, (3) with ADP channel establishment but without message piggybacking, and (4) with ADP channel establishment and with message piggybacking. The difference in performance between cases 1 and 2 is considerable. The performance of case 4 is very close to that of case 2, whereas the performance of case 3 is less than that of case 1. This shows that message piggybacking can keep the performance cost of channel establishment very small.

The results of our experiments therefore show that the performance gains of ADP, due both to the reduction in the total overhead of channel establishment and to the advantages of piggybacking, are indeed substantial.

5.9. Conclusion

Trust requirements can be traded for performance of channel establishment protocols. If channel composition is PKE-based, slightly increasing the trust requirements allows agent-to-agent channels to be built on top of host-to-host channels. This host-to-host approach to channel establishment can greatly increase the performance of agent-to-agent secure communication. The accompanying increase in trust requirements is still satisfied in the distributed system's name space. However, if channel composition is SKE-based, this approach requires global trusts which may not be satisfied in the system's name space. Protocols for establishing host-to-host channels can be handled at the subtransport level of a network protocol hierarchy. Experimental measurement of a prototype of a host-to-host channel establishment protocol confirms its expected performance advantages.
Figure 5.15: Effect of piggybacking on latency with increasing message arrival rates (ALL and TCP traces; piggybacking is denoted by $Q=\text{on}$)
No security mechanisms  
No piggybacking

No security mechanisms  
Piggybacking present

Subtransport security  
No piggybacking

Subtransport security  
Piggybacking present

<table>
<thead>
<tr>
<th></th>
<th>Latency (ms)</th>
<th>Throughput (kB/s)</th>
<th>CPU Overhead (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Without Security</td>
<td>67</td>
<td>350</td>
<td>39.9</td>
</tr>
<tr>
<td>With Security</td>
<td>11</td>
<td>760</td>
<td>20.09</td>
</tr>
<tr>
<td>Without Security</td>
<td>107</td>
<td>328</td>
<td>59.79</td>
</tr>
<tr>
<td>With Security</td>
<td>11</td>
<td>750</td>
<td>20.3</td>
</tr>
</tbody>
</table>

Figure 5.16: Table and Histogram showing performance effects of piggybacking (ALL trace)
CHAPTER 6

CONCLUDING REMARKS

6.1. Conclusion

We have developed an axiomatic theory of trust in distributed systems. Our theory of trust is based on modal logics of belief. Any well formed formula assumed to be valid in addition to the axioms of the logic is considered as a trust specification. This gives a lot of power and generality in expressing trust relationships. We have presented systematic methods for synthesizing protocols that are necessary and sufficient for implementing a given trust specification in a distributed system.

Trust arises primarily in establishing channels for secure communication. The only way to establish a new channel is by composing a sequence of existing adjacent channels. There are two kinds of channels: independent channels, which are provided at system configuration and do not have any trust requirements, and dependent channels, which are composed from independent channels and have trust requirements. Channel composition mechanisms are commonly based on either public key encryption (PKE) or single key encryption (SKE). PKE-based channel composition requires ternary trust relationships known as authenticity trusts. SKE-based channel composition has much larger trust requirements than PKE-based channel composition. The differences in the trust requirements of PKE and SKE-based channel compositions translate into several advantages of PKE-based over SKE-based channel composition with regard to replication, caching, permanence of trust requirements, and so on. Within each channel composition mechanism, the trust requirements are not symmetric with regard to the agents involved in the mechanism. Our analyses provide insight into the basic structure and limitations of mechanisms with regard to their trust requirements.

In a distributed system, it is desirable to have a tree of independent channels. It is convenient for the tree of independent channels to represent also the global name space of the system. There are two channel composition orders, namely, iterative and recursive. Iterative and recursive channel composition orders require different trusts and exhibit interesting duality properties. As one of the most important applications of a formal theory of trust, we have developed polynomial-time algorithms for synthesizing name spaces so that, given a channel composition order, and the trust relationships among agents, PKE-based channel composition between any two agents requires only a subset of the given set of trust relationships. The trust specifications are in general functions of three agents, but can also be functions of two agents, in which case the algorithms become simpler. Each node in a name space has to store the database of the public keys of its children, and it is desirable to put upper and lower bounds on the size of this database. However, the problems of putting upper or lower bounds on the number of children of each node in a name space are NP-complete.

