Modelling of cryptographic protocols
A concurrency perspective

FINAL DRAFT

Draft 4 December 1997 3:31 pm, 149 pages

Copyright © 1996, 1997 Pekka Nikander

All rights reserved.

Abstract

Understanding the real behaviour of a cryptographic protocol is a very demanding task. This is mainly due to the immense amount of possible actions, some of which are progressing in parallel, that may happen during even a single protocol run. If one takes into consideration several, concurrently running but interacting protocol sessions, even the modelling problem becomes almost unsolvable.

This is a literature study that attempts to collect and present the most important factors of modelling and verification of cryptographic protocols in an easily comprehensible and relatively concise form. In a way, this work covers over one hundred articles and conference papers published in the area of cryptographic protocols, plus quite a lot of introductory material.

The approach taken is aimed for the uninitiated reader, having basic background in communications protocols and cryptography, but not necessarily any deeper notion of applying cryptography in communications. In particular, it is assumed that the reader may not be familiar with modal logic or process algebra. Thus, these basic tools are covered in somewhat length.

Most of the protocol modelling approaches covered by this work can be roughly classified in two categories. One thread of research was forstered from the so called BAN logic, a form of so called authentication logic introduced by Burrows, Abadi and Needham back in 1989. This branch includes a number of modal logics that are usually denoted with cryptic acronyms like AT, GNY or SvO. The origins of the other offshoot, or the research based on utilization of process algebraic formalisms, cannot be identified that easily. It seems to base on modelling of all kinds of communication protocols, security considerations being only a relatively new development. However, only the basics of this latter approach has been covered by this work. The interested reader should probably also refer directly to the newest papers. Good sources of information are the proceedings of the annual IEEE Symposium on Security and Privacy, as well as the annual IEEE Computer Security Foundations Workshop.

This work consists of three parts and a number of appendices. Part I presents the basic background. It contains introductory discussions about distrubuted systems security, cryptography, cryptographic protocols, modelling of communication protocols, and protocol flows as well as the goals that are attempted to be achieved by executing cryptographic protocols. Part II contains introductions of modal logic and process algebra. These presentations are both tailored towards the problem domain; therefore, they cannot be recommended as generic introductions to the formalisms. Part III contains a comparison of the various BAN logic based approachies, and gives some initial ideas how modal logic and process algebra based approaches could be unified by using a fairly general underlying semantic model. Within this discussion the built-in problems involved with comprehending encrypted data and trusting integrity protected data is highlighted. Finally, the appendices include a simplified protocol example, a brief discussion about a typical, rather complex real life protocol, and a summary of the formulae defined in the various BAN variations.

Foreword and Acknowledgements

The history of this work stems back to the end of 1994 when I became fascinated by the idea of encrypted communications. At that time, I pretty soon realised that implementing cryptographic protocols is considerably harder than implementing just plain protocols. Starting from this initial understanding, I started my study of the field. It became almost immediately clear that there was few if any comprehending works about modelling and verification of cryptographic protocols. On the other hand, lots of information appeared to be available in the form of scientific papers and other short publications. This work tries to ease the situation a little bit.

The real foundation for this work was laid in Spring 1996 during my five week stay at the Isaac Newton Institute for Mathematical Sciences at Cambridge University. I appreciate that Ross Anderson, Tom Berson and Peter Landrock took the burden of arranging the Computer Security, Cryptology and Coding Theory programme in the first hand. Without Kaisa Nyberg taking the action of arranging me the possibility for the participation, this work would have never achieved the level of insight presented.

During my stay in Cambridge I was happy to join several intriguing discussions with Matt Blaze, Li Gong, Richard Kemmerer, Kathy Meadows, Birgit Pfitzmann, and Paul Syverson, to name just a few of all the wonderful people that were there. Also, I am deeply in debt to Shareen and Ross Anderson and Dorothy and Tom Berson in another respect. With their generous help I was able to take part in the lively social life that was going on. That gave me the opportunity to meet and discuss with some of the most knowledgable and experienced people in the area of cryptography and computer security.

During the lengthy process it has taken to get all this written, I have received constant support from my supervisor Arto Karila. I am grateful for that. I also want to thank Petri Aukia, Oiva Karppinen, and Jonna Partanen, as well as the rest of the staff of Nixu Oy for their interest and support during this work. Especially, I want to thank Lea Viljanen; without her help the last few weeks of proofreading and fixing small mistakes would have been much more painful.

Finally, I want to thank Kirsi Sarvas for her patience and support. With her love and caring everything has been a lot easier.

Helsinki, December 1997

Pekka Nikander

I gratefully acknowledge the funding received from MATINE under contract number XXX.

Table of contents

Abstract i
Foreword and Acknowledgements iii

PART I

CHAPTER 1 *Motivation* 3

Organization of the material 4
Notation 5

CHAPTER 2 *Background* 7

Distributed system security 8
Distributed systems — a definition 8
Security functions 9
OSI security services 12
CORBA security architecture 12
Security contexts and associations 14
Discussion 15
Cryptology 15
Symmetric cryptosystems 16
Public key cryptosystems 17
One-way hash functions and digital signatures 17
Random number generation 18
Security of cryptosystems 18
Cryptographic protocols 20
Modelling of communication protocols 21
Expressing protocol goals 23
Models, states and actions 24
Logical formulae 25
A protocol example 26
The protocol 26
Run based protocol specification 27
Protocol model 28
LTS, ACP and CSP specifications 29
Knowledge and beliefs 30
Cryptanalysis and protocol failures 31
Flaws in protocols 32

Elementary flaws 32
Password-guessing flaws 32
Freshless flaws 32
Oracle flaws 32
Type flaws 33
Internal flaws 33
Cryptosystem-related flaws 33
Lessons learned 33
Summary 33

CHAPTER 3 *The purpose of cryptographic protocols* 35

Introduction 35
High level protocol goals 36
System integrity and data confidentiality 36
Authorization, audit trail and intrusion detection 37
Peer identification 37
Authentication and non-repudiation 38
Intermediate level protocol goals 39
Key agreement, confirmation, freshness and secrecy 39
Correspondence and secrecy 40
Message and data item level goals 40
Message integrity and identification of the originator 40
Confidentiality 41
Freshness, timeliness, nonces and replay prevention 42
Redundancy at message level 43
Summary 44

PART II

CHAPTER 4 *Modal logic* 47

Syntax and semantics 48
Frames and models 48
Truth value of formulae 50
Truth and validity 51
Logics, proofs and consistency 51
Logics 51
Theorems, axioms and provability 52
Soundness and completeness 52
Consistency 53
Some standard logics 53
Well-known axioms 53

The logic S5 54
The logics KT4 and KD45 54
Logics of knowledge and belief 55
Possible-worlds interpretation of protocol models 56
Knowledge of different agents 57
Knowledge, common knowledge and distributed knowledge 58
Belief as conditional knowledge 59
Temporal logic 60
Linear time temporal logic 60
Branching time temporal logic 61
Combining knowledge and time 63
Summary 68

CHAPTER 5 *Model checking and Process Algebra* 71

Introduction to models of concurrency 72
States, actions and events 72
Communication 73
Deadlocks and divergencies 73
Abstraction and hiding 74
Traces and equivalencies 74
Process graphs and Labelled Transition Systems 74
Process graphs 74
Examples 75
Labelled transition systems 76
Algebraic approaches 76
ACP — Asynchronous Communicating Processes 76
CCS — Calculus of Communicating Systems 81
CSP — Communicating Sequential Processes 83
Semantics 85
Graph isomorphism 85
Traces 86
Strong and weak bisimulation 86
Observational congruence, or rooted τ-bisimilarity 88
Branching bisimulation 89
Handling divergencies 89
Failures-divergencies semantics 90
Model checking 91
Tackling state space explosion 91
Comparing process models 92
Visualisation approaches 92
Checking validity of formulae 93
Summary 93

PART III

CHAPTER 6 *Comparison of some BAN-based approaches* 97

Introduction to the selected papers 97
The original BAN logic by Burrows, Abadi and Needham . 98
CKT5 by Bieber 99
The GNY logic of Gong, Needham and Yahalom 101
The Abadi-Tuttle (AT) logic 102
Towards unified semantics: SvO 102
Adding time by Paul Syverson 103
Yet another approach: AUTLOG by Wedel and Kessler .. 104
Comparison of syntactic approaches 104
Differences in the semantic approaches 106
Models of computation 106
Adding time 108
A set of beliefs vs. beliefs based on possible worlds relations 108
Idealization vs. explicit recognition of messages 109
Summary 110

CHAPTER 7 *Future directions* 111

Process algebras and protocol models 111
Action vs event based models 112
Temporal and modal interpretation 112
Summary 114

CHAPTER 8 *Conclusions* 115

APPENDIX A *A protocol example* 117

The protocol 117
Actions 118
Run based protocol specification 120
Protocol model 121
LTS, ACP and CSP specifications 122
Knowledge and beliefs 125

APPENDIX B *ISAKMP / Oakley — A real world example* 127

ISAKMP framework 127
Establishing the initial association: base exchange 128
Using Oakley to establish the initial association 130
Defining an Internet AH/ESP association: Oakley Quick Mode 130

APPENDIX C *Rules in the modal approaches* 133

BAN-logic (Burrows, Abadi, Needham) 133
Beliefs 133
Saying (writing, sending) 133
Seeing (receiving, reading) 133
Message authentication 134
Freshness 134
Jurisidiction 134
Key derivation and generation 134
GNY logic(Gong, Needham, Yahalom) 134
Reasoning rules 134
Seeing (reading, receiving) axioms 134
Possession axioms 135
Freshness axioms 135
Recognition axioms 135
Interpretation axioms 136
Jurisdiction axioms 137
AT-logic (Abadi and Tutle) 137
Reasoning rules 137
Belief axioms (modalities) 137
Message authentication axioms 137
Seeing (reading, receiving) 137
Saying (writing, sending, meaning) 137
Jurisdiction 137
Freshness 138
Key derivation and generation 138

SvO-logic (Syverson and van Oorshot) 138
Reasoning rules 138
Believing 138
Message authentication 138
Key agreement 138
Receiving (seeing, reading) 138
Seeing 138
Comprehending 139
Saying (writing, sending, meaning) 139
Jurisdiction 139
Freshness 139
Nonce verification 139
Goodness of keys 139
Having 139
Wedel-Kessler logic (AUTLOG) 139
Reasoning rules 139
Modalities 139
Jurisdiction axioms 139
Posession axioms 140
Recognition axioms 140
Freshness axioms 140
Seeing (receiving, reading) 140
Nonce verification 140
Saying (sending, writing, meaning) 140
Authentication and key confirmation axioms 140
Comprehension axioms and localization equivalences ... 141
Localization equivalence axioms 141
Key derivation and generation axioms 141

APPENDIX D *References* 143

PART I

CHAPTER 1 *Motivation* *3*

Organization of the material 4
Notation 5

CHAPTER 2 *Background* *7*

Distributed system security 8
Cryptology 15
Cryptographic protocols 20
Modelling of communication protocols 21
A protocol example 26
Cryptanalysis and protocol failures 31
Flaws in protocols 32
Summary 33

CHAPTER 3 *The purpose of cryptographic protocols* *34*

Introduction 35
High level protocol goals 36
Intermediate level protocol goals 39
Message and data item level goals 40
Summary 44

CHAPTER 1

Motivation

Experience has shown that designing reliable and correctly functioning communication protocols is hard. Designing secure communication protocols is even harder. The reason for this is that while non-cryptographic protocols are designed to recover from random or statistical errors, cryptographic protocols must be able to securely recover when a malicious, deliberate attack is being conducted by an untrustworthy communication party, or some number of such parties. In spite of this difference, most of the problems encountered are more or less the same. However, it appears to be the case that the designer of a cryptographic protocol must be far more rigorous in specifying the initial assumptions, goals and failure models than a colleague working with "normal" communication protocols.

Besides the designer, also the implementor often needs much more insight to the protocol design than his or her non-crypto colleague. There are numerous reported incidents where an apparently innocent neglect, an unfortunate implementation choice, or a simple misunderstanding has led to disastrous results. For example, the initial implementation of the Internet Secure Socket Layer (SSL) protocol used a cryptographically weak random number generator as a supply for session keys. Thus, even though the protocol appeared to be secure in most other respects, the weak random numbers used allowed an attacker to easily determine a relatively small range of possible key values. In fact, the set was small enough so that the actual key value could be determined in mere seconds by trial-and-error [42].

The diversity of the security needs of distributed systems and the difficulty of protocol design has led to a number of different approaches to solve, model and reason about the security of communication protocols. In this study we concentrate on the modelling and verification of communication protocols that are designed to be used in a hostile environment and that take a black box approach towards the cryptographic algorithms used. That is, we explicitly leave out all distributed systems where the specific details of the cryptosystem applied play a crucial role in the correct behaviour of the system.

Furthermore, we concentrate only on approaches that either use a modal logic approach (i.e. the BAN logic [20] or a similar approach) or a model checking approach. In the latter case, our intention is to apply process algebraic methods to the models used. When beginning this work, this choice was not obvious. However, lately the good results reported by Kevin Lowe and others on using CSP and its derivatives [68] show that the approach is, indeed, worthwhile. Furthermore, we have tried

to map some of the common and corresponding aspects between the modal logic and model checking approaches. The few noteworthy results are collected in Chapter 7.

Thus, the purpose of this study is to collect together various results achieved in modelling and verification of cryptographic protocols using modal logic and process algebraic approaches. The aim is to give the uninitiated reader an overview of the field, the necessary theoretic background to understand the formalisms used, and an up-to-date perspective to the research being conducted. It is assumed that the reader has a basic working knowledge in theoretical computer science, cryptography, distributed systems and communication protocols. No prior knowledge of modal logic or process algebra is assumed.

While this is a descriptive presentation reporting work conducted by others, towards the end we present a comparison of the two families of methods considered. In addition, an outline of a unified protocol model that can be used as a semantic model for both modal logic and process algebraic formulae is given.

The reader should note that while mostly being a theoretical study, the emphasis of this presentation is on the practical applicability of the methods described. Therefore, in Chapters 4, “Modal logic” and 5, “Model checking and Process Algebra” where most of the theoretic background is given, few proofs are presented. Similarly, those aspects of the theories that have no apparent practical application and that are not necessary for later development have been left out or only lightly touched. On the other hand, there are a number examples binding the theory described to the practical aspects of protocol design and implementation.

1.1 Organization of the material

In Chapter 1, this chapter, we introduce the problem domain and notation used. The next chapter, Chapter 2, “Background”, outlines the basic ideas of security in distributed systems, gives a brief introduction to cryptology as it applies to the issues being studied, gives basic notion of cryptographic protocols in general, introduces the ideas of protocol modelling, and briefly describes and discusses the difference between cryptographic attacks and protocol failures.

Chapter 3, “The purpose of cryptographic protocols”, on page 35, contains a description of cryptographic protocols and protocol goals. Theoretical aspects of protocol security as well as protocol goals on various levels are discussed.

Chapters 4, “Modal logic” and 5, “Model checking and Process Algebra”, introduce modal logics and process algebraic theories. In particular, the logics of knowledge and belief as well as temporal logics are discussed. The problems of combining both knowledge and temporal operators within a single system is briefly touched. In addition to Bergstra and Klop’s ACP process algebra, Milner’s CCS and Hoare’s CSP are described. Various equivalence and refinement relations are described, emphasis being on those that have probable practical applicability. The problem of state space explosion is described along with some attempts to solve it.

Chapter 6, “Comparison of some BAN-based approaches” describes work reported in literature. In Chapter 7, “Future directions”, a semiformal protocol modelling framework is outlined, describing how temporal and modal logic formulae could be interpreted in this framework and how different kinds of process algebraic semantics could be applied to the framework.

Chapter 8, “Conclusions”, collects the results of this study into a concise form.

1.2 Notation

Some of the more usual notations used in this study are summarized in Table 1, below. In addition to the formulae shown here a large number of other formulae are also used. However, they are typically used only in one part of this study, and explained when first used.

TABLE 1. Notations

Notation	Explanation
Generic notations	
$\langle e_1, \ldots, e_n \rangle$	An ordered set (a tuple) of the elements $e_1, \ldots, e_n$
$\{e_1, \ldots, e_n\}$	An (unordered) set of the members $e_1, \ldots, e_n$
$x \in S$	x is a member of (belongs to) the set S
$F : S_1 \to S_2$	A function F whose domain (source) is S_1 and codomain (target) is S_2
$\forall, \exists$	Quantifiers, used in semiformal examples[a]
Protocol models	
Alice, Bob, …	Protocol parties, see Table 4 on page 21 for more information
$S \to R : m$	The party S sends a message m to the receiver R who receives it
$\mathrm{send}(S, R, m)$	The party S sends a message m to the receiver R
$\mathrm{receive}(R, m)$	The party R receives the message m
Logical formulae	
$\varphi, \psi, \ldots$	Logical variables, i.e. placeholders for arbitrary logical formulae
$\top, \bot$	Verum (true) and falsum (false)
$\neg\varphi$	Negation of φ
$\varphi \wedge \psi$	Conjunction of φ and ψ
$\varphi \vee \psi$	Disjunction of φ and ψ (i.e. φ or ψ)
$\varphi \to \psi$	Logical implication, i.e. if φ then ψ
$\varphi \leftrightarrow \psi$	Logical equivalence
Modal and temporal formulae	
$\Box\varphi$	Necessitation, or any corresponding modal operator. See Chapter 4, “Modal logic”, on page 47. “Always (in the future)” when used in temporal sense.
$\Diamond\varphi$	Possibilitation, or any corresponding modal operator. “Sometimes (in the future)” when used in temporal sense.
$\Diamond\!\!-\varphi$	A temporal operator “at least once in the past”.
$\Box\!\!-\varphi$	A temporal operator “always in the past”.
Knowledge formulae	
$\mathrm{Role}\ \mathsf{knows}\ \varphi$	The party acting in Role knows φ
$\mathrm{Role}\ \mathsf{believes}\ \varphi$	The party acting in Role believes that φ

TABLE 1. Notations

Notation	Explanation
Process algebraic formulae	
xy *(concatenation)*	Sequential composition
$x + y$	Alternative composition
$x \mid\mid y$	Parallel composition

a. The actual logical theory developed is based propositional, not predicate logic. Therefore the quantifiers are used only in order to express semiformal formulae and facts, and not as a part of the actual formal treatment.

Note on the use of the gender referring pronouns. In this study, we have always assumed that the *first* protocol party mentioned is Alice, i.e. *she*. The *second* protocol party, in turn, is Bob, and therefore *he*. When there are more parties involved, we assume that both female and male actors are playing the game.

CHAPTER 2

Background

In this chapter we give the necessary background needed to understand the rest of this work. First, in Section 2.1, “Distributed system security”, we describe the terminology and goals of distributed systems security in general. After a definition (Section 2.1.1) we will discuss the security functions needed to achieve network security (Section 2.1.2), the OSI security model (Section 2.1.3) and the CORBA object oriented security model (Section 2.1.4). Possibly the most noteworthy detail here is our discussion of the semantics of the word “authentication”, on page 11. After the CORBA security model, in Section 2.1.5, we will define the concepts of security context and associations, which play a crucial role as a semiformal outline model for several security protocols.

The second major section, Section 2.2, “Cryptology”, is a fairly standard brief introduction to cryptology in general. In this work, we will not go very deep into the details of cryptology, and the level of discussion here reflects that. Thereafter, in Section 2.3, “Cryptographic protocols”, we briefly introduce the idea of cryptographic protocols. A deeper discussion is deferred until Chapter 3, “The purpose of cryptographic protocols”.

Section 2.4, “Modelling of communication protocols”, works as an introduction to the main subject of this work. Here we introduce the basic concepts of models (Section 2.4.2) and the way logical formulae can be used to express facts about protocols (Section 2.4.3).

To give the uninitiated reader a better intuition of the modelling aspects, the next section, Section 2.5, “A protocol example”, gives a brief example of modelling a toy protocol. This section is intended to work as an orientation basis for understanding the later chapters.

The chapter is concluded with two sections, Section 2.6, “Cryptanalysis and protocol failures”, on page 31 and Section 2.7, “Flaws in protocols”, on page 32, that discuss the sorts of problems, failures and flaws that have been found in cryptographic protocols. The purpose is to give a glimpse of some of the more delicate aspects of protocol security.

2.1 Distributed system security

Cryptographic protocols are designed to provide security in distributed systems. Whereas ordinary communication protocols enable the parts of a distributed system to communicate with each other (thereby creating a distributed system to begin with), a cryptographic protocol enables the parts to communicate *in a secure way.* Note that the usage of the indefinite form here is intentional: there are several meanings for the word *security*, as we shall see.

In fact, the meaning of the word *security*, or how to reach security, is not at all straightforward or simple. To make our intentions clear, we next present a *model for distributed systems.* Thereafter we briefly define the meaning of some of the most common security terms such as *availability*, *confidentiality*, *integrity*, *identification*, *authorisation*, *delegation*, *authentication* and *non-repudiation*. The OSI Security model is described as a comparison to our model. Finally, we will discuss how our definitions differ from the other definitions used in literature, and why.

2.1.1 Distributed systems — a definition

A distributed system is a computer system that consists of a number of *nodes*, which communicate with each other through a *network*. The nodes are typically general purpose computers; they have the capability to perform computations and store limited (but possibly large) amounts of information. Sometimes the nodes are referred to as *agents* or protocol *parties*. The word agent is commonly used when discussing the (intensional[1]) knowledge or beliefs a node can be thought to have, and the word (protocol) party when discussing the system from a protocol's point of view. When discussing cryptographic protocols, sometimes the term *principal* is used. This emphasized the party's knowledge of cryptographic keys.

The network typically consists of a number of links and broadcast networks. However, in this study the physical implementation or connectivity topology of the network is not interesting. Instead, we suppose that the network is fully connected, or that all pairs of nodes are equally capable to communicate with each other through the network.

We assume that all the nodes are relatively independent of each other. Each of them executes a potentially different program. After some initial phase (of key distribution), the only means of communication between the nodes is the network. All communication is accomplished by sending and receiving messages[2]. A node sends a message to another node by placing the message along with the address of the intended recipient onto the network. Whenever the network operates reliably, the recipient will receive all the messages other nodes have sent to it, maybe after some delivery delay. In practical terms, this corresponds closely enough to most modern data communication networks, e.g. the Ethernet and the Internet.

The network is assumed to be unreliable and potentially hostile. That is, there are no guarantees that a message sent will ever be delivered, nor that a delivered message was ever sent by the claimed sender. Another point of view is to think that the network is operated by a powerful adversary who is capable of eavesdropping, deleting,

1. The term "intensional knowledge" is commonly used in artificial intelligence research. It is used to emphasize that the knowledge is ascribed by the system designer to the agents, not computed by the agents themselves. See e.g. Chapter 9 of [37] for more details.
2. In data communication terms, this model can be referred as connectionless datagram service.

modifying and generating any messages. The adversary may deliver a legitimate message unaltered to the intended recipient, or he may not.

In the formal models, we usually assume that the adversary is *not* able to generate *some messages,* i.e. legitimate looking but forged (cryptographically protected) messages. Even though it is typically *possible* that a party is capable of generating any single one of all possible messages, there are so many possible messages altogether that in practice it is not possible to generate all of them. That is, there are so many possible messages that an adversary cannot enumerate them all. When the messages are protected by cryptography, it is *possible* that the adversary may create a legitimate looking message by pure *chance*. However, the probability of such a chance is so small that the possibility is neglected, and it is assumed that the adversary *cannot* create legitimate looking messages without having the right key(s) in its possession.

A further assumption is that at least some of the nodes in the network are trustworthy. That is, we assume that at least some of the nodes execute a program that behaves according to the assumptions given in the protocol specification. The number and identity of trustworthy nodes, as well as the level of trust we have in them, differs from one protocol to another. Thus, it is desirable to be able to explicitly specify how much and in which ways we have to trust to the nodes in order to maintain the security of the system being studied.

A *secure channel* is a means of communications between two parties that can be consider secure. (The actual security of the channel is not an issue in the definition, even though it is in practice.) Similarly, an insecure channel is a means of communication that cannot be considered secure. The purpose here is to model that some communication is insecure, some secure, and to figure out on what basis insecure channels may be turned into secure ones, e.g. by means of cryptography.

The goal is to design protocols that ensure the security of system as long as all trustworthy nodes behave according to the level of trust expected. That is, whatever the network decides to do, and whatever the untrusted nodes do, the system remains secure. It is quite possible that no computation will be accomplished (e.g. the network does not forward any messages at all). We are not interested in that; our only interest is that if the system is initially in a secure state, it will remain secure independent of what happens within the system.

2.1.2 Security functions

The usually defined security functions (also called security services) are *availability*, *confidentiality*, *integrity*, *identification*, *authorization*, *delegation*, *authentication* and *non-repudiation* [6, 58]. We will mostly ignore availability, define confidentiality and integrity as the fundamental functions, consider identification orthogonal to authorization, combine authorization and delegation into a compound framework, and view authentication and non-repudiation as functional adjectives instead of actual functions[1]. In the following, these functions are discussed in more detail.

However, before going into details, we want to emphasize one point. In our terminology, *identification* is used for what is typically called authentication. That is, in the general cryptographic literature the original meaning of the word authentication, i.e. ensuring authenticity, or ensuring something being authentic, has been extended to denote authentication of identification, or ensuring that the identity claimed is authentic. In our opinion, this convention leads to error prone thinking. If authentica-

1. We are well aware that considering authentication and non-repudiation as properties of the other functions does not follow the usual conventions. See page 11 for a more detailed explanation.

tion means only authenticity of claimed identity, the other forms of authenticity are easily neglected, e.g. authenticity of authorization. In this text the word authentication is used in the more general way, meaning that the authenticity of something is being assured.