The polynomial-time name space synthesis algorithms have been implemented and experimented with. Sample runs of these algorithms show that small differences in trust relationships can cause substantial differences in name spaces, thus demonstrating the practical usefulness of these algorithms.
No synthesis is complete without performance considerations. Trust requirements and performance of channel establishment mechanisms can be traded for each other. If the channel composition is PKE-based, slightly increasing the trust requirements allows agent-to-agent channels to be built on top of host-to-host channels. This host-to-host approach can greatly increase the performance of agent-to-agent secure communication. The accompanying increase in trust requirements is still satisfied in the distributed system name space. However, if channel composition is SKE-based, this approach requires global trusts, which may not be satisfied in the system's name space. Protocols for establishing host-to-host channels can be handled typically in the top portion of the network layer or in a special subtransport layer of a network protocol hierarchy. The experimental measurement of a prototype of a host-to-host channel establishment protocol confirms its expected performance advantages.

6.2. Future Work

In Chapter 2, we showed that the implementation of trust specifications requires constraints on message transactions among agents. Many of these constraints are of the form: send/receive a particular message before/after sending/receiving some other message. A more general approach to representing such constraints would involve temporal reasoning. We propose to investigate the possibility of combining temporal reasoning with modal logic for this purpose.

The trust relationships considered in Chapter 3 were functions of at most three agents. Trusts can also involve more than three agents. Figure 6.1 shows a scenario in which a trust involves four agents. Suppose it is known that, when IBM queries IBM-J for the key of ibaraki, IBM-J either returns the correct key of ibaraki or collaborates with jap to return jap's key. When IBM-J returns a key key1 in answer to IBM's query for ibaraki's key, IBM obtains jap's key key2 through an independent path, namely, IBM-USA-I-jap. IBM then compares key1 and key2, and accepts key1 as ibaraki's key only if the two keys are not identical. Here a trust relationship involving IBM, IBM-J, ibaraki and jap is necessary. We propose to study the formalization of such complex trust relationships that may involve more than three agents.

In Chapter 4, we developed algorithms for synthesizing global name spaces from trust specifications. Additional considerations, such as fault tolerance, may be used in these synthesis algorithms. In a fault-tolerant name space, some minimum number of trust relationships must become false before the name space becomes disconnected. Redundancies in trusts will have to be used in constructing such fault-tolerant name spaces. Another desirable feature of name spaces is the localization of dynamic reconfiguration when trust relationships change.

In Chapter 5, we outlined the trust requirements involving user processes and their local hosts. An interesting extension would be to study trust relationships involving user processes and remote hosts, such as those involving a client and a server's kernel, or a server and a client's kernel.
Figure 6.1: Illustration of a trust that involves more than three agents
BIBLIOGRAPHY

[AHU74]  
A. Aho, J.E. Hopcroft and J.D. Ullman.  
The Design and Analysis of Computer Algorithms.  
Addison-Wesley, Reading, MA., 1974.

[Akl83]  
Selim G. Akl.  

[Akl]  
Selim G. Akl.  
On The Security of Compressed Encodings.  

[AnV87]  
David P. Anderson and P. Venkat Rangan.  
A Basis for Secure Communication in Large Distributed Systems.  

[AFV87a]  
David P. Anderson, Domenico Ferrari, P. Venkat Rangan and Shin-Yuan Tzou.  
The DASH Project: Issues in the Design of Very Large Distributed Systems.  

[AFV87b]  
David P. Anderson, Domenico Ferrari, P. Venkat Rangan and Bruno Sartirana.  
A Protocol for Secure Communication in Large Distributed Systems.  

[AFV87c]  
David P. Anderson, Domenico Ferrari and P. Venkat Rangan.  

[AFV87d]  
David P. Anderson, Domenico Ferrari, P. Venkat Rangan and Bruno Sartirana.  