Availability, confidentiality and integrity. Availability, confidentiality and integrity are the basic dimensions of security. A system being *available* means that it is able to perform its intended computations and able to deliver the results in the intended way. Usually the term also implies that there is some upper bound for the time waited. Data *confidentiality*, on the other hand, means that whatever happens, the data stored into and computed by the system will be available only to legitimate parties. In other words, data is never revealed to illegitimate people or unintended parts of the system. Data *integrity* is the dual of confidentiality: the data (and operations) of the system may be modified only by the intended parties.

In this study, availability issues are mostly ignored. The confidentiality and integrity of data are seen as the basic security properties to be maintained. Our approach may or may not be consistent with a given real world situation. Sometimes data confidentiality is not an issue at all, e.g. if the data being processed is publicly available elsewhere. On the other hand, sometimes data confidentiality may be much more important than integrity, e.g. when communicating intelligence data. In that case, in addition to the contents of the data, the existence of the data will probably be confidential.

Confidentiality and integrity properties can be found on different architectural levels of a system. At the highest level, the system may be divided into two or more subsystems (e.g. subnetworks), one or more of which may contain information that may not be seen nor modified by the others. Respectively, at the lowest level a single field within a message being transferred may be confidential, i.e. may not be revealed to anyone but the intended recipient. This may be the case even if the contents were to be disclosed a moment later.

Identification, authorization and delegation. Identification and authorization are typically means to ensure data confidentiality and integrity. Sometimes they can be considered primary security properties of their own, but more often their sole purpose is to ensure that remote messages or remotely originated operations are initiated by legitimate parties, thereby ensuring that read or modification requests do not endanger data confidentiality or integrity.

Identification denotes detecting and ensuring the identity of a communicating party. For example, one may want to identify the sender or the recipient of a message. In our context, we require that identification is always done without doubt. This is sometimes called *strong identification* as opposed to weak identification, which typically means detecting identity without gaining much assurance about it.

As we noted earlier, in literature the word *authentication* is often used as a synonym for strong identification. We reserve authentication to denote gaining assurance about the authenticity of an (electronic) document; this is discussed below in more detail. As an example of how authentication is used to denote what we call identification, Li Gong defines authentication as “a procedure by which one principal [party] assures his identity to another principal”. [45, page 37]. He further makes the point that it is a common practice to base authentication (i.e. identification in our terminology) on indirect bases, e.g. on proving possession of a secret that is supposedly known only by the intended parties. More generally, identification is usually based on something a party *knows, possesses* or *is*.

When speaking about distributed systems, two different types of identification are typically distinguished. *Message origin identification*[1] denotes gaining assurance about the identity of the originator of a message. Message origin identification shall always be combined with message integrity in order to assure that the message was received intact.

Peer identification[2], on the other hand, denotes assurance that the communicating party at the other end of a connection has the assumed identity. Peer identification is usually meaningful only in the context of secure channels. For example, we may be interested in knowing that a remote user issuing a command really has a legitimate identity in order to do so. Peer identification is typically combined with connection integrity, confidentiality, or both.

Whereas identification denotes assurance about identity, *authorization* refers to the legitimate rights of a user. It is important to notice that in this sense identification and authorization are orthogonal to each other. It is a common practice to use a user's identity as a base for access control decisions, thereby requiring identification in order to check authorization. This is unnecessary, and in fact we will later argue that whereas most current cryptographic protocols try to identify a communicating party, it would often be more appropriate to ensure the party's authorization to communicate instead of (or in addition to) ensuring her identity.

Authorization can be somewhat simplistically seen as a two step process where the source of authorization (e.g. a security officer) authorizes a given party to perform some operation, and later on, when the party intends to perform the operation, a security system checks that the party has the authorization needed. Authorization may also be delegated, i.e. the authorized party may authorize another party to perform the operation on his behalf. In fact, if the source of authorization never performs any authorized operations herself, all authorization can be considered delegated.

Authentication and non-repudiation. While confidentiality and integrity are system or data properties, and identification, authorization and delegation are typically means to identify users and ensure legitimacy of operations, authentication and non-repudiation measure the level of confidence one can place upon an electronic document. That is we, consider authentication and non-repudiation as attributes denoting the strength of e.g. identification.

In our terminology, *authentication* means that the party seeking authentication is able to convince itself about the authenticity of an electronic document or other information, i.e. that the document has been constructed by the party or parties that claim so, and that the document is unaltered. The notion of an electronic document shall be understood in a very broad sense here. In fact, any information may be authenticated. This means that the agent considering the authenticated information *believes*[3] without doubt that the information is true. In addition to having been sent to the authenticating party as a message, the authenticated information may be deduced by the authenticating party from e.g. data received during a protocol run. To make distinction between generic strong beliefs and authenticated information, we define authentication to refer only to such *information about other agents that has been gained*

1. What we call message origin identification is often called message origin authentication, see e.g. [58]
2. Similar to message origin identification, our peer identification is more commonly referred as peer authentication [58].
3. The agent believes in the information, it does not *know* it, since the authentication method used may be based on initial beliefs that might be wrong. For example, the agent may initially believe that a private key is only known by a single party while the key has already been compromised.

during the normal operation of the authenticating agent, i.e. after any initial key distribution or other similar period.

Again, we emphasize that our definition of authentication slightly differs from the common usage in literature. The word authentication is usually associated with identification, i.e. defined to mean the process of gaining assurance about the identity of some party, e.g. the originator of a message. Our sense is more general, and can be easily understood in the usual way when referring to identification. For example, peer authentication can be seen as detecting the authenticity of the message exchange with which the peer is able to convince the other about its identity.

Non-repudiation means that the authenticity of a document can be proven in such a way that the intended parties cannot repudiate, or deny, the authenticity of the document. That is, a non–repudiable, or indisputable, document can be presented to an independent judge who is able to determine the authenticity of the document by inspecting solely the document (and maybe some publicly available information, such as a public key available from a directory).[1] Thus, the difference between authenticated information and non–repudiable information is that the party having authenticated information believes in it, but cannot necessarily convince others to believe in it, while a party believing in non–repudiable information is able to convince others to believe in it as well. In this sense authentication can be seen as a weaker form of non-repudiation [61].

2.1.3 OSI security services

The OSI Security Architecture (ISO/IEC 7489-2) [58] defines a number of security services. The defined services are enumerated in Table 2 above. As the service names alone indicate, the OSI security framework uses a terminology different from ours. In particular, authentication is seen as a service similar to our identification function, and non-repudiation is seen as a separate application level service. This is in sharp contrast with our terminology where authentication and non-repudiation are used as attributive terms describing the validity level of security information.

The OSI terminology also lacks the system architecture point of view. The basic security services, confidentiality and integrity, are only applied to communication protocols, not to distributed systems as whole. In the OSI world, the distributed systems viewpoint is partially covered in the ODP (Open Distributed Processing) architecture, which is more or less separate from the basic OSI protocol architecture. However, issues pertaining to ODP are beyond the scope of this study.

In fact, it has been argued that the OSI security architecture contains little (if any) insight to the significance of or relations between the security services defined. This has been criticized and alternative models have been proposed in e.g. [61]. However, even there no clear distinction has been made between identification and authentication in the sense we present it here.

2.1.4 CORBA security architecture

As an alternative, more system oriented view to distributed systems security we want to briefly introduce the CORBA security architecture. CORBA, the Common Object Request Broker Architecture, is a framework standard developed by OMG, the

1. When speaking about digital signatures, most signatures are seen as non-repudiable in our sense. However, there are digital signature schemes where co-operation by either the originator or the receiver of a document is needed to determine the authenticity of a signature.

TABLE 2. OSI security services according to ISO/IEC 7489-2

Security service name	Possible layers	Service description
Peer entity authentication	3, 4, 7	The identity of peer protocol party is known and believed, i.e. authenticated
Data origin authentication	3, 4, 7	The identity of the party having created a message is authenticated
Access control service	3, 4, 7	A host refuses or accepts connections based on some authorization data
Connection confidentiality	1-4, 6, 7	All data flowing on a connection are protected against eavesdropping
Connectionless confidentiality	1-4, 6, 7	A single message is protected against eavesdropping
Selective field confidentiality	6, 7	A single or a number of selected data within a message are protected against eavesdropping
Traffic flow confidentiality	1, 3, 7	Not considered in this study
Connection integrity with recovery	4, 7	All data flowing on a connection are protected against modification in such a way that the protocol tries to recover from detected modifications
Connection integrity without recovery	4, 7	All data flowing on a connection are protected against modification. However, modification is only detected, and no attempt is made to recover from it.
Connection integrity selective field	7	Some data flowing on a connection are protected against modification.
Connectionless integrity selective field	7	Some data within a message are protected against modification.
Non-repudiation at origin	7	The identity of the originator of a message is known and believed, and it can be proven that he has once sent the message.
Non-repudiation at receipt	7	The identity of the recipient(s) of a message is (are) known and believed, and it can be proven that she has (they have) once received the message.

Object Management Group. The CORBA security architecture defines a number of distributed systems security services from the applications' and manager's point of view.

The CORBA architecture defines a number of distinct security services:

- The identification and authorization framework is built around the definition of an abstract *principal*. A principal may have many different identifies for different purposes (e.g. audit identity, non-repudiation identity). Authorization is based on principal's privilege attributes (e.g. capabilities), not identity. This allows privilege attributes to be delegated from a principal to another. The principal identities and privileges are represented as credentials. The purpose of authentication is to ensure the legitimacy of given credentials.

- Security of communication between objects is arranged around security associations. A security association represents a level of trust between the parties taking part in the association. An association may be used to establish security contexts which allow the traffic to be encrypted or otherwise protected. Method invocation level confidentiality and integrity are seen as separate protection functions.
- Both communicating parties may optionally implement access control services. The access control decisions are based on the parties' privilege attributes represented in the credentials. Access control can happen both at the communication (ORB) and at the application level.
- Non-repudiation is arranged as a separate service at the application level. An application wishing to have non-repudiable information to be later presented invokes an API function in order to obtain a signature to a message. The recipient of the message can verify the signature using another API function.
- Administration issues are arranged around the concept of security policy domains. There are separate security policy domains for separate security services.
- The framework provides an audit trail as a separate service.

The services of CORBA security architecture are summarized in Figure 1 on page 14.

<table>
<tr><th></th><th colspan="2">authentication and security associations</th><th>authorization and access control</th><th colspan="2">accountability</th></tr>
<tr><td rowspan="2">application level services</td><td>principal authentication</td><td>access to security context</td><td rowspan="2">application level access decisions</td><td colspan="2">application level audit log messages</td></tr>
<tr><td colspan="2">credentials</td><td colspan="2">non-repudiation services</td></tr>
<tr><td rowspan="2">communication infrastructure services</td><td colspan="2">security association and delegation</td><td rowspan="2">communication level access control</td><td colspan="2" rowspan="2"></td></tr>
<tr><td colspan="2">message protection</td></tr>
<tr><td>host system infrastructure</td><td>protection of long term keys</td><td>security context implementation</td><td>policy based access decisions</td><td colspan="2">audit log implementation</td></tr>
<tr><td rowspan="2">administration services</td><td colspan="2">message protection policy</td><td rowspan="2">access control policies</td><td>auditing policy</td><td>audit classification policy</td></tr>
<tr><td></td><td>delegation policy</td><td colspan="2"></td></tr>
<tr><td colspan="5">domain management services</td></tr>
</table>

FIGURE 1. CORBA security services [86]

2.1.5 Security contexts and associations

Distributed systems security can be abstracted with the concepts of security context and security association. A *security association* is a set of security related variables shared by two systems. A security association is usually considered to be unidirectional, or one-way. That is, a given security association AB only covers traffic originated by Alice and destined to Bob, but not traffic originated by Bob being delivered to Alice. A separate security association is needed for the opposite direction. A security association is considered to cover only those variables that are needed to protect data pertaining to the related communication session such as the cryptographic functions and session keys used. Policy related data, for example, is typically not considered a part of the association. [6]

In some texts, what we call security associations are sometimes called secure channels. Such a language usually wants to emphasize the channel nature, or wholeness, of a secure means of communication, while the term security associations emphasizes the constructive view, or a view where an association is composed of variables belonging to it.

A *security context* is a collection of security associations and other security related data such as variables describing the host security policy. While the concept of security association is closely connected to the concept of a communication session, a security context emphasises the contextual aspect of the system security variables. In particular, when thinking in broad terms a security context may be considered to contain any initial and evolving beliefs and trust relationships the protocol parties may have[1]. [61]

2.1.6 Discussion

It seems to us that the primary goal of distributed systems security is to ensure the availability, confidentiality and integrity of the system data. Identification, authentication, delegation and other security functions, such as auditing and intrusion detection, are means to work towards the primary goals, or to alleviate damage in case a security violation occurs. In our terminology, authentication is the event of gaining assurance about authenticity of digital data, independent of what the information content of the data is or how the data has been obtained. Along the same line, non-repudiation refers to the property that the information content of some digital data can be proven to an impartial judge.

The view presented is somewhat different from most of the more practical views on distributed systems security. However, we consider it important to make a clear distinction between identification and authorization, because these functions are often unnecessarily mixed due to the requirements of old, host based access control systems. Furthermore, thinking about the everyday meaning of the word ‘authentic’, it seems natural to us to enlarge the domain of authentication outside the sole function of authenticating an identity. This generalization leads naturally to the notion where authentication and non-repudiation are both seen as adjective terms measuring the level of assurance a protocol party may have. To our knowledge, this observation has not been explicitly made before, even though the word authentication has been implicitly used to refer also to authentication performed for other than identification purposes.

2.2 Cryptology

Cryptology is the branch of mathematics that covers cryptography and cryptanalysis. *Cryptography*, for one, is the science and art of designing and using cryptographic algorithms. *Cryptanalysis*, respectively, is the science and art of breaking cryptoalgorithms.

In this section we briefly describe the terminology and common practice of cryptography as it relates to our study of cryptographic protocols. In particular, we describe the concepts of symmetric and public key cryptosystems, one way hash functions and digital signatures as well as cryptographically strong random number generators. The view is external; the cryptosystems described will be used as *building blocks* in the

1. Note that this definition of security context is slightly different from that in [61]. This is mainly due to the recently arisen practice of using the concept of security association.

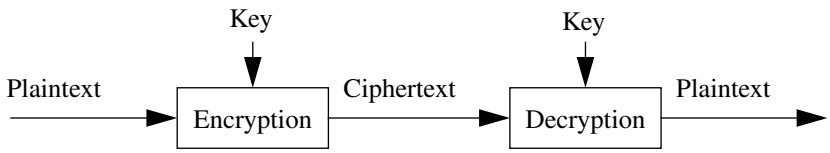


FIGURE 2. Encryption and decryption with a symmetric key

cryptographic protocols being studied. In this study we concentrate on the properties the cryptosystems have and how these properties can be formally described within the model being developed.

The reader should note that this section does not try to give a comprehensive view to cryptology or even a section of it. For such a view from a practical point of view, refer e.g. to *Applied Cryptography — Protocols, Algorithms and Source Code in C* by Bruce Schneier [99].

Terminology. The following terms are assumed to be known by the reader and only briefly described:

- *cipher*, or *cryptosystem*, refers to a particular (numerical) algorithm (which may or may not be distributed) that converts plaintext into ciphertext and possibly vice versa.
- *plaintext* is a message that has not been encrypted. The structure and contents of a plaintext message is clear to anyone having access to it.
- *ciphertext* is the result of encrypting plaintext, or in the case of encrypting multiple times, an already encrypted message. A ciphertext can be converted back to unencrypted value by decrypting. The structure and contents of ciphertext cannot be determined without decryption or successful cryptanalysis.
- *hash value* is a value calculated from a message by applying a one way cipher. One way ciphers, usually called cryptographic hash functions, and hash values are described in more detail below.
- *encryption* and *decryption* are the operations or algorithms of converting plaintext to ciphertext and vice versa. Together they form a cipher.
- *key* is an arbitrary number (or a pair of numbers) that is used to encrypt and decrypt messages. In this study, it is considered impossible to decrypt messages without the appropriate key.

2.2.1 Symmetric cryptosystems

A symmetric cipher is a cryptosystem where the same key is used for both encryption and decryption. The basic model is depicted in Figure 2 above. The basic benefit of symmetric algorithms is that they are typically relatively fast. Thus, their main use is to protect large amounts of data. Respectively, the drawback is that both the sender and the recipient(s) must all share the same key. Therefore symmetric encryption is not alone sufficient evidence to determine origins or authenticity of a message, but must be combined with other knowledge such as a log of all messages sent or recognition of a freshly generated random number, i.e. a nonce.

In addition to protecting large amounts of data, symmetric ciphers are often used in identification systems that use a trusted third party (e.g. Kerberos) as well as in reauthentication protocols where the possession of an earlier session key is taken as evidence of authentication.

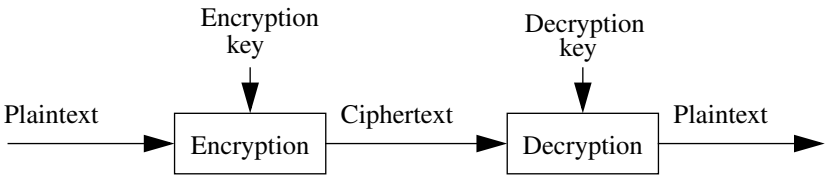


FIGURE 3. Encryption and decryption with asymmetric keys

2.2.2 Public key cryptosystems

A public key algorithm is designed so the key used for encryption is different from the key used for decryption (see Figure 2). Furthermore, it is computationally infeasible to deduce the decryption key from the encryption key (and often also vice versa). This is a clear improvement from a symmetric algorithm since the information needed is different for the different parties. This allows public key systems to be more easily used for authentication purposes than symmetric algorithms.

The fundamental drawback of public key systems is that they are relatively slow; typically in order of 100–1000 times slower compared to symmetric algorithms of similar security level. Therefore they are normally used for encrypting relatively small amounts of data, for example a number of symmetric keys that will be later on used for transferring larger amounts of data.

While public key systems are the most common means used to prove the possession of a private secret (key) known only to the holder, thereby authenticating e.g. the identity of a communicating party, or to transfer session keys that will be used later, these both goals can be achieved by other means as well. For example, zero knowledge protocols can be used to prove possession of some secret, and key agreement systems such as Diffie-Hellman can be used to securely distribute session keys.

2.2.3 One-way hash functions and digital signatures

A cryptographic one-way hash function is a function that is relatively easy to compute, but virtually impossible to reverse. That is, given x it is easy to compute $\mathrm{h}(x)$, but given $\mathrm{h}(x)$, it is very hard to find a value y with $\mathrm{h}(y) = \mathrm{h}(x)$. In addition to this, a one-way hash function shall be collision-free; it must be hard to generate a pair x, y with $\mathrm{h}(x) = \mathrm{h}(y)$. The most common purpose of a hash function in a cryptographic protocol is to provide evidence that a message has not been tampered with. Given that a party somehow trusts in a hash value (e.g. the hash value itself or some information needed to generate the value was received encrypted), it is easy for him to verify that a received message is the same message as the sent one.

Strictly mathematically, there is no evidence that there are at all functions for which the theoretical complexity of the function and its reverse are different[1]. In this study, we don't care about this; in practice, there seem to be plenty of functions that are good enough for our purposes.

A *message authentication code*, or MAC, is a one-way hash value generated from a shared secret and the authenticated message. That is, Alice, the sender of a message, calculates e.g. $\mathrm{h}(k, m, k)$ where k is a shared key and m is the message sent, comma

1. The algorithm itself must be polynomial for obvious reasons. According to [99, p. 238], the scientists have not been able to prove that there is no polynomial time algorithm for cracking or reversing the cryptosystem.

denoting concatenation, and sends the pair $m, \mathrm{h}(k, m, k)$ to Bob. If Bob believes that he has never sent the value $\mathrm{h}(k, m, k)$, and that k is only known to Alice and himself, he can deduce that the message was necessarily sent by her.

A digital signature, on the other hand, is a hash value encrypted with the private key of a party. For example, if Alice wants to prove to someone that she is the originator of a message m, she calculates $\mathrm{enc}(k_{\mathrm{Alice}}, \mathrm{h}(m))$ where $\mathrm{enc}(k, m)$ denotes the encryption of m using key k and k_{Alice} is the private key of Alice. When Bob, or anyone else knowing Alice' public key, receives the message $m, \mathrm{enc}(k_{\mathrm{Alice}}, \mathrm{h}(m))$, he can calculate $\mathrm{h}(m)$, decrypt the signature $\mathrm{enc}(k_{\mathrm{Alice}}, \mathrm{h}(m))$ with Alice's public key, and compare the values.

The basic difference between a MAC value and a digital signature is that a MAC value can only be verified by the intended recipient (having the shared key) whereas a digital signature can be verified by anyone. One should note, however, that there are many types of digital signature schemes, each having different properties with respect to what is authenticated, whether the signature is repudiable or not, or whether the sender or receiver has to actively operate when resolving disputes. For more information, see e.g. [91].

We shall see that one-way hash functions, message authentication codes and digital signatures play a crucial role in cryptographic protocols. Some basic reasons behind this are that hash functions are typically fast to compute, potentially reduce the amount of data sent through the network, and in some cases make known-plaintext cryptanalysis harder when compared to e.g. plain encryption.

2.2.4 Random number generation

Many cryptographic protocols need newly generated, or fresh, random numbers. These are typically used as session keys or cryptographic nonces[1]. Typically the random numbers are produced by *cryptographically secure pseudo-random* number generator. This means that the generated number sequence

1. looks random, i.e. passes all (or most) statistical tests of randomness that can be found,
2. is unpredictable, i.e. it must be computationally infeasible to predict what the next random number will be even given the complete knowledge of the algorithm and all the previous numbers in the stream, and sometimes
3. cannot be reliably reproduced, i.e. even when given the same input values, the generator will generate a different sequence each time. [99, page 45]

The last property is usually desirable (e.g. when generating session keys). However, if a random number generator is used to construct a stream cipher, it is not wanted.

2.2.5 Security of cryptosystems

The security of a cipher depends on the algorithm and keys used. The quality of the algorithm and the length of the keys are somewhat different issues and shall be considered separately. However, they are both based on information theory.

From an information theoretical point of view, a cryptoalgorithm having *perfect secrecy* is defined as one where a ciphertext gives a cryptanalyst no additional information about the encrypted message. This is only possible to achieve using non-

1. The term "nonce" is defined later in detail, see section 3.4.3, "Freshness, timeliness, nonces and replay prevention", on page 42.

repeating, completely random keys at least as long as the messages sent. Such a cryptosystem is called the *one-time pad*. One-time pads are hard to use in practice since they require that a key of the length of messages to be encrypted must be exchanged in the key distribution phase. Furthermore, when all the bits in the key have been used, a new key must be agreed upon using means external to the system, or the encrypted communication stops. [29, 99, 101]

Unconditionally secure cipher is a cryptosystem where the key value cannot be determined for sure independent on the amount of ciphertext available. $H_C(k)$, key equivocation, measures the uncertainty in key k given an amount of ciphertext C. Hellman has shown that given enough ciphertext encrypted with the same key, it is always theoretically possible to determine the key, if there is a means to distinguishes a meaningful plaintext message from a meaningless one [29]. However, if the plaintext message is an evenly distributed random string so that there are no statistical or other means to differentiate plaintexts from each other, the key cannot be determined. Due to this reason, when developing cryptographic protocols, it is wise to restrict encryption to random strings, if otherwise possible.

In practical terms, perfect secrecy is only achievable by using a perfect one-time pad. However, it is possible to create a cryptographic protocol where all encryption is unconditionally secure. To achieve such a case, only random data may be sent encrypted. For example, if the purpose of the encrypted contents is to work as a nonce, or a fresh random identifier, unconditional secrecy may be achieved. However, if there is any meaningful data whose confidentiality must be protected, the only way to achieve unconditional security is to use a one-time pad. From this point of view, perfect secrecy and unconditional security may be considered (almost) identical.

From a computational complexity point of view, all cryptographic algorithms have the worst case complexity of at least NP. Given a plaintext, a ciphertext, and a key, it is always possible to determine, in polynomial time, whether the given ciphertext corresponds to the plaintext and key. Therefore a cryptoalgorithm cannot be more complex than NP. However, the computational complexity is not the best means to measure the security of a cryptoalgorithm, since it measures worst case complexity. Good worst case complexity is not enough, since a secure cryptosystem must be hard to solve in (almost) all cases. [29, 99] Alternative means to measure cryptographical complexity is beyond the scope of this text.

What comes to symmetric key length, Schneier argues by thermodynamic principles that a brute-force attack against a key length of 192 is only possible if the computer used is able to losslessly compute using all the energy from several stars, and a brute-force attack against key length of 256 is impossible in our universe unless computers are built “from something other than matter and occupy something other than space” [99, page 158]. Thus, if there were no differential cryptanalysis or other better-than-brute-force attacks for a given symmetric cryptosystem, key length of 192 would certainly be good enough for all purposes.[1]

The case of public key cryptosystems is somewhat different. Majority of the public key cryptosystems used today are based either on the difficulty of factorising large numbers or taking a discrete logarithm in a finite field (for example, modulo prime). Neither of these problems have been proved to be hard, i.e. NP–complete. Thus, it is possible (though apparently unlikely) that a breakthrough will appear in discrete mathematics which will make solving these problems much easier than today. It is also possible that so called DNA-computing or quantum computers may yield solv-

1. Note, however, that if the quantum computers ever become reality, this argument may become false.

ing some NP–complete problems somewhat more tractable. Putting these possibilities aside, Schneier argues that “in every decade we can factor numbers twice as long as in the previous decade” [99, page 162]. Based on that, he gives a recommendation for RSA key length for the next 50 years. The recommendation is replicated inTable 3 on page 20.