[BLN82]  
A. D. Birrell, R. Levin, R. Needham and M. D. Schroeder.  
Grapevine: An Exercise in Distributed Computing.  
[BiN84]  
Implementing Remote Procedure Calls.  

[Bir85]  
A. D. Birrell.  
Secure Communication using Remote Procedure Calls.  

[BLN86]  
A. D. Birrell, B. W. Lampson, R. M. Needham and M. D. Schroeder.  
A Global Authentication Service without Global Trust.  

[CGH81]  
Verifying Security.  

[Che84]  
D. R. Cheriton.  
The V Kernel: A Software Base for Distributed Systems.  

[Che86]  
D. R. Cheriton.  

[Den82]  
D. E. Denning.  

[Den84a]  
D. E. Denning.  
Cryptographic Checksums for Multilevel Database Security.  

[Den84b]  
Dorothy E. Denning.  
Digital Signatures with RSA and Other Public-Key Cryptosystems.  

[DiH76]  
W. Diffie and M. Hellman.  
New Directions in Cryptography.  

[Dif82]  
W. Diffie.  
Conventional Versus Public Key Cryptosystems.  

[Dif85]  
W. Diffie.
Security for the DoD TCP.


[EKW74]
A. Evans, W. Kantrowitz and E. Weiss.
A User Authentication Scheme Not Requiring Secrecy in the Computer.

[FNS75]
H. Feistel, W.A. Notz and J.L. Smith.

[Fer78]
D. Ferrari.
Computer Systems Performance Evaluation.

[GaJ79]
M.R. Garey and D.S. Johnson.

[HaM85]
J. Y. Halpern and Y. Moses.
A Guide to the Modal Logics of Knowledge and Belief: Preliminary Draft.
Proc. 9th International Joint Conference on AI, pages 479-490, Los Angeles, California, 1985.

[Hin62]
J. Hintikka.
Knowledge and Belief.

[Ken77]
S. T. Kent.
Some Thoughts on TCP and Communication Security.
Local Network Note, MIT, Laboratory for Computer Science, May 1977.

[KIP79]
C. S. Kline and G. J. Popek.
Public key vs. conventional key encryption.

[Koc]
M. Kochanski.
Developing an RSA Chip.
Advances in Cryptology: Proceedings of Crypto' 85.

[Kon84]
K. Konolige.
A Deduction Model of Belief and its Logics.

[Kon86]
K. Konolige.
J.Y. Halpern, editor.
What Awareness isn't: A Sentential View of Implicit and Explicit Belief.

[Kri63]
S. A. Kripke.
Semantical Considerations on Modal Logics.

[Lan81]
C. E. Landwehr.

[Lu86]
W. P. Lu and M. K. Sundareshan.
A Hierarchical Key Management Scheme for End-to-End Encryption in Internet Environments.

[Men87]
E. Mendelson.
Introduction To Mathematical Logic.

[Mor86]
L. Morgenstern.
J.Y. Halpem, editor.
A First Order Theory of Planning, Knowledge, and Action.

[MuT84]
S. J. Mullender and A. S. Tanenbaum.
Protection and Resource Control in Distributed Operating Systems.

[NBS77]
NBS.
Data Encryption Standard.

[NeS78]
R. M. Needham and M. D. Schroeder.
Using encryption for authentication in large networks of computers.

[PoK79]
Gerald J. Popek and Charles S. Kline.
Encryption and Secure Computer Networks.

[RSA78]
R. L. Rivest, A. Shamir and L. Adleman.
A method for obtaining digital signatures and public-key cryptosystems.
[Sal74]  
J. Saltzer.  
Protection and the Control of Information Sharing in Multics.  

[SRC84]  
J.H. Saltzer, D.P. Reed and D.D. Clark.  
End-To-End Arguments in System Design.  

[SJR86]  
Extending a Capability Based System into a Network Environment.  

[STB86]  
R.E. Schantz, R.H. Thomas and G. Bono.  
The Architecture of the Cronus Distributed Operating System.  

[SIP86]  
E.L. Slate and J.A. Popko.  
The Next Five Years in Communications.  