TABLE 3. Long-range factorising predictions [99, page 162]

Year	RSA key length in bits
1995	1024
2005	2048
2015	4096
2025	8192
2035	16,384
2045	32,768

2.3 Cryptographic protocols

The term *cryptographic protocol* is used in the literature to denote several, somewhat different classes of protocols. Perhaps the largest definition includes all communication protocols that use cryptography in one or another way. A less covering definition restricts the term to denote those protocols where the specific type of cryptography used is an intrinsic part of the protocol security. From the wider point of view, PGP encrypted electronic mail might be considered a cryptographic protocol. From the narrower point of view, only protocol such as zero knowledge proofs etc., where the cryptography is bound with the protocol, are considered cryptographic protocols.

As already mentioned, in this study we concentrate on a class of cryptographic protocols often called *authentication protocols*. However, also other kinds of protocols are occasionally lightly touched.

Basically, an authentication protocol is a cryptographic protocol whose purpose is to create enough of evidence that one or more of the communicating parties may reasonably believe in something. That is, an authentication protocol has a *goal* (or a set of goals) that the parties want to achieve. For example, in a client–server system the client may want to convince the server that it has legitimate rights to access the service provided by the server, and the server wants to allow access only to legitimate clients. By exchanging cryptographically protected information the client may prove that it really is a legitimate client, having right to access the service. In other words, the goal of an authentication protocol is achieved by means of exchanging cryptographically protected messages.

Modelling formally the goals of a cryptographic protocols is by no means simple nor straightforward. Event the actual meaning of the words *authentic* or *authentication* differ slightly from situation another. Usually this involves information about the identity of the parties as well as timeliness. With timeliness, which is often also referred as freshness, we mean that a party has all reason to believe that the purported other party is involved with the communication *now*. With this property one can assure that the authentication process is performed on-line, and that it is not a question of a replay attack. We will return, in more detail, to the goals of cryptographic protocols in Chapter 3, “The purpose of cryptographic protocols”, on page 35.

An example. Let’s consider a case where one party, namely Alice, wants to send another party, Bob, a letter. Alice and Bob do not know each other very well, but they

have a common friend, Trent, that they both trust. Alice and Bob have both been previously communicating with Trent, and they have shared a symmetric cryptographic key with Trent. Trent, being very decent person, has one key for dealing with Alice and another for dealing with Bob.

Now, given this situation and the trust relationships, it is well possible to device a protocol that allows Alice and Bob to establish a shared symmetric key for mutual communication. Trent could be trusted to introduce Alice and Bob to each other, and to guarantee that the newly established key is known by anyone else. However, designing, modelling, verifying, and correctly implementing such an protocol is not at all easy, as we shall see.

2.4 Modelling of communication protocols

A communication protocol is a predefined sequence of message exchanges between two or more parties. Most protocols involve more or less nondeterminism, at least in the form of failure behaviour. Typical protocols also allow some nondeterministic behaviour for one or more parties, i.e. a party may opt to send a message A or a message B in a certain situation. However, most of the protocols that we consider have very little nondeterminism: either the messages flow in the strict predefined order, or a failure occurs.

The purpose of a communication protocol is to convey information or knowledge between the parties involved. The actual information flow (as opposed to the message flow) may be unidirectional or bidirectional; the same applies to knowledge. The difference between information and knowledge is subtle. In this study, information will refer to any information in the Shannonian sense whereas knowledge refers to intensional knowledge of a party within a distinct protocol model (cf. section 4.4.3, “Knowledge, common knowledge and distributed knowledge”, on page 58).

Notation. We have adapted the common practice of referring to nodes participating in protocols with names. The names are defined in Table 4, adapted from [99].

TABLE 4. Dramatis Personae (adapted from [99, page 23])

Alice	First participant in all the protocols
Bob	Second participant in all the protocols
Carol	Participant in the three-party protocols
Eve	The Environment, or an Eavesdropper
Mallory	Malicious active attacker
Sue	More or less trusted Server in a protocol
Trent	Truster arbitrator or other trusted party

Two formats are used to denote messages exchanged between nodes. In the first format the sender S, the intended recipient R and the message m are given in a straightforward way:

$$S \rightarrow R : m$$

In the second format, the sending and receiving of messages are seen as separate actions. In the send action the sender, recipient and message are denoted. In the receive action only the receiver and message are given, since it is not clear which message of a potentially large number of identical messages the network has decided to deliver:

send(S, R, m)
receive(R, m)

Sometimes these formats are abbreviated, leaving out the sender, the recipient, or both. The abbreviated forms are used when the sender or the recipient are unambiguous from the setting.

Terminology. An *action*, or event, is the basic building block of all protocols. An action is any externally visible or internally meaningful operation a party is at least theoretically able to perform. Typical actions are e.g. sending a message, receiving a message, and generating a new random number. All the formal models we will consider have actions in a form or another.

A *state* is a element of a model that denotes a period of no actions. After each action, a protocol party enters a state. When another action is executed, the state is left, and another state is entered. A state may be associated with properties, often expressed in the form of propositions. These can be used in giving semantics to a protocol model.

A *protocol run*, or *trace*, is a sequence of actions. For example, in a protocol where Alice sends Bob message m and Bob receives it, the possible actions can be defined as $\text{send(Alice, Bob, } m)$ denoting that Alice sends Bob the message m, and $\text{receive(Bob, } m)$ denoting that Bob receives the message m.

A protocol *specification* is a formal statement that specifies the desired properties of a protocol. What the desired properties are, on the other hand, depend on the goals of the protocol (see Section 2.4.1, below). For example, a temporal logic based specification implies that a correctly behaving implementation will never execute certain runs. Respectively, a knowledge based specification defines the knowledge state of the involved parties in the end of any successful protocol run.

A *protocol model*, on the other hand, is a formal model that can be directly used as a basis of a protocol implementation. A protocol model is often expressed in terms of actions, states and state-transition rules. For example, a protocol model may define that when Alice has received the message m from Bob, she may either send $m + 1$ or $m - 1$ back. In this case, if Alice sends anything else to Bob (or if she does not send anything at all), it is considered a violation of the model.

The basic distinction between these two is that a specification typically describes how a protocol shall *behave* whereas a model gives an internal, implementable representation of the protocol. The idea of *verification* is to ensure that a given protocol model fulfils the given specification(s).

An example. Let's have a closer look at the simple case of the two possible actions, $\text{send(Alice, Bob, } m)$ and $\text{receive(Bob, } m)$. From these actions, we can construct a set of runs.

The set of all (even impossible) protocol runs of length 2 is

$$\{\langle \text{send(Alice, Bob, } m), \text{send(Alice, Bob, } m)\rangle,$$
$$\langle \text{receive(Bob, } m), \text{receive(Bob, } m)\rangle,$$
$$\langle \text{send(Alice, Bob, } m), \text{receive(Bob, } m)\rangle,$$
$$\langle \text{receive(Bob, } m), \text{send(Alice, Bob, } m)\rangle\}$$

From these, the only *possible* runs are

$$\{\langle \text{send(Alice, Bob, } m), \text{send((Alice, Bob, } m))\rangle,$$
$$\langle \text{send(Alice, Bob, } m), \text{receive(Bob, } m)\rangle\}$$

The latter one is the only *possible* ones since only these preserves the information properties of the protocol. That is, it is naturally not possible to receive a message before it is sent. However, it is well possible to send the same message again, even though the first one has not been received.

The only *successful* run is $\langle \text{send}(\text{Alice}, \text{Bob}, m), \text{receive}(\text{Bob}, m) \rangle$.

This is the only *successful* one since only this leads to the intended result. That is, only in this case Alice manages to send the message m to Bob.

2.4.1 Expressing protocol goals

As already mentioned, the purpose of a protocol is to convey information or knowledge. This goal can be formally expressed in several ways. An external specification may describe the goal as requiring that the future system behaviour will be restricted in a certain way after a successful protocol run. This is sometimes implied by a more generic specification which defines the legitimate system behaviour during both the protocol run and after it. For example, a specification may state that a subsystem never expresses any confidential information except when encrypted with a symmetric key that is only known to the subsystem itself and to a single trusted party. This implies that after a protocol run where the subsystem and a trusted party have agreed on a session key, the subsystem may send confidential information using the given key, but not in any other conceivable way.

As an alternative to a behavioural specification, one can make statements about the successful final states of a protocol model in terms of information or knowledge. For example, a specification can state that after a successful protocol run the key generated during the run is *only* known by the participating nodes, that the key *is* known by the nodes, and that each of the nodes "knows" that the other node also has the key. A state restriction can also be expressed as a relation between the states of nodes within the model. For example, a specification may state that after a successful protocol run, if Alice is in state A, Bob's local state is either B or C, but not any other state.

Safety properties. A *safety property* states that if the protocol fulfils the property, nothing "bad" will happen. More specifically, if a protocol run starts in a good initial state, the system will never enter an undesired state. Safety properties are often expressed in terms of sets of runs. The property defines a set of runs that are considered good according to the property. If it can be shown that a protocol implementation never executes any traces outside the set, the implementation fulfils the given property.

For example, a safety property may require that if an agent learns any confidential information, it has been earlier given the explicit authority to read the information:

$$\forall a \in \text{Agent}, m \in \text{Messages}, \text{Confidential}(m) : \quad \text{learn}(a, m) \rightarrow \text{access}(a, m, \text{read})$$

Liveness properties. While safety properties require that nothing bad happens, liveness properties require that eventually something good happens. There are two possible reasons why a system may stop performing visible actions: deadlock and livelock. Deadlock means that the system has reached a dead end: no further actions can be performed. A livelock, instead, occurs when the system continues to perform actions indefinitely, but all actions performed are ones we are not interested in. Furthermore, we will require that in a real livelock situation the system is *not able to* proceed to any of the desirable states, i.e. to return to a state where a desirable action is possible. This is in contrast to situations where the system *is* able to proceed, but just nondeter-

ministically happens to choose an undesirable action every time. In the latter situation, if the nondeterministic choices are performed randomly, the system will *eventually* return to a sane state.

The question about livelocks is connected to issues of scheduling, fairness of the scheduler, and timeouts. From the security point of view, liveness properties can be used to model availability. These are interesting and valid issues; however, in our study we will mostly ignore all liveness properties. In a typical protocol model, a failure will be modelled as a deadlock whereas successful operation will lead to a successful termination. There is seldom even a possibility for livelocks within the models used in this study.

Benefits gained. Given all this, one may ask why to formally express the protocol goals in the first hand. There are several reasons for this. First, and maybe foremost, natural language, being informal, is often inaccurate or even ambiguous. Formal specification means accuracy, coverage and explicitness. It also allows one to use a language that is more close to the verification semantics, thereby reducing complexity. However, this all comes, of course, with a cost. Formal specifications are sometimes harder to understand. And, unless special care is taken, using a formalism may abstract away one or more properties that are important after all.

From now on, we will mostly work on the assumption that all the desired protocol goals are, to the extend needed, possible to express in the formal notations used. However, we urge the reader to remember that this assumption has often turned out false in the light of new developments.

2.4.2 Models, states and actions

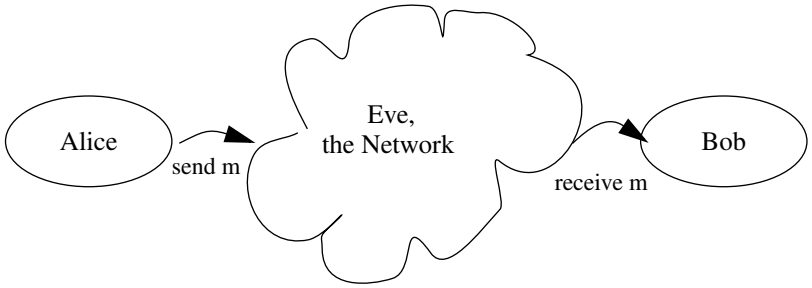
The internal behaviour of a concurrent system can be modelled using states and actions. The two main brands of such models are Petri nets [90] and process algebraic formalisms[1] such as Labelled Transitions Systems (LTS), Algebra of Concurrent Processes (ACP) [8, 13], Calculus of Communicating Systems (CCS) [80] and Communicating Sequential Processes (CSP) [55, 56, 54][2]. The basic idea in all the mentioned formalisms is pretty similar: to build a model where the global state of a system consists of a number of local states and an action means a change in one or more local states simultaneously[3].

From a non-formal, applicative point of view the main difference between Petri net formalisms and process algebraic models is that Petri nets are not easily composeable while process algebraic models are. That is, it is not straightforward to take two Petri nets representing subsystems of a large system and to combine them together, while process algebras have been developed from the beginning to make this easy. Due to this reason we will concentrate on process algebraic models in this study.

The basic formalism we will use to define protocol models is Labelled Transition Systems (LTS) [41]. ACP and CSP formulae will be used to specify and discuss protocol models when appropriate. As an example, an LTS model along with ACP and

1. Compared to plain state machines, these formalisms are more suitable for representing parallel actions. Given some semantics for parallelism (e.g. interleaving semantics), they can be converted into state machines. However, the resulting state machines are often very large and clumsy to handle.
2. In addition to these, there is a large number of other formalisms, some of which are more industrially oriented (e.g. SDL, Estelle, Lotos). However, the formal model theoretic semantics of many of these are either incomplete, or too complex for practical full scale verification.
3. This is not strictly true for Petri nets, but close enough for our purposes.

CSP formulae describing Alice sending Bob a message through the Network is given in Figure 4. All these models will be discussed in more detail in Chapter 5, “Model checking and Process Algebra”.



ACP specification: $A \parallel E \parallel B$, $\gamma(send_m, send_m) = send_m$, $\gamma(receive_m, receive_m) = receive_m$

CSP specification: $A \parallel E \parallel B$

FIGURE 4. Alice sends a message to Bob through the Network

A standard method to determine whether a protocol model fulfils a property is exhaustive search, or reachability analysis. All reachable states of a model are created, and in each state the tested property is evaluated. If the property evaluates true in all reachable states, all protocol runs fulfil the property.

In practice, exhaustive reachability analysis is not possible but for the simplest models. Due to concurrent operations and nondeterministic choices the number of possible states explodes as the system complexity increases. A typical realistic protocol model may easily contain a huge number of reachable states. With current technology, an exhaustive search can be applied to models having $10^7 – 10^8$ states, sometimes even 10^{12} states.

There are quite a few methods that have been used to reduce the number of states that have to be checked. Most of the proposed and used ones have been more or less heuristic, i.e. dependent on the problem being studied (see e.g. [76]). However, two model refinement techniques called Failures-Divergencies Refinement (FDR) [68] and Chaos-Free Failures-Divergencies Refinement (CFFD) [110] seem to be universally applicable and very promising, especially when modelling real life protocols. These methods will be briefly discussed in more detail in Section 5.4.7.

2.4.3 Logical formulae

A logic is a set of well formed formulae that includes all tautologies and is closed under the rule of detachment (if φ and $\varphi \rightarrow \psi$ then ψ) [43]. In practice, a logical formula is a sentence using the logical operators $\neg \ \wedge \ \vee \ \rightarrow \ \leftrightarrow$ and $\Box$ to describe properties concerning a model (here the box $\Box$ is an example of a modal operator). A number of primitive propositions $p_i \in \Phi$ are given a truth value at all possible protocol states using a valuation $V : \Phi \rightarrow 2^S$. For each proposition p_i the valuation V gives a subset of states where p_i is true. The primitive propositions are then combined using the logical operators to logical formulae.

The propositions typically denote facts about the model and can easily be evaluated at the various states of the model. For example, the string $\text{sent}(a, m)$ may be understood as denoting that at a given state the agent a has once sent the message m or another message consisting of the concatenation of m and some other data.

Using logical connectives and the type of propositions described, one can express facts about possible and permissible runs, acceptable initial and final states etc. As an example, a requirement that each received message must have once been sent can be expressed as $\forall a \in \text{Agents}, m \in \text{Messages} : \exists b \in \text{Agents} : \text{received}(a, m) \rightarrow \text{sent}(b, m)$. Using a temporal operator $\Diamond$ denoting "at least once in the past", a similar requirement can also be expressed as $\text{receive}(a, m) \rightarrow \Diamond\text{send}(b, m)$. The truth of the latter formula in a given state can be evaluated by determining whether the agent a receives the message m at the given state (i.e. the previous local action was $\text{receive}(a, m)$) and whether the agent b has sent the message m in the past.

Considering the local states of agents, it is also possible to express the intensional knowledge or beliefs using modal operators. For example, if the operator B_a denotes that the agent a *believes* that the formula following the operator is true, the formula $\text{receive}(a, m, h(k, m)) \rightarrow B_a\Diamond\text{send}(b, m)$ denotes that if a receives a message m along with a hash value constructed from a key k and the message, a believes that the message was once sent by b. In order to be able to evaluate the truth of this formula, the semantics of the belief operator B_a must be given. We will return to this issue later in Section 4.4, "Logics of knowledge and belief".

2.5 A protocol example

Let's consider a practical example. Our system shall consist of three parties: Alice, Bob and Eve. Here Eve denotes the network. In order to keep the example utterly easily understandable from the protocol modelling point of view, the range and domain of both encryption and hash functions has been kept unrealistically small. This may sometimes make it hard to believe that the protocol has anything to do with security. However, the reason for the security of our example lies in the unrealistically small computational capabilities of the network (it is assumed that the parties have equally small computational capabilities, but that doesn't really matter). In real life, the reason why cryptographic protocols are secure partially lies in the limited computing capabilities of the adversaries. This works, since given large enough key and hash value domains it is — as mentioned above — thermodynamically infeasible to launch a brute-force attack against the functions.

In concrete terms, we assume that all the parties are only able to remember three digits in the range {0...9} and nothing else. That makes it impossible for the parties to store more than one earlier protocol run in order to use that information for cryptanalysis. It also makes it impossible for the parties to store the representations of encryption and hash functions in memory as tables. Furthermore, it is assumed that a random numbers in the range (0…9) are unpredictable enough and that it is infeasible to try them all. It is also assumed that the encryption function $\text{enc} : \{0\ldots9\} \rightarrow \{0\ldots9\}$ and hash function $\text{h} : \{0\ldots99\} \rightarrow \{0\ldots9\}$ cannot be cryptanalysed nor inverted (not even by exhaustive search). We also suppose that only Alice is able to encrypt anything and Bob decrypt, even though we do not explicitly denote any keys.

2.5.1 The protocol

The purpose of the example protocol is to send one digit of information from Alice to Bob without revealing it to Eve. The protocol will function as follows. Let m be the digit to be sent.

1. Alice *generates* a random nonce n

2. Alice calculates $\mathrm{enc}(m)$ and $\mathrm{h}(n, m)$, and *sends* the message $n, \mathrm{enc}(m), \mathrm{h}(n, m)$ to Bob (via Eve).
3. Upon *receiving* the message, Bob decrypts $\mathrm{enc}(m)$ revealing m and calculates $\mathrm{h}(n, m)$ to make sure that the message was not tampered with.
4. To inform Alice that he has got the message, Bob calculates $\mathrm{h}(n, (m + 1) \bmod 10)$ and *sends* it to Alice (again via Eve).
5. When Alice *receives* Bob's response, she is able to convince herself that Bob has got the digit.

The flow of messages is illustrated in Figure 5.

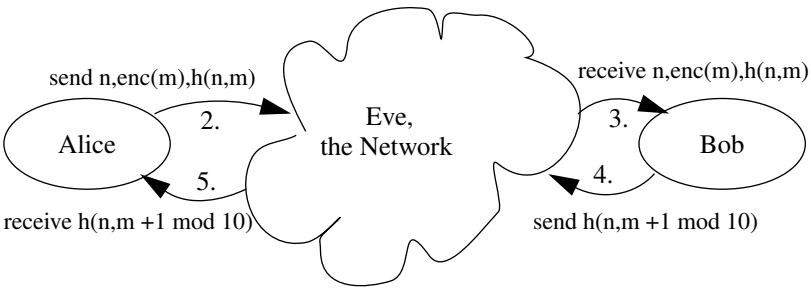


FIGURE 5. An example protocol flow

The actual formalization is quite lengthy and somewhat boring, even though highly instructive. To keep the presentation here brief, the full protocol example is given in Appendix A.

Actions. To model the protocol the following actions are defined:

send(Alice…Bob, 0…9, 0…9, 0…9)	Alice or Bob sends a message consisting three digits (n, $\mathrm{e}(m)$, and $\mathrm{h}(n, m)$) to the network. (2*10*10*10 = 2000 different actions).
receive(Alice…Bob, 0…9, 0…9, 0…9)	Alice or Bob receives a message consisting of three digits.
send(Alice…Bob, 0..9)	Alice or Bob sends a digit $\mathrm{h}(n, (m + 1) \bmod 10)$.
receive(Alice…Bob, 0…9)	Alice or Bob receives a digit.
generate(0…9)	Alice or Bob generates a random number.

2.5.2 Run based protocol specification

Given the protocol descriptions, we can formally specify the following sets:

- set of possible actions Σ (the alphabet), which is finite,
- the set of all runs $\Re_{\mathrm{all}}$, which is infinite but enumerable,
- the set of possible runs $\Re$, which is also enumerable,
- a subset of $\Re$ containing only the runs of length 5 (a finite set), and finally
- the set of legal runs $\Re_{\mathrm{succesfull}}$.

More of these sets are shown in the appendix. For brevity, here we give only a formal trace specification for the set of legal runs:

$$\begin{aligned}\mathfrak{R}_{\text{succesfull}} = \{\langle s_0, s_1, \ldots, s_4\rangle \in \mathfrak{R}_5 : \mathrm{P}(s_0, s_1, \ldots, s_4)\}\\ \mathrm{P}(s_0, s_1, \ldots, s_4) = \exists n, m :\\ s_0 = \text{generate}(n) \wedge\\ s_1 = \text{send}(\text{Alice}, n, \text{enc}(m), \text{h}(n, m)) \wedge\\ s_2 = \text{receive}(\text{Bob}, n, \text{enc}(m), \text{h}(n, m)) \wedge\\ s_3 = \text{send}(\text{Bob}, \text{h}(n, (m+1)\text{mod}10)) \wedge\\ s_4 = \text{receive}(\text{Alice}, \text{h}(n, (m+1)\text{mod}10))\end{aligned}$$

2.5.3 Protocol model

Next we develop a protocol model consisting of a local state machine for Alice and Bob. The appendix shows how a (partial) state machine for Eve, or the environment, can be modelled. Its states includes all the information Eve is able to gather during a single protocol run. To keep things simple in this example, we assume that none of the parties are able to remember information from earlier protocol runs.

The local state of Alice consists of the message to be sent m_{Alice}, a nonce n_{Alice}, and a state variable s_{Alice} denoting the actions Alice has performed so far. The latter also dictates exactly what Alice is ready to perform next:

$$\begin{aligned}m_{\text{Alice}} &\in \{0, \ldots, 9\}\\ n_{\text{Alice}} &\in \{\varepsilon, 0, \ldots, 9\} \qquad \text{where } \varepsilon \text{ denotes the empty value}\\ s_{\text{Alice}} &\in \{\text{init}, \text{generated}, \text{sent}, \text{received}\}\end{aligned}$$

The symbols init, generated, sent, received denote the initial state of Alice, the state where Alice has generated n_{Alice}, the state where Alice has sent the first message and is waiting for a reply from Bob, and the state where she has received his reply, respectively.

In the same way, Bob's local state consists of m_{Bob}, n_{Bob} and s_{Bob} (see the appendix).

Using the state variables, we can define the set of legal states for Alice and Bob:

$$\begin{aligned}S_{\text{Alice}} &= \{\langle s, m, n\rangle : s = \text{init} \rightarrow n = \varepsilon, s \neq \text{init} \rightarrow n \neq \varepsilon\}\\ S_{\text{Bob}} &= \{\langle s, m, n\rangle : s = \text{init} \rightarrow n = m = \varepsilon, s \neq \text{init} \rightarrow n \neq \varepsilon \wedge m \neq \varepsilon\}\end{aligned}$$

It is quite straightforward to define the sets of acceptable actions in each state, e.g.:

$$\begin{aligned}A_{\text{Alice}}(\langle \text{init}, m_{Alice}, \varepsilon\rangle) &= \{\text{generate}(0), \ldots, \text{generate}(9)\}\\ {}_{\text{Alice}}(\langle \text{generated}, m_{\text{Alice}}, n_{\text{Alice}}\rangle) &= \{\text{send}(\text{Alice}, n_{\text{Alice}}, \text{enc}(m_{\text{Alice}}), \text{h}(n_{\text{Alice}}, m_{\text{Alice}}))\}\end{aligned}$$

denoting that the acceptable actions for Alice, A_{Alice}, in the initial state, $A_{\text{Alice}}(\langle \text{init}, \ldots\rangle)$, are the generate actions. When Alice has generated a nonce, she can perform only one action, the send action corresponding to the message to be sent, m_{Alice}, and the nonce generated, n_{Alice}. This set is described above as $A_{\text{Alice}}(\langle \text{generated}, m_{\text{Alice}}, n_{\text{Alice}}\rangle)$.

A successful protocol run is when Alice and Bob both end in their final states. All other sequences shall lead to a failure, i.e. a deadlock.

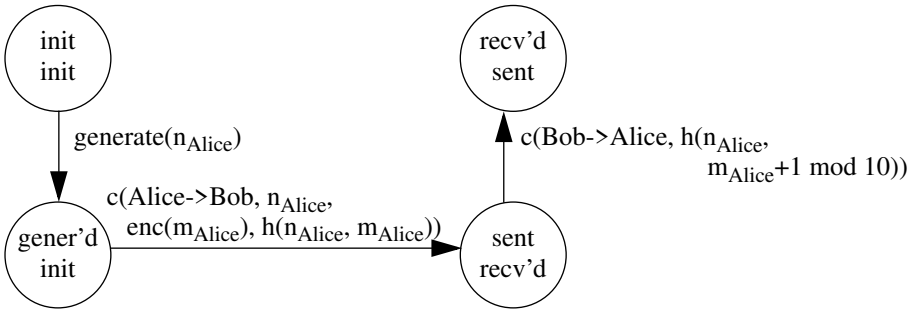


FIGURE 6. The combined LTS representing both Alice' and Bob's actions

2.5.4 LTS, ACP and CSP specifications

The individual LTS models for Alice and Bob are given in Figures 27, on page 122 and 28, on page 122 in the Appendix A. Figure 6 gives the combined LTS without any explicit network model, i.e. $A|\{\mathrm{c}(\mathrm{Alice} \rightarrow \mathrm{Bob}, x), \mathrm{c}(\mathrm{Bob} \rightarrow \mathrm{Alice}, x)\}|B$.

The behaviour of Alice and Bob can be described as ACP formulae. Table 5 on page 29 explicitly gives the behaviour of Alice at different states using ACP specifications. Given the definitions in the table, the total behaviour of Alice can be defined as

$$ALICE = \sum_{0 \le m < 10} INIT_m$$

It is worth noting that in state $SENT_{m,n}$ (see the table), Alice is ready to perform only one action: to receive a message that is sent to her and that contains a one way hash over n and $m + 1 \bmod 10$. In this way we can express, in ACP, that the system will deadlock unless someone, i.e. Bob, sends a correct reply, and that reply is indeed received by Alice.

TABLE 5. ACP formulae for Alice' behaviour in different states

State	ACP formula[a]
init	$INIT_m = \sum_{0 \le n < 10} (\mathrm{generate}(n) GENERATED_{m,n})$
generated	$GENERATED_{m,n} = \mathrm{send}(\mathrm{Alice}, n, \mathrm{enc}(m), \mathrm{h}(n, m)) SENT_{m,n}$
sent	$SENT_{m,n} = \mathrm{receive}(\mathrm{Alice}, \mathrm{h}(n, (m+1) \bmod 10)) RECEIVED_{m,n}$
received	$RECEIVED_{m,n} = \varepsilon$

a. See Appendix A for the explanations of the formulae.

Global protocol states. The combined LTS based protocol model given in Figure 6 is based on synchronous communication and therefore unrealistic. That is, the receipt of a message appears *simultaneously* with the sending of the message. This is unrealistic and does not conform to the environment model described earlier.

In order to produce a more realistic protocol model, we keep the models for Alice and Bob similar, but add a third component, Eve the environment. Eve is always able to receive messages from both Alice and Bob, and it is able to send any of the messages she has received so far to Alice or Bob. However, communication will only happen if a party is ready to send or receive a message. In this sense, communication between

Alice and Eve as well as between Bob and Eve is synchronous. The presence and behaviour of Eve makes the communication between Alice and Bob asynchronous. The most important consequence of this is that if Alice has sent a message to Bob, she does not know if he has received it before she got a response back from him.

A possible model for Eve, where she is always able to receive anything, and able to send anything she has earlier received, is given in the appendix. Naming the ACP behaviour of Eve as EVE, the behaviour of the overall system can be defined as the ACP formula $ALICE||EVE||BOB$. Analysis of this shows that the only possible successful protocol run is the one intended.

The environment model given in the appendix allows the environment to send only messages that it has already learned. However, if we consider a real world situation, an adversary can send absolutely anything to the protocol parties. It relies on the responsibility of the parties to decide whether the message received is acceptable or not. In our toy model, Eve has 1/10 probability of generating a message Bob will accept as a genuine one from Alice, and 1/10 probability of responding to Alice' message in such a way that Alice believes it coming from Bob. In real life, where the ranges and domains of cryptoalgorithms are much larger, these probabilities will be negligible and considered zero in this study.

Using the alternative environment model where Eve is able to both send and receive anything, the total model of the system becomes much more complex. Yet another environment model would allow Eve to generate some messages, but in such a way that all these messages would be "illegal". In the case of our toy model, this is easy by deliberately choosing a number of messages where the hash function checking does not succeed. Another direction is to allow Eve to remember messages from previous protocol runs. Using this approach, it can be shown that Eve is able to launch a reply attack against Bob but not against Alice. In other words, Eve is able to make Bob believe that Bob has received a message from Alice, even though Alice has not sent any messages during the protocol run, but Eve is not able to convince Alice that she has succeeded in sending a message to Bob if Bob has not received the message.

2.5.5 Knowledge and beliefs

There is one issue left to be discussed in the light of the example: the question of knowledge. The apparent purpose of the protocol seems to be to transfer the one digit of information, m, from Alice to Bob. Unfortunately it can be shown that since sufficiently powerful Eve (one that remembers earlier messages) can make Bob believe that he has received a message even when he has not, this primary goal has not been achieved. Instead, it can be shown that after a successful protocol run, Alice knows that Bob knows m, and that Bob indeed does know m, but Bob does not know whether the m he knows was actually sent by Alice during the protocol run, or if it is a reply attack sent by Eve.

Using modal formulae, and the notion to be fully introduced later in Chapter 4, the end result can be written as follows:

Alice knows (Bob has m)	Alice knows that Bob has received m.
(Bob has m)	Bob indeed has received m.
$\neg$(Bob sees $m \rightarrow$ Alice writes m)	It is not true that if Bob receives an m that it would really be sent by Alice.

The reader should notice that what is referred as knowledge above (i.e. Alice knows) is actually beliefs that hold only if the initial premises hold. The question how to formally make these kinds of statements and to reason about them will be discussed in Chapter 4, “Modal logic”.

2.6 Cryptanalysis and protocol failures

There are roughly three different kinds of methods to break a cryptographic protocol. One of these, external compromise, cannot be effectively coped with within the system. For example, if someone uses social engineering or blackmailing to get hold of a user’s long lasting keys, there is not much that can be done at the protocol level in order to alleviate the problem. The other two breaking methods — cryptanalysis and protocol failures — can be made more difficult by careful protocol design and analysis.

Cryptanalysis refers to methods where, for example, a session key or other sensitive information is revealed by breaking a cryptographic algorithm used. The ability to produce two meaningful messages having the same hash value (the so called birthday attack) can be considered cryptanalysis as well. Putting the design and analysis of the ciphers aside, there is still quite a lot a protocol designer can do to make cryptanalysis as hard as possible. For example, using encryption only to protect confidential information and hash values to retain message integrity and authenticity, one can effectively increase the unicity distance of a given cipher (cf. section 2.2.5, “Security of cryptosystems”, on page 18). [71]

Protocol failures, on the other hand, are situations where the protocol contains a design flaw that allows an adversary to break one or more of the protocol goals. A protocol failure may make it possible to impersonate as a legitimate party or break the confidentiality or integrity of the system *without* breaking a cryptoalgorithm. For example, it may be possible for an attacker to run several runs of protocol simultaneously, thereby getting from one protocol run a message that can be substituted into another protocol run. A classification of protocol flaws is given in section 2.7, “Flaws in protocols”.

Gustavus Simmons describes a number of different protocol failures in his famous article in the Communications of the ACM [102]. He describes a number of cryptographic protocols where the use of a particular cryptosystem (e.g. RSA or DSA/DSS) is more or less crucial to the success of the protocol failure. Even though in this study our main concern is in protocol failures that are *independent* on the cryptosystem used, there is a lot to learn. In particular, we want to re-emphasize the following points from Simmons’ article:

- Carefully enumerate all of the properties of all of the quantities involved, both explicitly stated and implicitly assumed.
- Take nothing for granted. In particular, if there is a way to violate a property, carefully analyse what security considerations this violation may have.
- If the outcome of the protocol can be influenced by violating one or more assumptions, carefully determine if this can be exploited to advance some meaningful deception.

In this study, the prevention and analysis of protocol failures is the main concern. What comes to cryptanalysis, the goal is to enumerate methods that make cryptanalysis harder, not to try to make it altogether impossible.

2.7 Flaws in protocols

Ulf Carlsen has presented a classification of cryptographic protocol flaws in [23]. An understanding of different types of flaws helps one to design better protocols and to understand the design choices others have made planning their protocols. In this section we present Carlsen's classification in an abbreviated form.

Carlsen makes a distinction between *specification flaws*, *implementation-dependent flaws* and *implementation flaws*. Specification flaws are mistakes at the high level definition of the protocol. Examples of functional flaws are freshless and oracle flaws, both discussed below. Meanwhile, an implementation-dependent flaw is a deficiency at the specification level in such a way that the specification can lead to at least one faulty and one non-faulty implementation. That is, the specification itself is not necessarily wrong, but it is sufficiently imprecise so that it is possible to produce a malfunctioning protocol implementation. Implementation flaws, as the name suggests, are mistakes made at the implementation phase of a (correct) protocol specification.

In this study, our main concern is in specification flaws, and to a lesser extent in implementation-dependent flaws. Handling of implementation flaws is beyond the scope of this study. However, where possible we try to device methods which make it harder to make mistakes at the implementation level.

2.7.1 Elementary flaws

Carlsen considers basic violations of cryptographic functions or understanding of what to protect and what not as elementary flaws. As an example he gives the original X.509 authentication protocol where messages where encrypted before signing. This allowed an attacker to replace the signature with his own, thereby allowing him to trivially impersonate the originator of the message.

2.7.2 Password-guessing flaws

Some cryptographic protocols use passwords to generate keys or other cryptographic information. If the protocol allows an adversary to test if a given password is valid, he can launch a brute force search on the password space, i.e. test all possible passwords one after another. Typically the number of passwords people tend to generate is much smaller than the key space of cryptographic information. In this way, a password-guessing flaw makes cryptanalysis much easier.

2.7.3 Freshless flaws

In many cryptographic protocols it is important to be sure that some datum, e.g. a session key, has been recently generated (cf. the definition of freshness in Section 3.4.3, "Freshness, timeliness, nonces and replay prevention", on page 42). The classical example of a freshness flaw is the flaw in the original Needham-Schroeder private key protocol, where B had no possibility to ensure the freshness of the session key.

2.7.4 Oracle flaws

An oracle flaw refers to a situation where a cryptographic protocol can be misused to produce illegal cryptographically protected messages or reveal the contents of cryptographic information. A typical example is a situation where the attacker simultaneously uses two protocol sessions in such a way that one protocol session can be misused to produce valid looking forged messages on the second session.

The basic method to prevent oracle flaws is never to encrypt (or decrypt) anything unknown, i.e. anything whose structure is not completely known or that is not perfectly authenticated. [70]

2.7.5 Type flaws

Explicit typing is a method of ensuring that the structure of cryptographically protected data cannot be misused. Using explicit protocol identifiers, run numbers, transmission steps, message types etc. helps to prevent an adversary from using some colleted message as a forgery for another message. However, it is not advisable to use explicit typing in encrypted parts of messages, since using explicit typing there allows the adversary to launch partially-known plaintext attacks.

Another typing flaw refers to randomly generated items. All random items transferred unmodified or protected in a reversible way, i.e. encrypted, form potential covert channels. For example, the generator of a random number may each time ensure that the low order bit of the random number conforms to information he wants to send. [102]

2.7.6 Internal flaws

In Carlsen's classification, flaws caused by a failure of a protocol party to perform a necessary operation are called internal flaws. For example, it may be crucial for a protocol party to check the structure of a MAC protected message before forwarding it. Similarly, a party's neglect to check the contents of a message before encrypting it may lead to oracle flaws. Internal flaws are typically implementation-dependent.

2.7.7 Cryptosystem-related flaws

The basic approach in this study as well as most other papers is to specify cryptographic protocols without any particular cryptosystem in mind. However, some cryptosystems have properties that make them unsuitable for a particular protocol. Some of the flaws considered by Simmons in [102] can be considered cryptosystem related as well.

Many cryptosystem related flaws are highly complex and hard to detect. According to Carlsen, "Countermeasures to cryptosystem-related flaws can only be determined on a case-by-case-basis." [23, page 198]. However, to us it seems to be preferable to try to make explicit the properties a cryptosystem is assumed to have, and the properties it is assumed not to have (if possible).

2.7.8 Lessons learned

Carlsen suggests that formal protocol specification and analysis should have explicit models for typing, type checking and internal actions. These would definitely diminish the number of new type and internal flaws, thereby probably making the probability of oracle flaws smaller as well.

2.8 Summary

In this chapter, we have briefly covered the necessary background. First, we discussed distributed systems security in general. From our point of view, the real security objectives of any system are availability, confidentiality and integrity. Identification, authorization and delegation are means used to restrict the users' access to the system, thereby trying to ensure confidentiality and integrity. In our par-

lance, authentication (or authenticity) and non-repudiation are used more as adjective terms, denoting level of assurance one has over identity, authorization, delegation or other information. Auditing, intrusion detection and other similar methods are used to detect and analyse security incidents, while the other security services are used to prevent or alleviate them.

We also briefly discussed OSI and CORBA security architectures, noting how the OSI model shows relatively little understanding of the role of various services. The CORBA security architecture is clearly better in this respect (and much more recent than the OSI model); however, even there identification is seen as one of the basic goals of system security.

After the discussion of distributed systems security in general, a brief rehearsal of the basics of cryptology was given. Various cryptographic algorithms, i.e. symmetric and asymmetric ciphers, one-way hash functions, digital signatures, message authentication codes and cryptographically strong random numbers were briefly described. This section was concluded with some notions about the security of cryptosystems.

In Section 2.4, “Modelling of communication protocols”, we introduced some basic notion used to describe cryptographic protocols. Some methods of formally describing protocol goals were briefly introduced. The concepts of models, states and actions were defined from the point of view of process algebras and logical model theory. Basic logic formulae were introduced. After this, in Section 2.5, “A protocol example”, we gave an example on how the various formalisms can be used to model an utterly simple sample protocol. The example is given in full detail in Appendix A, “A protocol example”, on page 117.

Finally, in Sections 2.6 and 2.7 we discussed the various types of flaws that have been found in cryptographic protocols. In this work, our main interest will be in modelling cryptographic protocols in such a way that possible specification flaws and implementation-dependent flaws could be avoided.

CHAPTER 3

The purpose of cryptographic protocols

Cryptographic protocols are communication protocols that have been designed to operate in a potentially hostile environment. They use cryptographic algorithms to protect messages or parts of messages from disclosure and modification. The reason for using ciphers is that it seems to be the only feasible means to protect data in a hostile environment. The protection is based on computational complexity: given a large enough domain of values, it is practically impossible to test each of them.

In this chapter the concept of *cryptographic protocols* is discussed in detail. In the light of this discussion, the scope of study is limited to a subclass of cryptographic protocols. In particular, we will not consider protocols based on more "advanced" cryptosystems such as zero-knowledge protocols, blind signatures or secret sharing schemes. The focus of this study is on identity authentication and key exchange protocols, as well as on the applicability of these protocols to authorization methods.

The *goals* of cryptographic protocols are considered from a few points of view. In particular, we will first consider high level protocol goals, or what is the purpose of a cryptoprotocol from the system point of view. The specific goals of a (sub)protocol, in particular key agreement, correspondence and secrecy are considered. Also the low level concepts, such as message authenticity and freshness, are considered. Goals are expressed mostly informally in this chapter. We will return to the goals later on, and formalize some of them.

A real life example of the ISAKMP / Oakley protocol, a forthcoming IETF standard, can be found in Appendix B, "ISAKMP / Oakley — A real world example".

3.1 Introduction

According to Li Gong, a protocol can be seen as a specification for the format and timing of messages exchanged between two or more communicating parties. A cryptographic protocol, on the other hand, is a protocol that employs cryptographic mechanisms such as encryption and one-way hash functions to guarantee the integrity, confidentiality, identity of origin or destination, order, or timeliness of the messages. All these message properties contribute to the *meaning* of the messages, and therefore to the goals of the protocol. [45, page 1]

The *purpose* of a cryptographic protocol is to achieve a goal. The goals of different protocols vary quite much. As an example, the purpose of a protocol may be to exchange a session key that will be used to ensure integrity of a forthcoming session, to authenticate the identity of one or more parties involved in the protocol, or to check if a given protocol party is authorized to perform some action. The goals are the main subject of this chapter, and will be considered in more detail in sections below.

Some *initial assumptions*, or an initial state, must hold so that a protocol can achieve the intended goals. In most protocols some of the parties involved must be more or less trustworthy, or operate in a predefined way. For example, if a key distribution centre (KDC) in a symmetric key authentication protocol cannot be trusted, there is no possibility the protocol could achieve its goals. In addition to being trustworthy, the parties must share some common knowledge, possess some private or shared keys, and have capability to perform protocol actions and cryptographic functions.

Most of the literature seems to more or less ignore computational capabilities of the protocol parties. The issue of trustworthiness seems to be confusing, and there are a number of papers expressing explicit trust models, whereas most papers lack an explicit trust analysis (though, an implicit trust model is typically present). The knowledge of the parties in terms of keys and such is almost always clearly defined.

The *means* of achieving the goals is to exchange cryptographically protected messages. The properties of typical cryptographic protocols in the view of determinism and possible message sequences were already briefly mentioned in Section 2.4, “Modelling of communication protocols”. The main properties are relative determinism, i.e. only one message in transit at a time, and predetermined sequences of messages with no or very little branching.

3.2 High level protocol goals

In this study, the concept of *high level protocol goals* refers to the system level goals of a protocol, or the actual external behaviour the protocol is designed for. For example, a typical high level goal is to keep the data stored in the system confidential. However, it is not always easy to identify or not even clear what are the actual highest level goals of protocol. It is even probable that a given protocol will be used in various situations where the ultimate goals are different. Thus, it must be understood that sometimes the goals like the ones specified below will play the role of a subgoal, and sometimes they must be considered as standing on their own.

3.2.1 System integrity and data confidentiality

The primary purpose of information system security is to take care of the integrity of the information, keep it confidential, and ensure it is available[1] when needed. In this sense, integrity and confidentiality must be seen from a high level point of view. This is in contrast to message level integrity and confidentiality, which will be considered later in Section 3.4. Thus, the system level integrity and confidentiality may refer to the data in a shared relational data based, a GIS database or an ICCC system.

The system level security decisions are typically *policy* based. That means that the people who are responsible for the system have defined an informal or a formal *secu-*

1. Availability is beyond the scope of this study.

rity policy that specifies who are the legitimate users of the system, and what they are allowed to do.

Thus, typically the primary purpose of a cryptographic protocol is to ensure that the system confidentiality and integrity, as dictated by the applicable security policy, are not compromised by the communication channel protected by the protocol. This is often achieved by integrating the cryptographic protocol into the authorization and possibly audit trail systems. However, this can sometimes be dangerous if the primary goals are forgotten.

In contrast to formalisms we will be using, the prevailing techniques to study confidentiality (and integrity) seem to be information flow and inference analysis. In the models used, there typically are security classes, information containers (variables/files/etc.), and information elements (values). These are used to study whether there is any possibility for higher level information to end up in lower level containers (leakage) or, sometimes, for lower level algorithms to modify higher level information (loss of integrity) [29]. However, it does not seem to be clear how — if in any way — these formalisms should be applied to the study of cryptographic protocols.

3.2.2 Authorization, audit trail and intrusion detection

Ignoring the issues of information flow and inference analysis, the primary means to maintain the integrity and confidentiality of system data are authorization functions and intrusion detection mechanisms. An authorization function is a security subsystem within a computer system that tries to ensure that all operations performed within the system are legitimate from the high level security policy point of view. An audit trail is a gathering of security related events collected from all parts of the system. An intrusion detection mechanism, on the other hand, is a subsystem that tries to find any suspicious behaviour by inspecting the data colleted by the auditing subsystem.

From a cryptographic protocol point of view, authorization seems to be an important but relatively little studied area. Most of the work seems to have concentrated around how to extend an already existing identification system to carry authorization data as well (e.g. [57, 73, 85]), or more on the practical aspects of distributed authorization and delegation (e.g. [65, 81]). However, recent work conducted independently at AT&T Bell Labs on the PolicyMaker concept [15] and at MIT on the SDSI system [95] seem to be quite promising also from a formal point of view. In addition to that, these research systems are turning into real security infrastructure in the Internet in the form of the Simple Public Key Infrastructure, or SPKI, which is being defined by the IETF [36].

The main difficulty in the area of distributed authorization and access control seems to be the concept of delegation and its variations. As briefly mentioned earlier, it seems to be beneficial to treat all authorization information delegated. This also means that the actions of delegating authority and checking authority should be considered completely distinct. At least, this seems to be the case with PolicyMaker.

The areas of audit trail and intrusion detection facilitate postmortem analysis of breaches in confidentiality and integrity. These areas are beyond the scope of this study.

3.2.3 Peer identification

Mutual or unidirectional identification of communicating parties often seems to be a primary goal for cryptographic protocols. Even though there are good reasons to argue that in most cases the purpose of peer identification is to facilitate authorization and thereby securing confidentiality and integrity, there seems to be situations where

identification is a goal of its own. For example, in an electronic mail system the identity of the sender of the message is often equally or more important than the integrity and confidentiality of the message.

However, what is often ignored when considering peer identification is whether the identification should only be authentic or also non–repudiable. In the case of personal email, authentication is often enough. There are seldom if ever cases where a private person wants to sue someone because of a sent email message. However, if email is used to negotiate and agree about a business contract, it is reasonable to require the messages to be non–repudiable. Thus, if there is any argument about the intended meaning of the contract, the record of negotiations can be presented as evidence in the court.

The question of anonymity. In many cases, identification seems to be a controversial issue. Especially in the case of private persons using commercial services it seems to be that the user does not benefit from identification at all while the service provider can greatly benefit from knowing the customer's identity. The identification information allows the service provider to collect large amounts of data about the users, and they can use this data for marketing and other purposes. Commercial decisions and deeds based on these kind of data are often discriminating, irritating or in other respects harmful to the individuals or the society.

Thus, based on the argumentation above, we claim that

- in most cryptographic protocols identification is an undesirable "goal",
- in some (very few) protocols authentication of identity is required, and
- in a fair number of protocols identification should be non-repudiable.

In concrete terms, we claim that in many cases where a cryptographic protocol seeks for peer identification it should be checking the authority of a party or in other means insure the integrity of data.

3.2.4 Authentication and non-repudiation

In the terminology used, authentication and non-repudiation are seen as adjectives used to measure one's confidence and ability to transfer confidence. From this point of view, we can define six types of security data as given in Table 6. These dimensions are often combined; for example, in the case of non–repudiable event record (audit trail) it is usually desirable to have both the occurrence of an event *and* the identity of involved parties non-repudiable.

TABLE 6. Measures of authentication and non-repudiation

	Only authenticated	Non–repudiable
Authorization information	Bob is sure that the peer party requesting an action is permitted to perform the action, but he cannot prove it. However, he doesn't necessarily know that the requestor is Alice.	Bob is sure and can prove that the peer party requesting an action is permitted to perform the action. However, he doesn't necessarily know that the requestor is Alice.
Audit trail information	Bob records that an action has been performed, but the record is based only on Bob's word.	Bob can prove that the action has been performed.
Identification information	Bob knows that it is Alice, but he cannot prove that.	Bob knows that it is Alice, and he can prove it.

3.3 Intermediate level protocol goals

Whereas system confidentiality and integrity are clearly high level goals, and authorization, identification and audit trail with various levels of authenticity and non–repudiability sometimes are subordinate to them and sometimes clearly represent goals of their own, we will next consider *intermediate level* protocol goals. These are measures that do not directly contribute to confidentiality or integrity nor are immediately connected to other high level goals, but allow protocols to be combined so that the higher level goals will be fulfilled.

The intermediate level goals are always almost "meaningless" when treated isolated, i.e. their relevance to the security of the whole system is not necessarily apparent. They typically focus more or less on the level of formal reasoning, and are explicitly expressed in formal models. Actual intermediate level goals used within a given formal framework depend on the high level goals and the framework itself.

Establishing a session key is a common goal which allows all subsequent traffic on a connection to be encrypted, authenticated or both. Key agreement, confirmation, freshness and secrecy will be considered first. Correspondence and secrecy are two intermediate level goals formalized by Woo and Lam [114]. Correspondence refers to full determinacy, whereas secrecy aims to formalize information flow. These will be considered in Section 3.3.2, after the properties of keys.

3.3.1 Key agreement, confirmation, freshness and secrecy

The most common intermediate goal for a cryptographic protocol is to establish a shared session key or some other shared key. There are several purposes for key exchange, the most common of which is to protect forthcoming traffic between the communicating parties, i.e. to create a new secure channel. Other purposes are, for example, to facilitate a possibly forthcoming reauthentication, to establish a security context for delegation of rights within an on-line delegation framework, and to use the key as a shared key in a subsequent different cryptographic protocol.

It is typically required that the parties *agree* on a key, and that the key is *confirmed*, *fresh* and *secret*. Agreeing on a key means that both parties have the same key value. Confirmation denotes that both parties *know* that the other party has the same key value as they have. Freshness indicates that the key value is new, i.e. that is has been generated during the protocol run. Being secret refers to the fact that the key may *only* be known by the involved parties (plus possibly a trusted third party).