[Tan81]  
A.S. Tanenbaum.  
Network Protocols.  

[Tan88]  
A.S. Tanenbaum.  
*Computer Networks*.  

[TPR84]  
The Berkeley Internet Name Domain Server.  

[Ter]  
D.B. Terry.  
Distributed Name Servers: Naming and Caching in Large Distributed Computing Environments.  
*Doctoral Dissertation*, Report No. 85/228, Computer Science Division, University of California.

[VeA87]  
P. Venkat Rangan and Ron Ashany.  
Application of AI to Secure Communication in Distributed Systems.  
[Ven88]
P. Venkat Rangan.
An Axiomatic Basis of Trust in Distributed Systems.

[VoK83]
V.L. Voydock and S.T. Kent.

[Wei69]
C. Weissman.
Security Controls in the ADEPT-50 Time-Sharing System.

[81a]
*Information Sciences Institute, University of Southern California*, September 1981.

[81b]
*Information Sciences Institute, University of Southern California*, September 1981.
### APPENDIX A

**Trust Specifications for Name Space Design Example of Section 4.9**

The sample set of trust specifications used in the example of Section 9 of Chapter 4 are as follows. For brevity, only those trust relationships that are assumed to be true, are enumerated.

<table>
<thead>
<tr>
<th>(T(IBM-1, IBM, ARC)) = true</th>
<th>(T(Jap, IBM, ARC)) = true</th>
</tr>
</thead>
<tbody>
<tr>
<td>(T(Sony, IBM, ARC)) = true</td>
<td>(T(Sony-USA, IBM, ARC)) = true</td>
</tr>
<tr>
<td>(T(IBM-1, ARC, IBM)) = true</td>
<td>(T(Jap, ARC, IBM)) = true</td>
</tr>
<tr>
<td>(T(Sony, ARC, IBM)) = true</td>
<td>(T(Sony-USA, ARC, IBM)) = true</td>
</tr>
<tr>
<td>(T(ARC, IBM, IBM-1)) = true</td>
<td>(T(Jap, IBM, IBM-1)) = true</td>
</tr>
<tr>
<td>(T(Sony, IBM-1, IBM)) = true</td>
<td>(T(Sony-USA, IBM-1, IBM)) = true</td>
</tr>
<tr>
<td>(T(Jap, ARC, Sony-USA)) = true</td>
<td>(T(Jap, ARC, Sony-USA)) = true</td>
</tr>
<tr>
<td>(T(ARC, IBM-1, Jap)) = true</td>
<td>(T(IBM, IBM-1, Jap)) = true</td>
</tr>
<tr>
<td>(T(Sony, IBM-1, Jap)) = true</td>
<td>(T(Sony-USA, IBM-1, Jap)) = true</td>
</tr>
<tr>
<td>(T(ARC, IBM-1, Sony)) = true</td>
<td>(T(IBM, IBM-1, Sony)) = true</td>
</tr>
<tr>
<td>(T(Jap, IBM-1, Sony)) = true</td>
<td>(T(Sony-USA, IBM-1, Sony)) = true</td>
</tr>
<tr>
<td>(T(ARC, Sony-USA, Jap)) = true</td>
<td>(T(IBM-J, Sony-USA, Jap)) = true</td>
</tr>
<tr>
<td>(T(IBM, Sony-USA, Jap)) = true</td>
<td>(T(Sony, Sony-USA, Jap)) = true</td>
</tr>
<tr>
<td>(T(ARC, Sony-USA, IBM-J)) = true</td>
<td>(T(IBM, Sony-USA, IBM-J)) = true</td>
</tr>
<tr>
<td>(T(Jap, Sony-USA, IBM-J)) = true</td>
<td>(T(Sony, Sony-USA, IBM-J)) = true</td>
</tr>
<tr>
<td>(T(ARC, Sony-USA, Sony)) = true</td>
<td>(T(IBM-J, Sony-USA, Sony)) = true</td>
</tr>
<tr>
<td>(T(IBM, Sony-USA, Sony)) = true</td>
<td>(T(Jap, Sony-USA, Sony)) = true</td>
</tr>
</tbody>
</table>