To give an idea how to express these properties formally, let's consider a (partial) protocol model with parties Alice, Bob and Eve, a local state variable k_a for each agent a, a set of local information l_a containing all the information a party knows or can compute, and the already familiar actions generate, send and receive. The knowledge operator a knows is used to denote intensional knowledge.

Agreement: $k_{\text{Alice}} = k_{\text{Bob}}$

(The key Alice has is the same key that Bob has)

Confirmation: Alice knows $(k_{\text{Alice}} = k_{\text{Bob}}) \wedge$ Bob knows $(k_{\text{Alice}} = k_{\text{Bob}})$

(Alice and Bob both know that the keys are the same)

Freshness: $\exists s_i : s_i = \text{generate}(k_{\text{Alice}})$

(The key has been generated during the current run)

Secrecy: $k_{\text{Alice}} \in l_a \rightarrow a \in \{\text{Alice, Bob}\}$

(The key is possessed or can be computed only by Alice or Bob)

3.3.2 Correspondence and secrecy

Woo and Lam introduced the concepts of correspondence and secrecy as intermediate level protocol goals in [114]. In their terminology, correspondence is the property of the protocol that after a successful protocol execution, the protocol parties must have proceeded in locked-step fashion. This is connected to authenticity and identification issues. Secrecy, on the other hand, specifies that certain information (e.g. session keys) is not accessible to any outside party (e.g. an intruder).

More specifically, correspondence can be specified using *correspondence assertions*. Such an assertion effectively requires that a global transition in a given set strictly precedes a given action. For example, let's consider a situation where Alice wants Bob to authenticate his identity to Alice. Now, correspondence requires, for example, that if Bob sends a message contributing to the authentication to Alice, Alice must have earlier sent a request requesting Bob to do so.

In the model of [114], secrecy is divided in two parts: *general secrecy* and *specific secrecy.* A *general secrecy condition* requires that an intruder cannot discover any secret information except through an explicitly modelled compromising action. The specific secrecy is expressed with *security assertions*. A specific assertion states who (and only who) may possess a secret piece of information after a protocol run. For example, in the Otway-Rees protocol, it can be required that in the end of the protocol, if a party has a key shared by Alice and Sue, the party *is* Alice or Sue.

Woo and Lam have developed their own formalism to denote these properties. However, standard branching time temporal logic seems to be quite suitable for the purpose.

3.4 Message and data item level goals

Message and datum level properties are the building blocks which all the intermediate and high level goals are based on. They indicate which protocol messages or parts of them have been protected. These properties will be indicated in various ways in the formalisms that will be used.

Message level protection itself is achieved using one or more cryptosystems (cryptographic algorithms), and is not of interest here. In this study, we will usually assume that the level of protection achieved with the use of cryptosystems is absolute. That is, we ignore the possibility of breaking message level protection by cryptanalysis. However, we will consider what kind of properties will make it easier to break the cryptosystem, and how to make breaking it harder.

3.4.1 Message integrity and identification of the originator

Integrity and authenticity of origin refer to the ability to detect that a message has not been modified during transit and the ability to believe in the semantic meaning of a message. A party may consider a message integral (i.e. not modified) but still not

consider its origin authenticated, i.e. keep it possible that the information conveyed in the message is not necessarily true. Furthermore, a party may consider a message's origin authenticated, i.e. believe to know where the message was originally created, but still choose not to believe in the semantic meaning. Thus, a party's ability to believe in the meaning conveyed by a message depends on the integrity of the message, on the authenticity of the origin of the message, and on the level of trust the party is able to place in the originator.

The usual methods to achieve message level integrity are digital signatures, MACs and other uses of hash functions. In [19] Boyd and Mao argue that integrity and authentication of origin should always be protected with hash functions and never by encrypting the data. The basic reason behind this is to make cryptanalysis harder. We will elaborate this below at section 3.4.4, "Redundancy at message level", on page 43.

In particular, integrity is the property of a message being unmodified, i.e. having, on receipt, the same format and information contents that it had on sending. Message level integrity alone is a relatively weak property; it does not say anything about *who* sent the message in the first place. However, if the alleviated sender of the message can be directly seen or indirectly determined (e.g. by using a session id) from the message, and the origin can be *authenticated*, a stronger notion is achieved. In this case, the recipient can be sure that the message was (once) generated by a known protocol party. Furthermore, if the protocol party is *trusted* only to send meaningful messages, it can be inferred that the party believed in the knowledge conveyed by the message at the time the message was sent (cf. freshness below).

Most of the formalisms covered by this study express message level integrity only indirectly. For example, in most authentication logics there are a number of deduction rules that effectively express that whenever the use of cryptographic functions ensures that the message integrity and origin can be authenticated (and the message is fresh, see below), the recipient can believe in the knowledge conveyed by the message.

Example. Let's consider a message interpretation rule of the GNY–logic [46], restated in a more easily readable notation:

$$\frac{a \text{ receives } h(x,k) \wedge a \text{ has } (x,k) \wedge a \text{ believes}(a \overset{k}{\leftrightarrow} b) \wedge a \text{ believes } (\text{fresh}(x) \vee \text{fresh}(k))}{a \text{ believes}(b \text{ conveyed}(x,k)) \wedge a \text{ believes}(b \text{ conveyed } h(x,k))}$$

This expresses that if a party a, e.g. Alice, receives a message that contains a hash function over x and k, she has both x and k (maybe because she just received them), she believes that k is a valid key to protect traffic between her and another party b, e.g. Bob, and she believes that at least one of x and k is fresh, then she can believe that the party b sent the hash value, thereby ensuring integrity of the message. The latter fact, i.e. ensuring the integrity, is expressed by the first half of the consequence: Alice believes that Bob conveyed the pair (x, k). Thus, effectively, Bob ensures the integrity of x.

3.4.2 Confidentiality

In cryptographic protocols that do not perform the actual data transfer but e.g. establish a security context for it, there is usually very little data that has to be kept confidential. Typical examples are session keys and sometimes nonces used to achieve authentication of the identity of the parties.

According to [19], one should only encrypt data that *must* be encrypted, and try to make sure that everything encrypted is statistically random. This helps to keep the

unicity distance of a cipher large[1], thereby making cryptanalysis harder. Furthermore, it should be enforced that a datum encrypted will never appear in plaintext.

Authentication logics generally do not explicitly consider confidentiality issues. The main use of confidentiality, namely secrecy of keys, is usually considered an implicit property of *good* keys. For example, the formula a believes $a \overset{k}{\leftrightarrow} b$ denotes that a believes that k is a good key for her and b. This, among other things, implies that k is only known by a, b and maybe by a trusted server, say Sue.

Many model checking approaches, on the other hand, have a strict notion of confidentiality. A model in such an approach usually contains an explicit notion of the adversary, or the environment Eve. A datum being confidential means that it is only available at the local states of the intended parties. Now, if the datum ever appears at the local state of the environment, the secrecy of the datum has been compromised.

A couple of basic problems in defining confidentiality is to exactly specify what confidentiality means in the given setting, and what are all the possible actions that can lead to loss of confidentiality. For example, in some situation it may be perfectly fine if a trusted server Sue knows a key in addition to the actual parties Alice and Bob, but in another situation this would be considered a catastrophic failure. Similarly, in some situations it can be assumed that Bob will never reveal to Carol any secrets Alice has given to him; in others, Bob may be eligible to do so.

Thus, it seems to be inherently more difficult to model and analyse confidentiality than integrity at the message level. It would be nice to better understand the reasons behind this.

3.4.3 Freshness, timeliness, nonces and replay prevention

The reason behind sending messages within a protocol is to convey meaning (i.e. knowledge or information). Such a meaning typically has temporal properties. In particular, most messages will remain meaningful, or retain their original meaning only for a limited period of time. A message is said to be *fresh* if it still has its original intended meaning when received.

Freshness is deeply connected to message sequences. First, a message may become obsolete due to its original sender sending a new message. Second, the freshness of a message is affected by message sequences in the first place. For example, if a received message contains a nonce, i.e. a freshly generated random number, which is known to be generated during the current protocol run, there are good chances that the message is fresh. Thus, freshness can be seen as a function of message sequences and knowledge about fundamentally fresh events.

Nonces are the most usual method to ensure message freshness. A protocol effectively contains a challenge-response pair where one party (say Alice) generates a new random number, which is the nonce. The party conveys this number to the other party (Bob) and expects a reply containing information about the nonce. The nonce and the reply are usually cryptographically connected to each other in such way that only the peer (Bob) can have generated the reply.

1. Usually the key (or other data) encrypted will be later used to protect other data that has redundancy, thereby making it at least theoretically possible to determine whether a guess for the key value is possible or not. Therefore it is not possible to keep the unicity distance of cipher infinite even if the key protected were totally random.

Li Gong has attempted to formalize freshness [45, page 11]. His *Principle of Message Freshness* states that a party, let us say Alice, can believe a message to be fresh iff

1. the message is originated by herself and she cannot have used an identical message before, e.g. the message contains a newly generated nonce, or
2. it is highly unlikely that the message could have been constructed (by anyone) without some information that can only be gained through a fresh event, e.g. receiving a nonce (from Alice), or
3. the party can base her belief on the authority of some trusted party (e.g. Sue), who explicitly denotes to believe in the freshness of the message.

Furthermore, Gong argues quite convincingly [45, page 66] that using timestamps to ensure freshness is doomed to fail in most situations. In particular, he shows that using time to prove freshness is risky unless all clocks are well synchronized at all times. Already before his observation it was well known that if the clock of the checking party is running too slow, there is a window for replay attack. Gong shows that if the clock of the party creating the timestamp is, or *has been*, running too fast, there is also a possibility for forgery. The alerting issue here is that a clock may *never* (i.e. not just at the time of the forgery) run too fast; this may be hard to achieve in practice.

3.4.4 Redundancy at message level

Redundancy is a key factor to ensure message integrity and to harden message confidentiality. Basically, adding the result of a cryptographic hash function (or other integrity preserving data) to the message is adding redundancy to the message. In other words, integrity means redundancy, in particular, authenticated integrity means redundancy that is hard to reproduce without some secret information. On the other hand, lack of redundancy in encrypted data increases confidentiality. If an encrypted datum is completely statistically random, there is no way for an attacker to determine the plaintext value of the datum or the encryption key from the message alone.

Li Gong [45, page 10] makes a distinction between *explicit* and *implicit* redundancy. According to Gong, if anyone can recognize the redundancy (after decrypting the message), the redundancy is explicit. On the other hand, if the recognition of redundancy can only be performed by the intended recipient(s), i.e. the party (parties) possessing some secret information, the redundancy is implicit. From the confidentiality point of view, only explicit redundancy must be considered harmful.

As already mentioned, Boyd and Mao [19] argue that encryption should only be used on data that has as little redundancy as possible, while integrity should be preserved by hash functions. In the light of Gong's notion, this can be extended slightly. Now, it seems to be beneficial to require that anything encrypted should only contain *implicit redundancy*, while message integrity is to be preserved with explicit redundancy in the form of hash functions or similar mechanisms.

The issue of redundancy is connected to attacks called verifiable-text attacks. A verifiable-text attack [45, page 73] is an attack where a cryptographic protocol can be compromised via an exhaustive search of some small key space. Typically the item that an adversary tries to determine is a user's password or some key directly determined by it. If a protocol somehow contains a possibility for the adversary to gather messages which can be used to test validity of different guesses, there is a possibility that the adversary will sooner or later find out a good candidate for the correct password (or other item) with high probability. Gong gives an explicit algorithm to determine whether a protocol is apparently vulnerable to verifiable-text attacks. However, there are situations where combining protocols (e.g. using the same password in two

different identity authentication systems) may open a possibility for verifiable-text attacks even though both protocols alone were immune to such an attack. [45]

3.5 Summary

In this chapter we have discussed the goals of cryptographic protocols at various levels. From the system point of view, the purpose of a cryptographic protocol is to help in maintaining the overall availability, confidentiality and/or integrity of the information system. Sometimes authorization, audit trail, intrusion detection and peer identification can also be considered as highest level goals, even though typically their purpose is to contribute to the system level goals.

At an intermediate level, or protocol level, the most important aspects are agreement, confirmation, freshness and secrecy of keys. From another point of view, some of these properties can be modelled by using the concepts of correspondence and secrecy as introduced by Woo and Lam [114].

at the lowest level, looking at an individual message or a part of it, important aspects are message integrity, authentication of origin, confidentiality and freshness. Redundancy plays an important role when considering integrity and confidentiality properties.

PART II

CHAPTER 4 *Modal logic* *47*

Syntax and semantics 48
Logics, proofs and consistency 51
Some standard logics 53
Logics of knowledge and belief 55
Temporal logic 60
Combining knowledge and time 63
Summary 68

CHAPTER 5 *Model checking and Process Algebra* ... *71*

Introduction to models of concurrency 72
Process graphs and Labelled Transition Systems 74
Algebraic approaches 76
Semantics 85
Model checking 91
Summary 93

CHAPTER 4

Modal logic

Modal logics are formal logics usually based on first order propositional or predicate logic. As we have already seen, modal formulae can be used to express knowledge, beliefs, temporal relations, and other similar modalities. For example, we can use a temporal formula to express that a session key is never (i.e. not in the past, not now, nor in the future) communicated in clear. Similarly, a knowledge formula may express that Alice knows that Bob knows that Alice has the key value k in her possession. Temporal and knowledge formulae can also be combined; we will return to this in section 4.6, "Combining knowledge and time".

Modal logics are generally considered to be originated in the work of C. I. Lewis early in this century [43, page 15]. They were initially used to express different kinds of modalities like necessity, possibility and obligation. Modal logics were later developed by Kripke who introduced the model theory used in this work. Jaakko Hintikka's seminal work *Knowledge and Belief*, which appeared in 1962, is the first book length treatment of the logic of knowledge. The study of modal logics gained more popularity from the beginning of the 1980s as its application to distributed computer systems began.

The knowledge and beliefs we will be discussing shall be considered intensional, external concepts. That means that a protocol party does not necessarily contain any data structures or computational capabilities to infer the knowledge or beliefs it is said to pertain. Instead, knowledge and beliefs are used to express assumptions the protocol designer or a protocol analyser may attribute as the reasons behind a party's behaviour. Since the purpose of a protocol is often to transfer knowledge, knowledge formulae are especially suitable for expressing protocol goals.

The purpose of this chapter is to motivate the reader and introduce enough of model theory so that the reader will be able to understand the relation of protocol models to modal formulae. The models used will work as frames for multimodal logics with different modalities for time and knowledge or belief. We will be considering multi-agent systems where each agent has its own knowledge or belief operator; the time modalities are the same for all, even though we do not assume the existence of a global clock. Our time structure is discrete and branching. Thus, only those parts of the theory of modal logic that are needed to understand these concepts are considered. For readers more interested in modal logic in general we direct to any of [24, 37, 43].

The rest of this chapter is largely based on *Reasoning about Knowledge* by Fagin, Halpern, Moses and Vardi [37], and *Logics of Time and Computation* by Goldblatt [43]. The basic ideas about knowledge based programming and multi-agent systems are adapted from the former, while the latter — being more concise — has been the main source of formal denotation.

4.1 Syntax and semantics

In the next few sections we will consider both ordinary modal logics, or modal logics with just one modality, and multimodal logics, i.e. logics with several separate modal operators. There is considerably less work published on multimodal logics than with logics with just one modality. This unfortunate fact affects our presentation on some occasions.

The language for a propositional (multi)modal logic is built from a set of atomic formulae (propositions) Φ, the usual propositional operators $\neg\ \wedge\ \vee\ \rightarrow\ \leftrightarrow$ and the truth constants $\bot$ (falsum) and T (verum), and one (or more) modal operators. The standard modal operators are the "box" $\Box$ and the "diamond" $\Diamond$, which are often, but not always, defined in terms of each other. Fagin et al. [37] have adopted the notion of using an operator $K_a\varphi$ to mean that the agent a knows the formula φ, and a similar operator $B_a\varphi$ for beliefs. In this text we use the convention introduced in [108] of expressing knowledge and beliefs explicitly as a knows φ and a believes φ. To express temporal modalities, the standard discrete time operators $\Box\Diamond\ \bigcirc\ \mathsf{U}\ \Box\!\!\downarrow\Diamond$ will be used (see Table 7 on page 60). All the modal operators will be momentarily discussed and defined formally.

Example. Using the language introduced we can express, as examples, some facts about keys:

- k is never expressed in clear:
  $\Box(\text{send}(m) \rightarrow k \notin \text{submsg}(m))$

  where $\text{send}(m)$ indicates that the message m is sent during the current round, and $\text{submsg}(m)$ is the set of (unencrypted) submessages of m. The modal operator $\Box$ denotes that the formula inside it is *always* true, i.e. at all occasions it holds that if any message m is sent, k is not contained in it as plaintext.
- Alice knows that Bob knows that Alice has the value k:
  $\text{Alice}\ \mathsf{knows}\ \text{Bob}\ \mathsf{knows}\ (k \in \text{has}_{\text{Alice}})$

  where $\text{has}_{\text{Alice}}$ is the set of data Alice has in her possession.

4.1.1 Frames and models

The model theory of modal logics is based on the concepts of frames and models. A frame is a relational structure representing a number of states, or worlds, and relationships between them. A model, on the other hand, is a frame augmented with a valuation function. The valuation function gives the truth value, i.e. true or false, for each basic proposition, i.e. the members of Φ, at all the states of a corresponding frame.

Formally, a multimodal *frame* is an $n+1$-tuple $F = (S, R_1, \ldots, R_n)$ where S is a non-empty set (of the states) and $R_1, \ldots, R_n$ are binary relations on S: $R_i \subseteq S \times S$. Given a set Φ of propositions, a *model* on a frame is a n+2-tuple $\mathcal{M} = (mpS, R_1, \ldots, R_n, V)$ where V is a valuation function $V : \Phi \rightarrow 2^S$. The valuation

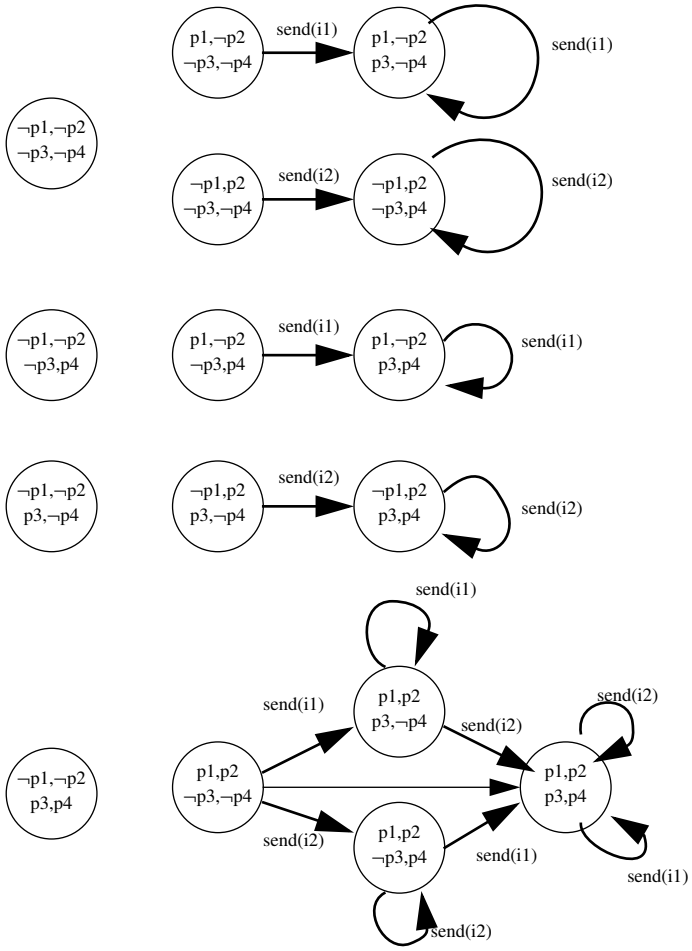


FIGURE 7. Temporal relations in the simple protocol example

defines the truth value of the elements of Φ at the various states. That is, $V(p)(p \in \Phi)$ can be thought as the set of states where the proposition p is true.

Example. Let us consider a very simple protocol model with Alice and Bob. The local state of Alice and Bob is modelled as a set K_a that represents the explicit information content of a a, a being Alice or Bob. In this simple model, Alice and Bob are only able to possess an arbitrary number of noninterpreted (large) integers. For example, Alice's local state might be

$$K_{\text{Alice}} = \{i_1 = 342457621567678, i_2 = 236523567732567\}$$

indicating that Alice has in her possession, the two numbers i_1 and i_2. The only possible action in the protocol is send(i) where Alice sends Bob the integer i.

Now, let's consider a model where Alice and Bob can only possess the two integers above, and nothing else. Let K_{Alice} the set of integers Alice knows, and K_{Bob} the set of integers Bob knows. In this case, the propositions Φ can be defined as following:

- $p_1 = (i_1 \in K_{\text{Alice}})$
- $p_2 = (i_2 \in K_{\text{Alice}})$
- $p_3 = (i_1 \in K_{\text{Bob}})$
- $p_4 = (i_2 \in K_{\text{Bob}})$

Thus, the only possible states of the system are

$$S = \{\langle \neg p_1, \neg p_2, \neg p_3, \neg p_4\rangle, \langle \neg p_1, \neg p_2, \neg p_3, p_4\rangle, \ldots, \langle p_1, p_2, p_3, p_4\rangle\}$$

Given the only possible action $\text{send}(i)$, we can define a relation R_F for future time. First, we will say that two possible states of the system $s_1, s_2 \in S$ are in immediate succession, $s_1 R_{\text{send}} s_2$, iff there is an action $\text{send}(i)$ such that at $s_1 : i \in K_{\text{Alice}}$ and at $s_2 : i \in K_{\text{Bob}}$. That is, Alice can send Bob i if she has it, and Bob have just received it if he has it in the next state. Now, in this context, our (somewhat unusual) definition of time R_F is the transitive closure of R_{send}. Both R_{send} (thick arrows) and R_F (all arrows, both thick and thin) are illustrated in Figure 7.

The reader should note that the model $M = (S, R_F, V)$ is only one possible model among several. The most apparent variations are ones where the set of possible states is a subset of states given above. For example, we can leave out all states where Alice initially knows nothing as uninteresting. If we do so, we have to restrict R_F and V appropriately as well.

4.1.2 Truth value of formulae

A statement saying that the formula φ is true (holds) at a state $s \in S$ in a model M is denoted as

$$M \models_s \varphi$$

It's truth value is defined inductively as follows:

$M \models_s p$	iff	$s \in V(p)$
$M \not\models_s \bot$		(i.e. never $M \models_s \bot$)
$M \models_s \varphi_1 \rightarrow \varphi_2$	iff	$M \models_s \varphi_1$ implies $M \models_s \varphi_2$
$M \models_s \Box \varphi$	iff	for all $t \in S$, sRt implies $M \models_t \varphi$

where the modal operator $\Box$ is defined by the relation R. If there are several modal operators, each of them is defined separately. The truth values of the other propositional formulas with the standard operators $\neg \wedge \vee \leftrightarrow$ are defined as usual using falsum and implication.

Example. Given the definition, the truth values of normal propositional formulae at the various states of the example should be clear. Now, using the example model, we can used the relations R_F and R_{send} to defined the temporal operators $\Box$, henceforth, and $\bigcirc$, next, as follows:

$M \models_s \Box \varphi$	iff	for all $t \in \{u \mid sR_F u\}$, $M \models_t \varphi$
$M \models_s \bigcirc \varphi$	iff	for all $t \in \{u \mid sR_{\text{send}} u\}$, $M \models_t \varphi$

Thus, for example, $p_i \rightarrow \Box p_i$ expresses that if either Alice or Bob possesses something, they will possess it also in all future times. The truth of this formula can be determined by examining all the states in the model, evaluating the values of p_i, $\Box p_i$ and $p_i \rightarrow \Box p_i$ in all states. Doing this reveals that the formulae are true in all states. Thus, we can say that the formulae are *true in the model* M.

Similarly, $p_1 \wedge \neg p_2 \rightarrow \bigcirc p_3$ denotes that if Alice knows the first integer but not the second, Bob knows the first integer in the next state. This will also appear true, since if Alice does not know the second integer, the only action she can perform is sending the first integer. Thus, all actions (i.e. the only possible action) lead to state where Bob also knows the first integer.

4.1.3 Truth and validity

A formula φ is said to be *true in model* M, if it is true in all states of the model:

$$M \models \varphi \text{ iff } M \models_s \varphi \text{ for all } s \in S$$

A formula φ is *valid* in a frame $F = (S, R_1, \ldots, R_n)$ if it is true in all models based on F.

For example, the formula $p_i \rightarrow \Box p_i$ is true in the example model. However, it is not valid since it is easy to make a new valuation where the truth values of p_i are assigned differently[1]. A formula can also be valid in a class of frames, in which case it is valid in all frames belonging to the class.

4.2 Logics, proofs and consistency

In the section above we defined a formal language based on a set of propositional formulae Φ. The next step is to define a logic, or a set of formulae that include all propositional tautologies and is closed under the rule of detachment, that corresponds to a given set of models or frames. We will skip the details of definitions of various kinds of logics, and concentrate on the issues of theoremhood, soundness, completeness and consistency. The style is practical and deviates somewhat from the standard presentation. The deviations are made to ease the reading for an uninitiated reader, and noted in footnotes where appropriate. For a more formal definition, see e.g. [43].

4.2.1 Logics

From our practical point of view, a logic is a set of formulae that are true in a given set of models or frames[2]. For example, the set of predicate logic tautologies PL is a logic, since all tautologies are true in all states in any model. However, the logic PL is not very interesting from our point of view, since it does not consider the formulae that are relevant for our discussion. On the other hand, we can define the set of true formulas in a given model as a logic:

$$\Lambda_M = \{\varphi | M \models \varphi\}$$

However, such a logic is not necessarily decidable or even axiomatisable. As our interest in logics is of practical nature, i.e. as we would like to use logical deduction to determine whether a given formula is deductible within the logic, this is undesirable.

1. What such a model might express is a completely different issue, and not considered here.
2. Thus, we ignore unsound logics from the very beginning.

4.2.2 Theorems, axioms and provability

A formula that is a member of a logic (and therefore true[1]), is called a theorem. The theoremhood of a formula is denoted as $\vdash_\Lambda \varphi$ meaning that φ is a member of the logic Λ, i.e.

$$\vdash_\Lambda \varphi \quad \text{iff} \quad \varphi \in \Lambda$$

The logics we will consider will be axiomatisable. This means that we can define a set of initial theorems, or axioms, and inference rules in such a way that all the theorems can be proven by applying the inference rules and axioms. More specifically, let AX be an axiom system, i.e. a set of axioms and inference rules. A *proof* in AX is a sequence of formulae, each of which is either an axiom (or an instance of an axiom schema), or follows by an application of an inference rule from the formulae earlier in the sequence. To denote that the formula φ is provable in AX, i.e. has a proof in AX, we write $AX \vdash \varphi$.

For example, the smallest of the so called normal modal logics is the logic called K. K can be given in terms of an axiom system consisting of two axiom schemata and two inference rules:

1. All tautologies of propositional logic are axioms in K.
2. $\Box\varphi \wedge \Box(\varphi \rightarrow \psi) \rightarrow \Box\psi$ (Distribution Axiom)
3. From φ and $\varphi \rightarrow \psi$ infer ψ (Modus ponens)
4. From φ infer $\Box\varphi$ (Necessitation or Knowledge Generalization)

Axiomatisability and logical proofs make it possible to reason about systems whose models are infinite or so large that model checking would be infeasible (Compare to Chapter 5, “Model checking and Process Algebra”). Given a proof for a formula, it is straightforward to check that the proof applies, i.e. to check that the formula is a theorem. However, the problem of finding a proof for a true formula is not easy; in fact, there are axiomatisable logics that are undecidable. However, that goes beyond the scope of this study.

4.2.3 Soundness and completeness

A logic (or an axiom system) is said to be *sound* with respect to a model (or a frame, or a class of frames), if all theorems are true in the model (respectively, valid in the frame or the class of frames), i.e. if for all formulae φ

$$\vdash_\Lambda \varphi \qquad \text{implies} \qquad M \models \varphi$$

The logic Λ (or a corresponding axiom system) is *complete* with respect to a model M if all true formulae are members of the logic (can be proven), i.e. if for all formulae φ,

$$M \models \varphi \qquad \text{implies} \qquad \vdash_\Lambda \varphi$$

If a logic is sound and complete (with respect to something), it is said to be determined (by that something).

So far, all logics we have considered (and even our definition of logic) have been implicitly sound. From now on, we will explicitly only consider sound logics, unless otherwise stated. In this chapter there will be a number of logics which are known to

1. We still assume that all logics are sound.

be complete; later on, most of our logics will be incomplete, though. In practical term this means that there will be true facts about our protocol models that cannot be proven within the logic used.

4.2.4 Consistency

A formula φ is said to be *consistent* with respect to Λ if $\neg\varphi$ is not provable in Λ. Equivalently, adding an *inconsistent* formula to a logic makes it possible to prove $\vdash \varphi \wedge \neg\varphi$, which makes it possible to prove anything (one can infer anything from false assumptions). A logic (i.e. set of formulae) is said to be consistent if $\not\vdash_{\Lambda} \bot$, i.e. it is not possible to prove impossible in the logic. Inconsistent logics are not of interest to us.

Because most of our logics will be incomplete, there will be cases where one can add φ and $\neg\varphi$ to the logic yielding two different consistent logics. Though, of course, it is not possible to add *both* φ and $\neg\varphi$, and still get a consistent logic.

4.3 Some standard logics

We are usually interested in logics that are sound (and possibly complete) with respect to some *class* of models or frames. It is not very useful to develop a logic for a single protocol model; for the next protocol, a new logic should be developed. Fortunately, there is a pretty well established taxonomy and theory of modal logics having a single modality. The main difference between different kinds of modalities is in the structure of the relations R_i defining the modalities. However, there seems to be less information about multimodal systems having several relations within the same frame or model.

4.3.1 Well-known axioms

The standard logics are usually described by the names of so called well-known axioms. Most of these axioms also correspond to simple first-order properties of the respective relations R.

Name	Schema	Name in epistemic logic	Property of corresponding relation
K:	$(\Box\varphi \wedge \Box(\varphi \rightarrow \psi)) \rightarrow \Box\psi$	Distribution axiom	valid in all frames
T:	$\Box\varphi \rightarrow \varphi$	Knowledge axiom	reflexive
D:	$\neg\Box\bot$	Consistency axiom	serial
4:	$\Box\varphi \rightarrow \Box\Box\varphi$	Positive introspection axiom	transitive
B:	$\varphi \rightarrow \Box\neg\Box\neg\varphi$		symmetric
5:	$\neg\Box\varphi \rightarrow \Box\neg\Box\varphi$	Negative introspection axiom	euclidean
L:	$\Box(\varphi \wedge \Box\varphi \rightarrow \psi) \vee \Box(\psi \wedge \Box\psi \rightarrow \varphi)$		weakly connected
Dum:	$\Box(\Box(\varphi \rightarrow \Box\varphi) \rightarrow \varphi) \rightarrow (\neg\Box\neg\Box\varphi \rightarrow \varphi)$		

The axioms T, D, 4 and 5 will be used to describe knowledge and belief. Axioms T, 4, L and Dum will be used in describing temporal relations. We will also use some other axioms to describe time.

Example. Let's consider again the above described simple example where Alice sends integers to Bob. By inspecting the relations R_{send} and R_F, we will notice that neither is reflexive, serial or euclidean. The relation R_F is transitive while R_{send} is not. However, if we remove the states where Alice initially has nothing and therefore cannot send anything, the resulting model is serial with respect to both R_{send} and R_F. Considering this latter model, the axioms K, D and 4 describe the modal operator corresponding to R_F.

Understanding the axioms. Some of the axioms have easily understandable, intuitive meanings. In the framework of knowledge and beliefs, T denotes that all known facts are true. This can be replaced by D, which requires that impossible cannot be believed. 4 describes that a party knowing something knows that he knows it, while 5 describes a situation where a party also knows what he does *not* know. This is sometimes undesirable.

In the context of time, T describes modal operators which consider the current time as a part of the operator's range, e.g. *now* and at all future times. Compare this to a similar system without T, where the same operator would mean "at all future times, but not necessarily now". If T is left out from a modal logic, it is necessary to add D (or a formula taking care of a similar function) in order to prevent one from achieving impossible results in the future. Transitivity (4) is a natural aspect of time. If the moment A is in the future, and B is in A's future, then B is certainly in the current future as well. 5 is simply meaningless in this context, and doesn't appear. L, on the other hand, is a property that describes linear time. As we will be using branching time models, L usually doesn't appear in this study. Dum (named after Michael Dummet) is used in section 4.5.1, "Linear time temporal logic", on page 60 to make the logic complete.

4.3.2 The logic S5

S5 is perhaps the best known modal logic. It is described by the axioms K, T, 4 and 5, and corresponds to a relation which is reflexive, transitive and euclidean, i.e. an equivalence relation. In our context S5 can be used to describe knowledge. We will return to this later, but this moment the reader can imagine a protocol model where a party at all times knows that the current global state is one of a number of states (though not which one). When something happens that changes this knowledge, another set of states is considered possible. These sets of states will, however, never overlap, i.e. their intersections are always empty.

When thinking of S5 in the light of the intuitive descriptions of the axioms, it becomes apparent that S5 describes a kind of logically *ideal* knowledge. An agent that knows everything it knows and everything it *does not know* seems slightly unnatural. However, if we think the knowledge expressed by S5 as the things that *can possibly be known*, the situation becomes more natural. In this study we will concentrate on beliefs, and S5 won't be much used.

4.3.3 The logics KT4 and KD45

The logic KD45 can be used to describe beliefs, and KT4 as a partial logic for discrete, branching time. To describe branching time properly, other axioms are also needed; they will be described later. Most of the logics that we will use later in this study are based on these logics.

KD45 is a logic where the T axiom of S5 is "replaced" with D. The underlying relation is no more reflexive, but it is still serial (reflexivity implies seriality). Considering the model in practical terms, whereas in S5 in each state the set of possible states contains the state itself, in KD45 in each state there is *some* (i.e. at least one) state that is considered possible. Looking at the difference from the point of view of knowledge and belief, in S5 every known statement is true, i.e. if $\Box\varphi$, then φ is true indeed, since the current state is included in the set of states that determine the truth of $\Box\varphi$. On the other hand, in KD45 this does not necessarily hold, i.e. in KD45 it is possible to believe in formulae that are false. However, even in KD45 it is not possible to believe formulae that are impossible, or false everywhere in the model.

While KD45 was "reduced" from S5 by replacing T with D, KT4 is "S5 without the axiom 5". The resulting system is no more euclidean, allowing the system to branch into separate futures. However, KT4 as itself is not strong enough to function as a proper model for time alone. For example, nothing requires a KT4 model to be connected whereas usually branching time models contain one common reference point where the time starts to branch. Transitivity, on the other hand, is typically an important aspect of time models.

4.4 Logics of knowledge and belief

In the context of communication protocols, knowledge and beliefs are used to denote facts and assumptions a protocol party has about the current state and about the knowledge and beliefs of other parties. If we think about a distributed system, each node has their own state which contains data, history record and other explicit information the node has gained during the protocol run. The node has no direct access to the local states of other nodes, nor the (abstract) local state of the environment. However, the node may hold (or may be thought to hold) assumptions about the local state of the other nodes (or the environment). These assumptions are typically based on some initial assumptions (that are thought to have been achieved through a mutual preagreement) and the communication events that the node has executed so far. These assumptions are called beliefs. If a node knows for sure that a belief is true (i.e. it does not depend on any assumptions), it can be called knowledge.

For example, in the light of our Alice-sends-Bob-integers example, when Alice has sent Bob an integer, she knows for sure that Bob has the integer, because the protocol model does not allow the integer to be sent unless it is simultaneously received. However, if the communication were asynchronous, i.e. if Alice didn't "know" that all sent messages are simultaneously received, she couldn't know nor even believe that Bob has got the message; the message might have been lost.

An alternative way to think about knowledge and beliefs is to imagine them as measures of certainty a node has about logical formulae describing the state of some other node or nodes. For example, Alice knowing Bob possesses a certain integer means that according to the model Alice has about the state of affairs, she does not consider any such state possible where Bob does not know the integer. However, there may be several global states where Bob knows the integer, and Alice does not necessarily know which one of these several states is the real one. Thus, knowledge and beliefs denote a protocol party's assumptions about the global state of the system. These assumptions affect the future behaviour of the party.

We want to emphasise again that this view of knowledge and beliefs does not mean that a protocol implementation would explicitly encode the knowledge or beliefs in its local state. Rather, the formulae described in this section are used by the protocol designer or external observer to describe the reasons behind the protocol party's

behaviour. That is, the knowledge and beliefs are intensional properties, not explicit information represented in the implementation.

4.4.1 Possible-worlds interpretation of protocol models

The global states in a protocol model denote the possible states of affairs, often called *possible worlds*. At each global state, a protocol party certainly knows its local state. However, since it does not have any direct access to the local state of other elements of the model, it cannot determine for sure in which global state the system is. Thus, is has to consider a number of states, or worlds, possible.

This view of keeping several global states (worlds) possible is represented as a modality in a logic. The relation R_i corresponds to agent a_i 's considerations of what is possible. The possibility relation and knowledge are interchangeable. Given a complete set of knowledge sentences, one can use each sentence to divide the set of all states into a set of possible states and a set of impossible states. The set of states a party considers possible is then clearly the intersection of possible states of individual statements. Respectively, given a possibility relation, one can calculate the value of the assumed statement in the states which are considered possible from the current state. If the assumed statement is true in all possible states, then the knowledge or belief statement clearly holds.

Example. Let's consider our already familiar simplistic protocol model of Section 4.1.1 from Alice's point of view. Initially, Alice knows everything about her own data possessions, but cannot know anything about the state of Bob. Thus, in any initial state where she has not sent anything yet, she has to consider all variations of Bob's state possible.

Due to the synchronicity of the model, whenever she sends a *new* message to Bob, her knowledge increases. This may seem counterintuitive; however, the synchronicity of communication can be thought to contain implicit acknowledgement. That is, whenever Alice sends something to Bob, she simultaneously receives an acknowledgement from Bob that he has got the message.

Now, since Alice's knowledge depends on what she has sent and what she has not, it is necessary to change the set of possible worlds from our earlier model. However, even though it would be possible to add propositions to explicitly express what Alice has sent and what not, this is unnecessary and won't be done.

The resulting set of possible worlds is illustrated in Figure 8 on page 57. In the figure, the dotted arrows represent Alice's considerations about what is possible and what is not, while solid arrows denote actions. In the model, Alice certainly always considers the actual state possible. However, this is not explicitly stated in the figure. The reader can imagine a dotted circular arrow next to each node.

From the figure we can clearly see how, independent of her local state, Alice initially considers four states possible (the exact states differ according to her local state, of course). After sending one integer, the number of possible states is reduced to two. And in the one case where Alice initially knows both the integers, after sending them both to Bob, Alice knows for sure that both Bob and she have both integer values in their possession.

What is interesting in the figure is that several distinct states now have the same truth values for all propositions in the set of propositions Φ. Thus, there are several states where all statements of propositional logic have the same truth value. However, there is at least one modal statement that makes a distinction between these states.

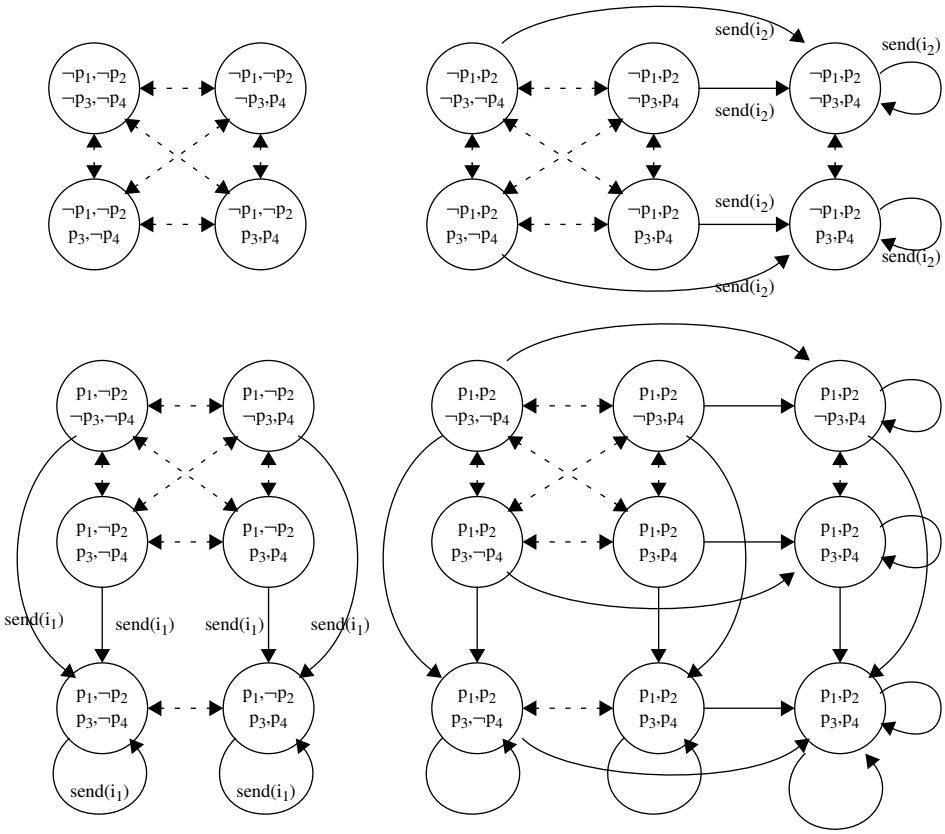


FIGURE 8. Evolving of Alice's knowledge in the example protocol

If Bob's knowledge was considered instead of Alice's knowledge, the figure would be remarkably similar in this case. The actual states would be different, of course, but the overall figure would look like pretty much the same. This is due to the synchronous nature of communication, and does not hold in asynchronous systems or in systems with more than two protocol parties.

4.4.2 Knowledge of different agents

Extending the scope of consideration from the knowledge (or beliefs) of a single protocol party to knowledge of multiple parties complicates the model considerably. A two party synchronous system is still rather straightforward to model since there are only internal actions and synchronous actions that affect both protocol parties at the same time. However, if there are more than two agents, or if the communication is asynchronous (which is roughly equal to introducing a third agent, the network, between the two agents), a protocol party has to consider the possibility of communication between other parties. For example, if there is a malicious attacker Mallory between Alice and Bob in the integer-sending-protocol, Alice has no way whatsoever to know if the integers she has sent have ever reached Bob. In such a case, a cryptographic protocol, maybe something similar to the one given as an example in section 2.5, "A protocol example", is the only remedy.

In a multi-party (or multi-agent) system the protocol model contains distinct possibility relationships for different agents. In other words, in different protocol states different agents consider different other protocol states possible. For example, in a protocol where Alice, Bob and Carol communicate, Alice must always keep it possi-

ble that Bob and Carol are communicating without her knowledge (unless, of course, we state that Alice somehow has a priori knowledge that such communication will not happen). Trying to model such a system yields extremely complex protocol models. However, there are two ways to alleviate this problem: one is to assume that Bob and Carol only communicate according to a protocol, and Alice knows the possible flows of the protocol. The second possibility is to assume that Bob and Carol will communicate everything they know, and thus consider them as one party in the protocol model. The latter method is often used when analysing security protocols.

Fortunately, in most of the protocols under study, one can distinguish one or two trusted protocol parties who are assumed to act according to the specification, and a number of untrusted parties that can be collapsed into one agent along with the network. This makes the protocol models somewhat easier to comprehend and handle.

4.4.3 Knowledge, common knowledge and distributed knowledge

Knowledge, in distinction to beliefs, is generally considered to state true facts. That is, if something is known it is by definition true. In a possible-worlds model this can be stated as a requirement that in all worlds the current world is considered possible. This corresponds to reflexivity of the possibility relation, and to the axiom T. The reason for this is simple: earlier we defined that a modal statement is true if the statement subject to the modal operator is true in all possible worlds. Now, if the current state is possible, and a knowledge statement holds, then the stated fact is certainly true.

Example. In the Alice-sends-Bob-integers model, when Alice has sent the first integer, she knows that Bob has that integer value:

$$\text{Alice knows } p_3$$

Now, since Bob has got the integer from Alice, he knows that Alice knows it as well:

$$\text{Bob knows } p_1$$

Because of the synchronicity of communication, it is clear that Alice knows that Bob knows that she has the first integer, and vice versa:

$$\text{Alice knows (Bob knows } p_1) \wedge \text{Bob knows (Alice knows } p_3)$$

By examining the model, we'll see that this can be continued ad infinitum, i.e. Alice knows that Bob knows that Alice knows that Bob knows that Alice knows etc.

Common knowledge. The example case where Alice knows something, and Bob knows that Alice knows, and Alice knows that Bob knows that she knows etc., is an example of *common knowledge*. In general, common knowledge refers to a situation where a single fact is known by a group, and everyone in the group know that all the others in the group know the fact, that everyone knows that everyone knows etc. Common knowledge is very usual in everyday life; some may say that it is so usual that most people don't pay any attention to it. For example, in a situation where a group of people are discussing an issue, and nobody is sleeping nor is suspected to sleep, the flow of the conversation can be seen and is assumed by the participants to be common knowledge.

Before we formally define common knowledge, we first extend our knowledge operator to groups. The notation

$$\{\text{Alice, Bob, Carol}\} \text{ knows } \varphi$$

denotes that all the members of the group, i.e. in this case Alice, Bob and Carol, each know φ. In other words, whatever the members of the group regard possible, they do consider it impossible for φ to be false.

Group knowledge is not necessarily common knowledge. For example, even though Alice knows that φ, she may consider possible a world where φ is true but where Bob considers possible a world where φ is not true. In such a situation, even if Alice knows φ, she does not know if Bob also knows φ or not.

Formally, group knowledge is defined in a straightforward way. In a given state s of a model M, the group G knows φ if and only if all the members of the group know φ:

$$M \models_s G \text{ knows } \varphi \qquad \text{iff} \qquad \text{for all } a \in G, M \models_s a \text{ knows } \varphi$$

Now, let $(G \text{ knows })^0 \varphi$ be an abbreviation for φ, and $(G \text{ knows })^{k+1} \varphi$ be an abbreviation for $G \text{ knows } ((G \text{ knows })^k \varphi)$. With this, we can define that φ is common knowledge within the group G, G common φ, as

$$M \models_s G \text{ common } \varphi \qquad \text{iff} \qquad \text{for all } k = 1, 2, \ldots \; M \models_s (G \text{ knows })^k \varphi$$

Distributed knowledge. While common knowledge is something that everyone in a group knows, distributed knowledge refers to something that someone would know if he had direct access to the knowledge of all the members of the group. That is, a group G has φ as distributed knowledge "if the 'combined' knowledge of the group implies φ" [37]. Considering the possibility relations, distributed knowledge combines knowledge from all of the relations of the group members, effectively taking an intersection of the worlds that group members consider possible.

Thus, formally, distributed knowledge G distributed φ can be defined as

$$M \models_s G \text{ distributed } \varphi \qquad \text{iff} \qquad \text{for all } t : (s, t) \in \bigcap_{a_i \in G} R_i$$

where the R_i are the possibility relations of the group members.

Example. In the usual Alice&Bob model, the distributed knowledge of Alice and Bob is always the total knowledge of the system; i.e. if Alice and Bob where able to combine their knowledge, they would always know in which global state the system is.

4.4.4 Belief as conditional knowledge

We now return our attention to belief. As noted, in a realistic protocol model, it is seldom possible to know anything significant for sure. All knowledge gained is more or less based on some initial assumptions which may hold or not.

From a formal point of view, the main difference between knowledge and belief is that what is known must be true while beliefs can be false. Using our protocol model, knowledge requires that the actual global state is among the states that an agent considers possible, while belief allows that the agent's knowledge is false, i.e. that he does not consider the actual global state possible.

The usual approach to model beliefs instead of knowledge is to drop the reflexivity requirement of the possibility relation, thereby the axiom T, and to replace it with seriality and the axiom D. Seriality is required to make it impossible to believe false sentences. However, here we take a slightly different approach and model belief as

conditional knowledge. That is, the formula A believes φ is equivalent to A knows$(\psi \rightarrow \varphi)$ where ψ denotes the initial assumption(s).

Given this, we typically only consider worlds (i.e. global states) where ψ holds. Restricting the consideration in this way, and ignoring actions that make any initial condition fail, belief can be handled as knowledge according the rules of S5 logic.

4.5 Temporal logic

Temporal logics are logics having modal connectives that express temporal relationships. The two most usual modal connectives are $\Box$, usually read as "henceforth" (at all future times) or "always" (at all, future and past, times), and $\Diamond$, usually read as "at some time" (future or future/now/past). Those temporal operators we will be using, along with their intuitive meanings, are given in Table 7.[1]

TABLE 7. Temporal operators

Operator	Intuitive meaning
$\Box\varphi$	φ is true now, and will be at *all* future states
$\Diamond\varphi$	φ is true now, or will be true at *some* future state
$\psi \mathrel{U} \varphi$	ψ is true now, and will be at all states *until* a state is entered where φ is true
$\Box\varphi$	φ has been true at all past times
$\Diamond\varphi$	φ has once been true at some past time

The time model used is always discrete. Given a global state, the set of possible actions gives the set of possible futures. A branching time temporal logic will be achieved by including the set of performed actions into the local state of the environment. But, before going to this, we'll consider some general aspects of temporal logics.

4.5.1 Linear time temporal logic

A modal logic defined by discrete, reflexive total orderings as the possibility relation, is a so called discrete Diodorean logic, often called S4.3Dum. This logic is determined by the frame $(\omega, \leq)$ where ω denotes the set of natural numbers. The axioms for this logic are the standard axioms K, T, 4, L, and Dum. T implies that the modal operators include the current state (hence $(\omega, \leq)$ instead of $(\omega, <)$), 4 states that time is transitive, and L denotes that time is weakly connected. The Dum axiom is needed to make the logic complete, and it's role is beyond the scope of this study.

A single run of a protocol can be considered as a model for the linear time temporal logic. The successive states s_0, s_1, … form the total order of states in the frame. However, if there ever is a possibility of different actions in a given state, there will be several runs per protocol. Typically the number of different runs grows so large that it would be infeasible to consider them individually. However, it is possible to give linear time semantics for such branching models. Doing this, the branching structure of the model is ignored, and only the possible outcomes of the computational process are considered. We will briefly return to this point in section 7.2.

1. Note that we have left out the *nexttime* operator $\bigcirc\varphi$ usually included in temporal logics. Leaving it out makes things simpler.

Syntax and semantics. As an example, we'll describe a linear temporal language for our usual models. The language LTL-X consists of atomic formulae Φ, the usual operators and constants of propositional logic, and the temporal operators of Table 7 on page 60. A temporal model is a triple $M = (S, R, V)$, where the possibility relationship R can be considered as a reflexive transitive closure of a relation indicating the possible actions at each state. The meanings of different formulae at the states of the model M are defined inductively in the following way

$M \models_s p$	iff	$s \in V(p)$
$M \not\models_s \bot$		
$M \models_s \varphi \to \psi$	iff	$M \models_s \varphi$ implies $M \models_s \psi$
$M \models_s \Box \varphi$	iff	for all future states $t \in \{u \mid sRu\}$, $M \models_t \varphi$
$M \models_s \Diamond \varphi$	iff	there is future state $t \in \{u \mid sRu\}$ such that $M \models_t \varphi$
$M \models_s \varphi \mathsf{U} \psi$	iff	there is a future state $t \in \{u \mid sRu\}$ such that $M \models_t \psi$ and for all intermediate states $v \in \{u \mid sRu \wedge uRt\}$, $M \models_v \varphi$
$M \models_s \boxminus \varphi$	iff	for all past states $t \in \{u \mid uRs\}$, $M \models_t \varphi$
$M \models_s \diamondminus \varphi$	iff	there is a past state $t \in \{u \mid uRs\}$ such that $M \models_t \varphi$

In this semantic interpretation, the following equalities hold:

$M \models_s \Box \varphi$ iff $M \models_s \neg\Diamond\neg\varphi$ and $M \models_s \Diamond\varphi$ iff $M \models_s \neg\Box\neg\varphi$

$M \models_s \Diamond\varphi$ iff $M \models_s \mathsf{T}\,\mathsf{U}\,\varphi$

$M \models_s \boxminus\varphi$ iff $M \models_s \neg\diamondminus\neg\varphi$ and $M \models_s \diamondminus\varphi$ iff $M \models_s \neg\boxminus\neg\varphi$

Thus, $\Box$ and $\Diamond$ can be defined by means of U, and the past time operators are interchangeable. Indeed, it would be possible to defined a past time operator S, *since*, corresponding to the future time operator until, and define the other past time operators with it.

4.5.2 Branching time temporal logic

A branching time temporal logic can be used to consider the branching runs generated by a protocol specification. To illustrate its intention, we will first consider an example.

Example. Let's consider a fragment of the earlier example where Alice initially has both integers $i_1 = 342457621567678$ and $i_2 = 236523567732567$ in her possession, and Bob has nothing. To make the event sequences explicit, we add the environment Eve to the model. Eve stores in her local state the sequence of events, thereby acting as a bookkeeper for time. To make things slightly more realistic, we also allow Eve to delay indefinitely the delivery of messages, i.e. she does not necessarily immediately forward any messages she has got. We also allow her to duplicate messages; this makes things easier since we do not have to count how many times a message is received and delivered by Eve.

The new model with Alice, Bob and Eve is illustrated in Figure 9 on page 62. In the figure, Eve's state is not explicitly shown; it can be seen from the message sequences.

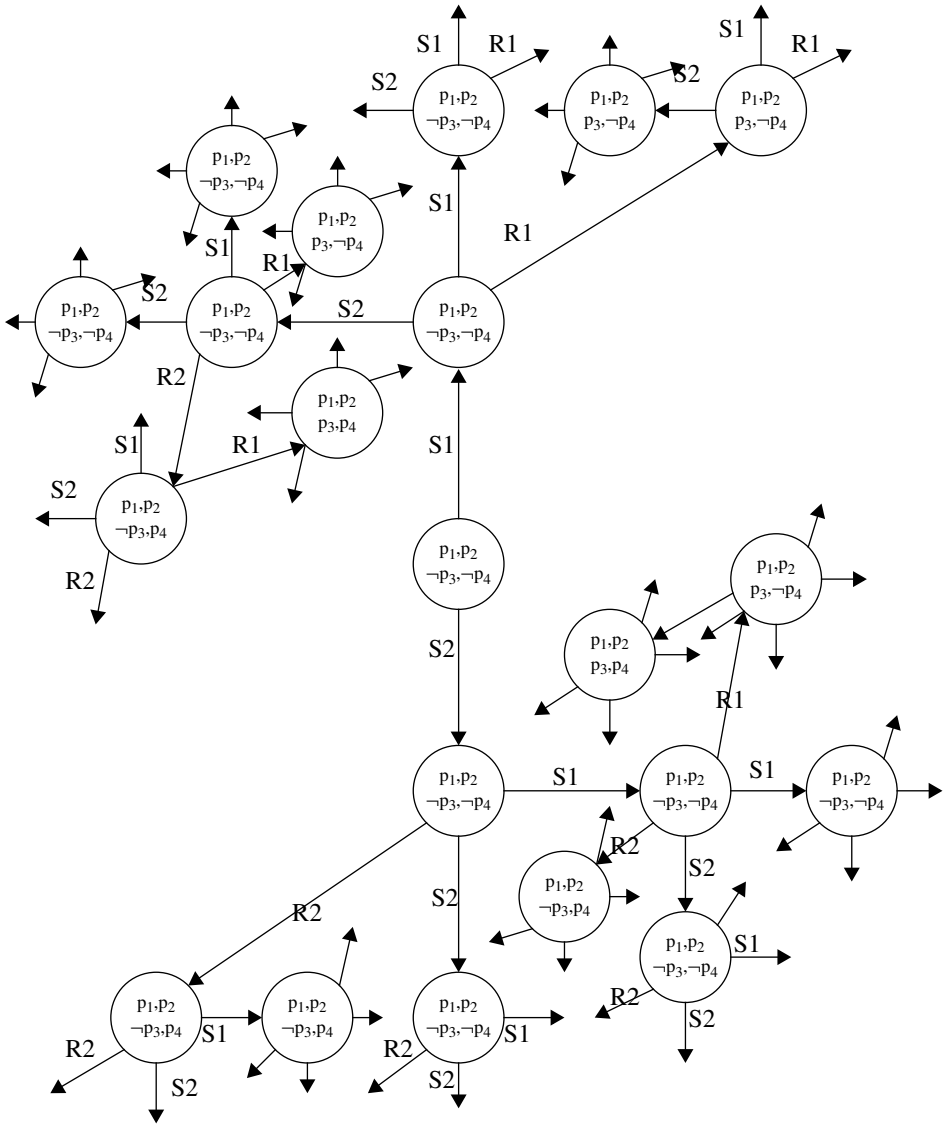


FIGURE 9. Event sequences in a system with Alice, Bob and Eve

Alice sending a message (to Eve) is denoted with S1 or S2. Bob receiving a message (from Eve) is denoted with R1 or R2. As it can be seen, the event sequences form a tree which branches very quickly. In the model, after receiving one copy of both messages, Eve effectively decides the next action. Because she is always able to receive either message and (after receiving a copy) send either message, there is typically four possible futures from a given state. However, there is also an infinite number of states where there is only three possible futures, i.e. the message sequences where Alice infinitely sends only one message or another.

Considering the model above, we can express the following statements:

- In the past, there has been a state where Bob hasn't known i_1, i.e. $\neg p_3$:

  $M \models \Diamond\neg p_3$

- In all future states, there is always some future state where Bob potentially will know i_1:

  $M \models \Box\Diamond p_3$

Syntax and semantics. We'll present a language CTL-X, based on Computational Tree Logic [27] without the next time operator. The presentation is partially based on [43] with some insight gained from [12].

To take the branching structure of a protocol model to the level of logic, it is necessary to distinguish between two different dimensions for the (future) time quantifiers: branching and necessitation. In the branching dimension, it is possible to consider only one branch, or all the possible branches. In the other dimension, a modality can denote that something is true at all states or only at *some* state. Thus, we get the following modalities:

Our notation	CTL notation	Description
$\Box\varphi$	$[\forall G]\varphi$	φ is true *everywhere* along all branches
$\Diamond\varphi$	$[\forall F]\varphi$	φ is true in some state along all branches, i.e. it is *inevitable*
$\neg\Diamond\neg\varphi$	$[\exists G]\varphi$	φ is true in all states along some branch
$\neg\Box\neg\varphi$	$[\exists F]\varphi$	φ is true in some state along some branch, i.e. it is *possible*

Handling of past time depends on whether the underlying model is history preserving or not. That is, if all joins in possible event paths in the model denote joins of truly parallel computations (instead of continuing from a choice), each path denotes a unique history, and past time can be considered linear [40]. We will adopt this view, and unwind process models to form history preserving models when necessary.

The future time operators can be defined in terms of two different until -operators, let's say $\mathcal{U}$ and $\mathcal{U}$, leading to a language

$$\varphi = p_i \mid \bot \mid \varphi_1 \rightarrow \varphi_2 \mid \varphi_1 \mathcal{U} \varphi_2 \mid \varphi_1 \mathcal{U} \varphi_2$$

where p_i denote the primitive propositions, $\bot$ is false constant as usual, arrow denotes implication, the other propositional connectives are defined in means of falsum and implication, modal sentences are formed using the until-operators, and other modal operators can be defined in terms of them. The modal semantics are defined as follows:

$M \models_s \varphi \mathcal{U} \psi$ iff for *all* branches, there is a state t such that $M \models_t \psi$ and $M \models_u \varphi$ for all intermediate states u, $sR^*u \wedge uR^*t$

$M \models_s \varphi \mathcal{U} \psi$ iff for *some* branch, there is a state t such that $M \models_t \psi$ and $M \models_u \varphi$ for all intermediate states u, $sR^*u \wedge uR^*t$

4.6 Combining knowledge and time

It's time to return back to the notions of knowledge and belief, and consider models where we can express both time and knowledge or belief. To keep things simpler, here we will only consider models where the assumptions of the protocol parties are assumed to hold, i.e. models with time and knowledge instead of time and belief.

In a distributed system, there is an intrinsic relationship between knowledge, time and communication [92]. From a given agent's point of view, the only possible way of gaining (or losing) knowledge is via communication actions and internal actions

generating fresh nonces. The communication actions, i.e. the sending and receiving of messages, as well as internal actions can be considered atomic, and therefore sequential in time. Thus, given a single agent, its history can be considered to consist of periods of inactivity where time flows by, interleaved with actions that change the knowledge state of the agent.

Thus, in a model where time and knowledge are combined, each agent has an initial knowledge state where it considers a number of worlds possible. This knowledge state, and therefore the set of possible worlds, is changed only (and always) when the agent sends or receives a message, or performs some internal action changing its internal state. As we always consider communication to happen synchronously between an agent and the environment, this means that each communication action changes the knowledge set of the communicating agent and the environment, but *not* the state of any *other* agent. The internal actions, on the other hand, only change the knowledge state of the agent itself.

Surprisingly, there seems to be little literature about models capable of representing both knowledge and time with modal operators. Thus, being unable to refer to any established theory, we'll suffice in presenting a simple example illustrating the intended direction of ideas.

A protocol example. Let's consider a simple example where Alice sends Bob one of a number of possible messages, and Bob sends an acknowledgement once he has got the message. Eve, the environment, is able to destroy and duplicate any messages, but not able to create any new messages by herself. Furthermore, we'll first consider a situation without any resending: if Eve decides to delete the message from Alice to Bob or the acknowledgement from Bob to Alice, the system deadlocks.

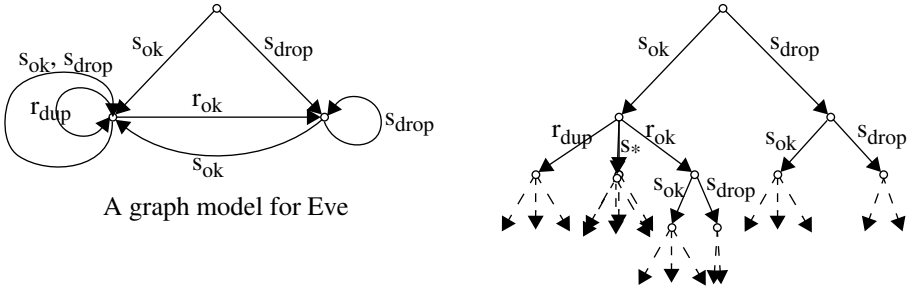
Without loss of generality, to ease illustrations, let's consider a situation where Alice has a choice between only two possible messages, m_1 and m_2. According to her protocol specification, she first decides whether to send m_1 or m_2 — the decision event is denoted by d_1 and d_2 — then sends the message according to her decisions — s_1 or s_2 — and finally accepts an acknowledgement, denoted as r_{ack}. Bob, on his behalf, is initially ready to receive either message, r_1 or r_2, after which he sends an acknowledgement s_{ack}. The models for Alice and Bob are illustrated in Figure 10.

The model for Eve is a little bit more complicated this time. It basically consists of three distinct channels operating in parallel. Each channel is able to receive a sent message, immediately deciding to delete it or to carry it and offer it for delivery, and in the latter case also able to deliver the message once it has been received. Two of the channels are needed to handle the two different messages, while the third is capable of handling the acknowledgement.

More specifically, for each channel, s_{drop} indicates the action where the channel receives a sent message, but immediately deletes it. The action s_{ok} is its counterpart after which the message sent is available for delivery. The label r_{dup} denotes the

A model for Alice A model for Bob

FIGURE 10. Models for Alice and Bob



A graph model for Eve

An event tree model of Eve

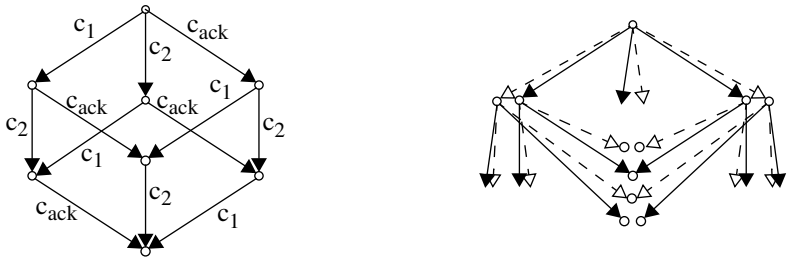
FIGURE 11. A model for a channel, and the same model unwound to a tree

action where a message is delivered and duplicated, so that it can be sent again, and r_{ok} is used for the situation where the delivered message is not duplicated. A possible model for a channel is given in Figure 11.

The three channels operating in parallel can be illustrated as a three dimensional cube where each dimension denotes the possible actions of one channel. Understanding each channel as a one action channel c_x, the combined model can be shown as a simple cube; the notion of other actions bring into life a number of "shadow cubes" denoting the states where a message was dropped. The simple cube and "shadow cube" pictures are shown in Figure 12, where the actions beginning and ending in the same state are not shown.

Combining the models of Alice, Eve and Bob by synchronizing in the appropriate actions leads to a event tree illustrated in Figure 13 on page 66. The temporal relationships are clearly visible in the figure. Each arrow represents a immediate successor relation in the temporal frame, and the reflexive transitive closure of the arrows fully define a model for a discrete branching time temporal logic. The past time operators, however, can be considered linear since no state has more than one immediate ancestor.

Because Alice's state, and thereby Alice's knowledge, can only be changed by an internal action of Alice or by a communication action Alice is taking part in, the only transitions changing Alice's knowledge are d_1, s_1, d_2, s_2 and r_{ack}. Similarly, the actions r_1, r_2 and s_{ack} change the knowledge state of Bob. Now, using this information we can impose the equality relationships denoting the worlds Alice and Bob consider possible at each state. This is represented in Figure 14, where the solid lines display the boundaries of Alice knowledge sets, and dashed lines the boundaries of Bob's sets.



Three channels of Eve, each as a solid arrow

Three channels of Eve, a partial figure. Solid arrows indicate succeeded sendings, dashed unsuccessful ones

FIGURE 12. A complete model for Eve consisting of three parallel channels

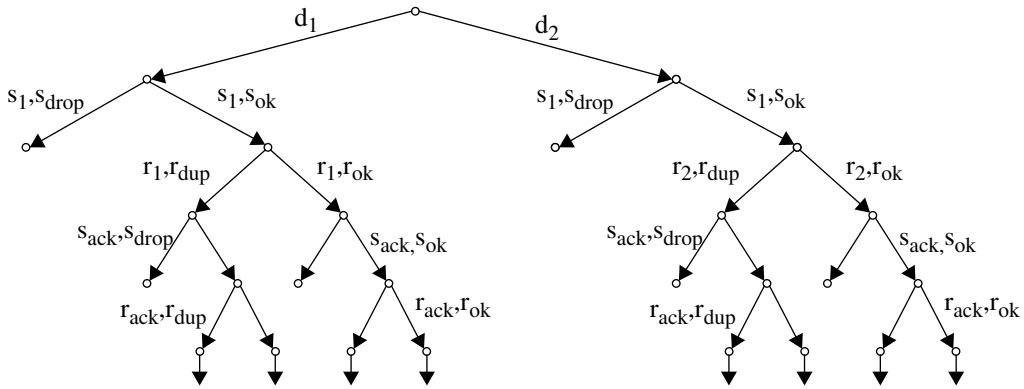


FIGURE 13. A combined protocol model

Based on this model, it is easy to give examples of formulae containing both knowledge and temporal operators. To do that, let's define a couple of propositions:

- s_1 Alice has sent m_1
- s_2 Alice has sent m_2
- r_1 Bob has received m_1
- r_2 Bob has received m_2

With these, we can express, for example, the following facts that hold at the initial state of the model:

- It is possible (i.e. there is a future along some path) that Alice knows that Bob has received the message she has sent:

  $\neg\Box\neg(\text{Alice knows } ((s_1 \wedge r_1) \vee (s_2 \wedge r_2)))$

  The reading goes something like this: it is not true that never along any branches Alice knows that Bob has received a message and she has sent the very same message, i.e. there is at least one branch where eventually this is true.
- It is possible that Bob has received the message Alice has sent, but Alice does not know if Bob has received anything:

  $\neg\Box\neg(((s_1 \wedge r_1) \vee (s_2 \wedge r_2)) \wedge \neg(\text{Alice knows } (r_1 \vee r_2)))$

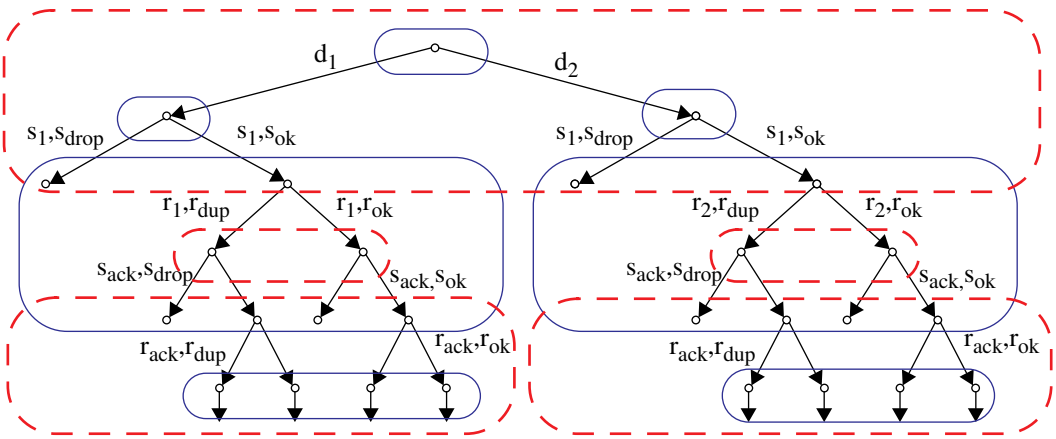


FIGURE 14. Alice's and Bob's knowledge sets

- Along every possible path, Alice sends either m_1 or m_2:

  $\Box(s_1 \vee s_2)$

- Along every possible path, Alice knows that if she knows that Bob has received a certain message, he knows that she has indeed sent it:

  $\Box(\text{Alice knows } (\text{Alice knows } r_1 \rightarrow \text{Bob knows } s_1) \wedge (\text{Alice knows } r_2 \rightarrow \text{Bob knows } s_2))$

- Bob does not ever know if Alice knows that he has received a message:

  $\Box\neg(\text{Bob knows } (\text{Alice knows } (r_1 \vee r_2)))$

In the examples above we have applied the temporal operators on formulae built from atomic propositions using propositional logic and knowledge operators. Considering the situation the other way round, i.e. applying knowledge operators on formulae continuing temporal operators, we encounter a slight philosophical problem. The model above, taking it literally, means that we, as an external observer, have a complete knowledge about all possible futures of every possible state. Now, how should we consider the situation to be from Alice's and Bob's point of view? Keeping our extensional point of view towards knowledge, it seems to be eligible to assume that Alice and Bob are also reconsidered to "know" all the possible futures. Thus, their only inability is to distinguish the current actual state from all the states that were possible based on their information.

Using this interpretation, we can express more statements:

- Alice knows that Bob will never know (i.e. along all paths Bob does not know) if she knows that he has received a message:

  $\text{Alice knows } (\Box\neg(\text{Bob knows } (\text{Alice knows } (r_1 \vee r_2))))$

- Alice knows that it is possible that she will know that Bob has received a message:

  $\text{Alice knows } (\neg\Box\neg(\text{Alice knows } (r_1 \vee r_2)))$

- Bob knows that it is inevitable that Alice knows that he knows that she has sent m_1

  $\text{Bob knows } \Diamond(\text{Alice knows } (\text{Bob knows } s_1))$

- Alice knows that here is a state where Alice has sent m_1, but Bob will never know that:

  $\text{Alice knows } \neg\Box\neg(s_1 \wedge \Box\neg(\text{Bob knows } s_1))$

A variation. Let's consider a slightly more complicated variation of the protocol where Alice tries to send the message two times, or until she receives an acknowledgement, and where Bob sends an acknowledgement two times in reply to a received message. The capabilities of Eve are not changed. The modified models for Bob and Alice are given in Figure 15. The combined model becomes rather complicated, as can be seen from Figure 16. It displays some possible execution paths and some incomplete paths of the combined model without action names.

In Figure 16, the grey (red) lines denote the second sending of m_1 on the behalf of Alice, and (the blue lines) behaviour caused by the handling of this message. This may happen immediately after the first sending, or at any later moment when Alice has not yet received an acknowledgement. To denote that this action is fully parallel with the behaviour that is happening in the meantime, we have drawn it from each

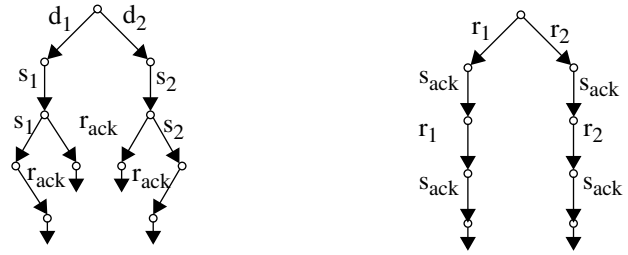


FIGURE 15. Modified models for Alice and Bob

possible state where it can be sent into a corresponding state where it has indeed been sent, but not acted on (yet). Due to the restrictions of the Eve model, actually some of the states where Eve has received both the first and the second sending of m_1 are indistinguishable, if a straight process algebraic parallel composition of the models are taken (cf. Chapter 5, "Model checking and Process Algebra"). However, we have represented even these states, or "phantom" states, in the figure in order to make the true parallelism (non-interleaving behaviour) more apparent. Furthermore, the three points with the light grey (yellow) background denote points where there are other possible futures in addition to those presented.

The complexity of the example clearly indicates that is would be very laborious and error-prone to construct these kinds of models for real life protocols without the help of some kind of formal calculus and maybe also computer-assisted tools.

4.7 Summary

We have introduced the concepts of a modal and multimodal logics, and given semantics for them. The semantics are based on the possible-world interpretation of Kripke-structures and corresponding frames. We briefly discussed some of the better

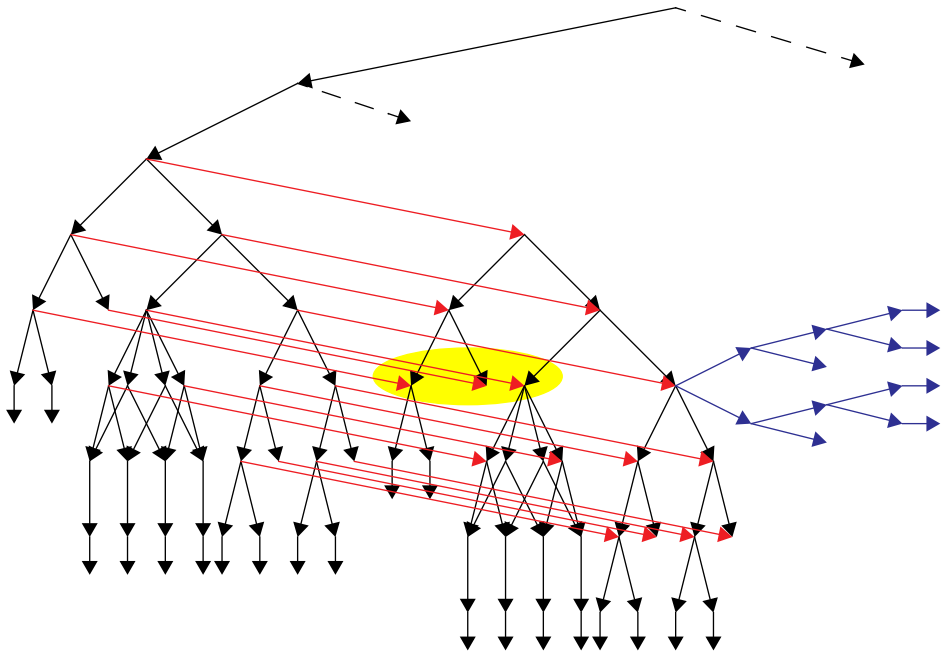


FIGURE 16. Some complete and some partial paths of a model with resending

known axioms of modal logic, including K, T, D, B, 4, 5, L and Dum. The intuitive semantic meaning of these axioms was discussed from both the epistemic and temporal point of view. We also briefly considered the concepts of soundness, completeness and determinedness of a logic; however, most issues pertaining to them are beyond the scope of this study.

Given the basic theory and concepts, we described a logic of knowledge in terms of examples. We also briefly considered two possible interpretations for beliefs. A linear time temporal logic without a nexttime operator was considered along with an example. A similar branching time temporal logic, CTL-X, was discussed.

Finally, in the end of the chapter we considered how one could combine both temporal and epistemic modal operators within a single logic. We presented a simple example using our usual protocol parlance, and noticed how the model becomes extremely large pretty soon as the complexity of the protocol is increased.

CHAPTER 5

Model checking and Process Algebra

Now we turn our attention to a number of ways of modelling concurrent systems, and especially introduce some process algebraic formalisms including CCS, CSP and ACP. The reason behind this presentation is to familiarize the reader with some concepts and tools used in analysing communication protocols in general. We will present a number of examples in order to illustrate the usage. The purpose of this presentation is to familiarize the reader with the concurrency perspective later apparent in Chapter 7, "Modelling of cryptographic protocols". Furthermore, a CSP based formalism has been lately used by Kevin Lowe and others at Oxford to model cryptographic protocols with very promising results [67, 68]. However, those are beyond the scope of this study.

All communication protocols can be represented as concurrent systems. The protocol parties, and in the case of cryptographic protocols the intruder(s) as well, act in parallel, starting and stopping actions without any *direct* causal relationships with the others. Experience has shown that understanding concurrent behaviour without formal tools is almost impossible. In a system where many actions are being performed in parallel, human intuition is typically not good enough to grasp all the possible behaviours of the system. A mathematical model tries to alleviate this problem.

There are many approaches to modelling concurrency, including Petri nets, automata, process algebraic formalisms, and event structures, to mention but a few. Different formalisms have different ways to express issues and different analysis tools. Our goal is to cover enough of concurrency theory to allow the reader to follow and understand the model checking approaches of protocol analysis used later in this study. We have chosen to present the theory from process algebraic view. The reason for this is basically twofold. First, the abstract algebras developed allow an easy construction of more complex models by combining simple ones. Second, a state based algebra seems to be easy to understand and a natural way of thinking to many computer professionals.

The rest of this chapter is organized as follows. First we will consider terminology and generic properties of state based models of concurrency. After that, we will introduce process graphs and labelled transition systems, two isomorphic semantic formalisms that can be used to illustrate system models. Next a number of theoretic process algebraic approaches are briefly considered, namely ACP, CCS and CSP. The

toy cryptographic protocol example of Section 2.5 is presented in ACP and CSP. This allows a comparison from a practical point of view.

Having the basic concepts in hand, it is possible to introduce some semantic equivalencies. This relatively large area is touched only lightly, in order to underline the difficulty of deciding which kind of semantics is appropriate in which situations. And finally, having discussed both notation and semantics, we will briefly consider the issue of model checking. In particular, a number of generic approaches, tools, and difficulties are discussed.

5.1 Introduction to models of concurrency

As we have already seen, all models of concurrency have a concept of state. However, this is about the only common definition between the various formalisms. They differ in they approach to events, concurrency of events, refinement of events, ways of expressing communication, composition of models, etc. In this section, the basic concepts of state, actions, events, communication, deadlocks, livelocks, abstraction, hiding, and traces are considered. Composition and other algebraic properties are left to the next section.

5.1.1 States, actions and events

A (global) state is a description of the system under consideration at a particular moment. In a real world system there are typically moments without any visible behaviour and moments of change. It is customary to consider each period of externally stable looking moments as a state and the periods of change as transitions between the states. Of course, it is possible to have a finer look and view the states as consisting of several internally distinguishable states and internal actions between them, and to consider periods of change as consisting of several, maybe even continuous number, of states each representing a different amount of computation having been performed. However, to be practical it is necessary to try to find out a suitable level of abstraction. This is often one of the hardest problems in modelling of systems.

We will adopt a view where states are seen as periods of time without any visible action, and transitions as instantaneous changes between states. If wished, transitions can be seen as the moments where some action is completed, the action itself having been performed during the state ending with the transition, or, if the action can be considered started already earlier, during some previous states. Thus, in our view, the progress of some (started) action is considered invisible to an external observer; only the completion of the action can be noticed.

It is customary to call models where transitions can be repeated during a protocol run *action based models*, and the transitions are *actions* in this case. On the other hand, if a transition represents a phenomenon that can only happen once (or not at all) during a protocol run, the transitions are called *events* and the models *event based models*. It is possible to transform an action based model into an event based model by associating with each possible occurrence of an action a separate event. Of course, and event based model can be transformed into an action based model by labelling the events with action names.

In an event based model states can be represented as collections of events that have happened. In such a system, when an event has happened it is impossible to return to a state where the event had not happened. Thus, there are no loops, i.e. when a state is left, there is no way to return to it. A disadvantage of this is that the models tend to

become very large, often infinite, whereas a similar system could often be represented with a small action based model. However, because of their structure, the event based models keep track of history, i.e. occurrence of actions, and can be provided with a schedule-automaton duality [50].

Many models can be given both interleaving and non-interleaving semantics. In a non-interleaving semantics two actions can be seen as happening truly in parallel, whereas in an interleaving semantics actions are seen as indivisible, and parallel execution of two actions is seen as executing the actions in arbitrary order. Thus, an interleaving semantics has a law like $a \parallel b = ab \sqcup ba$, expressing that a in parallel with b is equal to a executed before b or b executed before a. The basic benefit of interleaving models is that they typically lead to a more tractable algebra. On the other hand, expressing action refinement, i.e. replacing of one action with a subsystem, is typically easier in a non-interleaving model.

We will mostly use action based interleaving models, occasionally rolling out the model into a corresponding event based model. We will also sometimes consider non-interleaving semantics for a model; however, this is always explicitly indicated. Unless otherwise noted, the user should expect interleaving action based semantics.

5.1.2 Communication

For a protocol specifier, it is beneficial if the system can be represented as subsystems communicating with each other. This requires a possibility to explicitly represent communication between the subsystems. Various models have taken different approaches to this. In Petri nets, communication is represented with transitions between subsystems. These transitions are indistinguishable from transitions representing actions internal to the subsystems. Because Petri net transitions are basically unnamed, it is somewhat hard to represent and combine half-transitions expressing the individual acts of communication.

In process algebraic approaches, communication is constructed by introducing an implicit or explicit communication function which expresses which of the actions of a subsystem do communicate with actions of other subsystems. In CSP, actions carrying the same label are considered always to communicate, and — somewhat similarly — CCS communication is always the case between actions with the same dashed and undashed name (or name and co-name). On the other hand, ACP requires an explicit introduction of a communication function which basically transforms pairs of actions into new actions representing the communication. We will return to this below in section 5.3, “Algebraic approaches”.

5.1.3 Deadlocks and divergencies

Two properties of concurrent systems related to protocol error conditions are deadlocks and divergencies. A deadlock is a situation where the system inappropriately stops executing actions. This is in contrast to successful termination, where the end of execution is seen as a proper phenomenon. Deadlocks can be accidentally introduced by combining systems with different resource reservation policies or communication assumptions. Within a model, a deadlock can be represented as a state with no possible transitions. In models with successful termination, the set of final states must be made explicit in order to distinguish deadlocks from other terminal states.

A divergency, or livelock, is a situation where the system continuously executes (invisible) actions without being able to proceed. A divergency is represented as a loop in an action based model, and it translates to an infinite sequence of events in an event based model. A livelock can occur, for example, when two processes continuously send each other error messages that trigger new error messages.

It is possible to distinguish two types of divergencies. The first type, or true livelocks, are situations where there the system does *not* have a possibility of proceeding normally, i.e. whatever nondeterministic choices are made, the system keeps on the track of executing actions belonging to the loop. The second type of divergencies, on the other hand, refer to situations where it is *possible* that the system will continue to stay in the loop indefinitely, but where there is also a possibility of exiting the loop.

5.1.4 Abstraction and hiding

Abstraction, or hiding, in the language of a process algebras, refer to operations that abstract away, or hide, some information about a process model without changing its actual behaviour. Typically this means that all occurrences of one or more actions are replaced with a special externally invisible action τ. Some of these may then later be removed, depending on the exact semantics of the underlying model. It is important to notice that abstraction itself does not introduce nor delete deadlocks or divergencies. It only narrows our window to the process, i.e. allows us to observe less actions than before.

Abstraction is usually needed to figure out whether a more specific process model, often called an implementation, is equivalent to a coarser process model, a specification. Those actions not present in the specification are abstracted away from the implementation model, and the abstracted model is compared to the specification. Thus, in a way, abstraction can be seen to introduce a coarser equivalence class based on the used semantic equivalence, whatever it is.

5.1.5 Traces and equivalencies

As already discussed in section 2.4, “Modelling of communication protocols”, on page 21, a trace (or run) is a sequence of actions (or events) representing the externally visible behaviour of a system. Traces also form the most coarse equivalence semantics of systems. Under trace semantics, two systems are considered equivalent iff their sets of traces are equal. If the systems are considered in isolation, the trace semantics is often good enough. However, if two equivalent systems are placed into a communications environment, it is well possible that the systems will exhibit different behaviour. In particular, it is possible that one system will occasionally deadlock in a situation where the other system is always able to proceed. This indicates a need for finer set of equivalence classes, and indeed a number of semantics to express the differences have been developed. Some of these will be considered in Section 5.4, “Semantics”. We will see how some of them preserve information about deadlocks and/or divergencies.

5.2 Process graphs and Labelled Transition Systems

It is usually easier to understand the behaviour of a simple system from a pictorial notation. To make this possible, we introduce two graphical presentations, namely process graphs and labelled transition systems. These will be later used interchangeably to illustrate protocol models. It should be noted that it is possible to give other kinds of semantics to the algebraic approaches as well, and indeed e.g. [8] introduces several alternative semantics for ACP.

5.2.1 Process graphs

A process graph is a rooted directed multi-graph. Formally, a graph consist of a set of nodes or states S, an initial node (the root) r, a set of edges or transitions T, and two

functions $s : T \to S$ and $t : T \to S$ defining the source and target of all edges. A process graph is an graph with each edge has a label from the set of actions A in such a way that all edges between the same nodes carry a different label, and with a subset F of S defining the set of states which are considered terminal states. In formal terms, this adds a labelling function $\lambda : T \to A$ and the characteristic function of F, $\surd : S \to \{0, 1\}$. This leads to a definition of a process graph as a tuple $(S, r, T, s, t, A, \lambda, \surd)$. We will consider process graphs equal modulo isomorphism of root and termination preserving bijections of $S \to S'$ with other structure reflecting the state space changes appropriately.

A process graph can be represented as a picture with circles for the nodes and labelled arrows as the edges, each connecting the source of an edge to the target of it. A finite graph is a graph with finitely many nodes and edges. A process tree is an acyclic process graph where each node is the target of at most one edge. In this study, we will only consider finite process graphs as action based models, and process trees as action labelled event based models.

5.2.2 Examples

The most simple process graph has no nodes and no edges. The trivial graph, denoted 0, has one node and no edges. A one-step action has two nodes and one edge connecting them, the latter being an terminal node. A deadlock is represented by a trivial graph where the node is not terminal. A simple livelock is a graph with an edge forming a loop with the one node as the source and target of it. Some of these along with a couple of other examples are given in pictorial form in Figure 17 below. The initial node is represented as an incoming unlabelled arrow, and the terminal nodes with unnamed outgoing arrow.

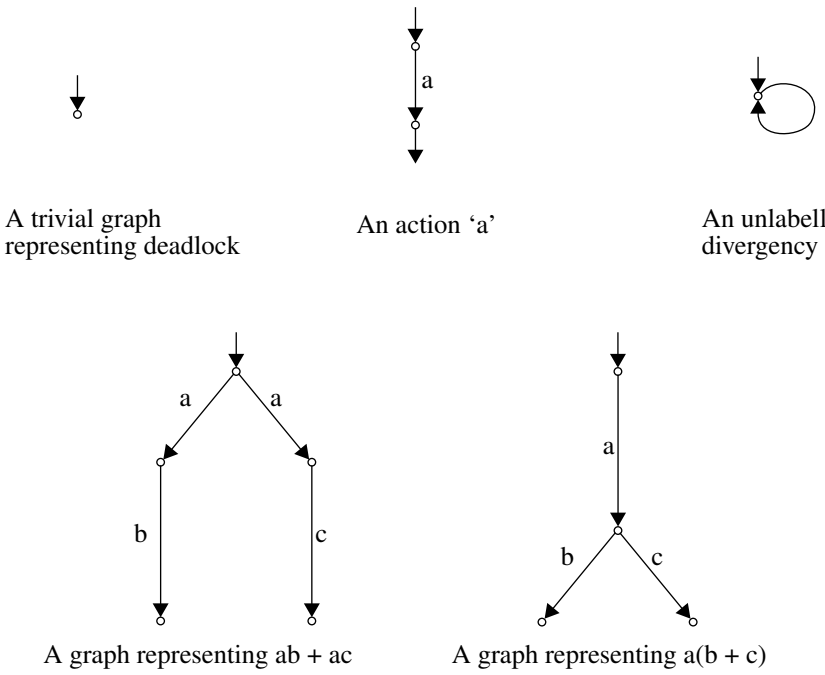


FIGURE 17. Examples of process graphs

5.2.3 Labelled transition systems

A labelled transitions system, or LTS for short, is a tuple

$$\left(S, T, \left\{ \xrightarrow{t} : t \in T \right\}, s_0\right)$$

which consists of a set of states S, set of transition labels T (the actions), and a set of transitions relations $\xrightarrow{t} \subseteq S \times S$, one for each $t \in T$, and an initial state (or root) s_0. In other words, a labelled transition system can be considered to be a set of states with a number of transition relations imposed on it. Each relation expresses a set of possible transitions in terms of their initial state, final state and associated action label.

It can be shown that there is a natural isomorphism from process graphs to labelled transition systems modulo terminal nodes. That is, given a LTS there is a unique corresponding set of process graphs where the set of states is the same, the set of graph actions A is equal to the set of LTS transition labels T, and the set of edges as well as functions s, t and λ are uniquely determined by the transitions relations, and the root r of the process graph is equal to the initial state s_0 of the LTS. Correspondingly, given a process graph there is a unique LTS with the same set of states, a set of transition labels equal to the set of actions, initial state equal to the root, and the set of transition relations uniquely determined by the set of edges and the functions s, t and λ.

Due to this isomorphism, from now on we will freely use the terms process graph and labelled transition system interchangeably. Moreover, it is usually natural to consider the terminal nodes of an LTS to represent terminal states. This makes the distinction even more indistinguishable. However, if we have an occasion where the different structure of these two semantics are meaningful, this is explicitly noted.

5.3 Algebraic approaches

The term process algebra was introduced by J. A. Bergstra and J. W. Klop of the Centre of Mathematics and Computer Science (CWI) at Amsterdam in the beginning of 1980's. It is closely related to Robin Milner's CCS and to the CSP work of Tony Hoare. In this study, we denote all these and other similar formalisms with the names algebraic approach or process algebra. From this view, their basic characteristic is an abstract algebra, or process calculus, which allows the construction of more complex processes from simple ones.

It is possible to construct several different models that can act as sound and/or complete models for the different algebras. It is also possible to give different equivalence semantics, and preorders based on the equivalencies. In this section, we will describe the algebras both from an usage point of view and using the process graphs as a model for the algebra. Equivalence and refinement semantics are covered later. A more formal introduction to process algebras can be found e.g. in [8].

5.3.1 ACP — Asynchronous Communicating Processes

The process algebra ACP was introduced by Bergstra and Klop in [13]. In this section and elsewhere in this study we follow the presentation given in [8].

TABLE 8. ACP axioms [8, page 94]

$x + y = y + x$		A1
$(x + y) + z = x + (y + z)$		A2
$x + x = x$		A3
$(x + y)z = xz + yz$		A4
$(xy)z = x(yz)$		A5
$x + \delta = x$		A6
$\delta x = x$		A7
$a \mid b = \gamma(a, b)$	if $\gamma(a, b)$ is defined	CF1
$a \mid b = \delta$	otherwise	CF2
$x \mid\mid y = x \between y + y \between x + x \mid y$		CM1
$a \between x = ax$		CM2
$(ax) \between y = a(x \mid\mid y)$		CM3
$(x + y) \between z = x \between z + y \between z$		CM4
$(ax) \mid b = (a \mid b)x$		CM5
$a \mid (bx) = (a \mid b)x$		CM6
$(ax) \mid (by) = (a \mid b)(x \mid\mid y)$		CM7
$(x + y) \mid z = x \mid z + y \mid z$		CM8
$x \mid (y + z) = x \mid y + x \mid z$		CM9
$\partial_H(a) = a$	if $a \notin H$	D1
$\partial_H(a) = \delta$	if $a \in H$	D2
$\delta_H(x + y) = \delta_H(x) + \delta_H(y)$		D3
$\delta_H(xy) = \delta_H(x)\delta_H(y)$		D4

The ACP system can be specified using an equational specification. It consists of

- a set of constants A (the actions),
- a partial communication function $\gamma : A \times A \to A$ which is commutative and associative,
- a constant δ denoting the deadlock process,
- binary operators for choice (+), sequential composition (.) and three different kinds of parallel composition ($\mid\mid$, $\between$ and $\mid$) , and
- unary operator ∂ for encapsulation.

The axioms for ACP are given in Table 8. An uninitiated reader is instructed to proceed to Figure 18 on page 86 and study the pictorial representations of some of the axioms before considering the explanations next. In the table, the symbols x , y and z denote arbitrary processes, while a , b and c are actions, i.e. members of the set A .

Basic process algebra. A1 - A7 define the basic properties of choice, sequential composition and deadlock. In particular, A1 - A3 define that choice is commutative, associative and idempotent. In other terms, A1 expresses that whenever there is a choice between two possible behaviours, it does not matter in which order the possibilities are presented. Similarly, A2 effectively tells us that if there is a choice between three possibilities, this can be regarded as first choosing between any pair of the two, and then between the result and the third. Together the first three define that whenever there is a choice of any number of possibilities, the possibilities can be considered as equal alternatives.

Axiom A4 says that if we have to make a choice between the alternative behaviours x and y, and will continue with a similar behaviour z after them in any case, it does not matter if we consider the behaviour z after x as distinct to behaviour z after y or not. A5 expresses the associativity of sequential composition, which is self-evident.

It is worth noting that there is no axiom $x(y + z) = xy + xz$. This makes a distinction between early and late branching, apparent in the lower half of Figure 17 on page 75. This point will be discussed later in the section describing semantics.

Deadlock behaviour is defined by the axioms A6 and A7, effectively expressing that whenever there is a choice between something and deadlock, something will be chosen, and that there is no possible behaviours after a deadlock. In particular, it must be noted that $a\delta \neq a$, since it is possible to continue after a, but it is not possible to continue after $a\delta$. This distinction is typically not made in process algebras without an explicit notion of deadlock, e.g. CCS.

Communication function. The axioms CF1 and CF2 say that the communication merge $|$ is actually an extension of the communication function γ. Along with CM5-CM9 it becomes clear that $x \mid y$ will only proceed if some combination of the possible initial actions (cf. CM8-CM9) do communicate, i.e. $\gamma(a, b)$ is defined for some pair (a, b) where a is an initial action of x and b is an initial action of y. If there are several communicating alternatives, any of them can be chosen.

Encapsulation. The encapsulation axioms D1-D4 are used to make some behaviours impossible. Encapsulation is typically used to restrict merge to a particular type of parallel composition (see the example on the next page), but they may be used on their own as well. By definition, a process x restricted by a set of actions $H \subseteq A$ is a process where all occurrences of actions $a \in H$ are replaced by the deadlock. That is, the restriction $\partial_H(x)$ behaves similar to x other than whenever x is able to perform some action belonging to the set H, the restricted process does not have this option, and deadlocks if all possible behaviours of x at the given situation start with some action belonging to H. This may sound confusing, but the practical usage should become clear from the examples.

Parallel composition. The merge axioms CM1-CM4 define the merge operator $\|$ in terms of the so called left-merge operator $\lfloor\!\lfloor$ and the communication merge $|$. A merge of two processes x and y, $x \| y$, is defined by CM1 as a process is capable to choose between three possibilities:

1. It is possible first to perform any first action of x, and to continue with the merge of the rest of x and y. This is indicated by the first right hand term of CM1, $x \lfloor\!\lfloor y$, along with CM2-CM4.
2. Symmetrically, it is possible to perform any first action of y, continued by the merge of the rest of y and x.

3. The third possibility is to perform a joint first action of x and y, if the communication function defines such an action for any pair of initial actions of x and y. This corresponds to the third term of right hand side of CM1, $x \mid y$, which is symmetric.

This, along with the encapsulation axioms D1-D4 makes it possible to define various kinds of parallel execution. First it must be noticed that if the communication function γ is defined for no action pairs of the merged processes x and y, the merge $x \parallel y$ is actually an interleaving parallel execution of x and y. However, if there is a possibility of communication, the truly parallel execution can be simulated by restricting away the results of the communication function from the merge. For example, let $A = \{a, b, c\}$, $\gamma(a, b) = c$, γ not defined for any other pair of actions, and $H = \{c\}$. Now,

$$\begin{aligned}
\partial_H((a + aa) \parallel b) &= \partial_H((a + aa) \between b + b \between (a + aa) + (a + aa) \mid b) \\
&= \partial_H(a \between b + aa \between b + b(a + aa) + a \mid b + aa \mid b) \\
&= \partial_H(ab + a(a \parallel b) + b(a + aa) + c + (a \mid b)a) \\
&= \partial_H(ab + a(a \between b + b \between a + a \mid b) + b(a + aa) + c + ca) \\
&= \partial_H(ab + a(ab + ba + c) + b(a + aa) + c + ca) \\
&= ab + a(ab + ba + \delta) + b(a + aa) + \delta + \delta a \\
&= ab + a(ab + ba) + b(a + aa)
\end{aligned}$$

The final formula is clearly the parallel execution. There are initially three alternatives:

1. to perform the alone a of the left process, in which case the only possible future is the execution of b,
2. to perform the first a of the aa construct, in which case in the next step either the second a can be performed, followed by b, or vice versa, and
3. to perform first b, naturally continued by the behaviour of the left process.

TABLE 9. Hiding axioms for ACP^τ [8, page 123]

$x\tau = x$	B1
$x(\tau(y + z) + y) = x(y + z)$	B2
$\tau_I(a) = a$ if $a \notin I$	TI1
$\tau_I(a) = \tau$ if $a \in I$	TI2
$\tau_I(x + y) = \tau_I(x) + \tau_I(y)$	TI3
$\tau_I(xy) = \tau_I(x)\tau_I(y)$	TI4

Abstraction. The basic ACP does not contain abstraction axioms. The system ACP^τ is equal to ACP augmented with the abstraction axioms of Table 9 on page 79. This basically adds the abstraction operator τ_I along with the rules B1-B2 that allow the silent operation τ to be sometimes removed. These latter rules define a semantics for the silent moves; this point will be considered in section 5.4.5, “Branching bisimulation”, on page 89.

It can be shown that all ACP^{τ} expressions can be rewritten into an equivalent expressions of basic process algebra consisting only of actions in A, deadlocks δ, and silent moves τ combined with the choice and sequential composition. In other words, it is possible to construct a sequential process specification equal to any ACP^{τ} formula[1].

Example. Let's reconsider the ACP specification of Alice's behaviour of the example protocol of Chapter 2 given in Table 5 on page 29. There the behaviour of Alice was given as

$$ALICE = \sum_{m \in M} INIT_m$$

where each alternative expression $INIT_m$ denotes a different initial state corresponding to a different message to be sent m. The set M is the set of possible messages, which in the toy example was $\{0, 1, \ldots, 9\}$.

The set of run alternatives is further expanded by each initial action of Alice. In each initial state, Alice generates a nonce. From the ACP point of view, each possible outcome of the nonce generation is considered as a distinct action. If we denote the action of generating a nonce $n \in N$, where the set N is the set of possible nonces, as generate_n, we can express each initial state $INIT_m$ with the formula

$$INIT_m = \sum_{n \in N} (\text{generate}_n \cdot GENERATED_{m,n})$$

That is, in each initial state there is a choice of possible future behaviours. Each choice is determined by a different nonce generation action, and continued by a next state uniquely determined by the initial state and the generated nonce. Considering the states where Alice has both the message m and a nonce n available, the protocol specification inimitably determines the message to be sent.

We can denote a message Alice can send in a particular state as $\text{send}_{n,e,h}$ where $n \in N$ is a nonce (of the possible nonces), $e \in E$ is an encrypted message belonging to the set E of possible encryptions, and $h \in H$ is a hash value, one of the values of the set H. Now, the message Alice is determined to send is $\text{send}_{n,\text{enc}(m),h(n,m)}$, which is uniquely determined by n, m. Thus, the possible behaviour of Alice in the state under discussion can be defined as

$$GENERATED_{m,n} = \text{send}_{n,\text{enc}(m),\text{h}(N,m)} \cdot SENT_{m,n}$$

where $SENT_{m,n}$ is the corresponding state of Alice where she has sent the message.

Understanding the rest of the Table 5 on page 29 in this light, we can combine the states and give Alice' behaviour with a single ACP formula:

$$ALICE = \sum_{m \in M} \left(\sum_{n \in N} (\text{generate}_n \cdot \text{send}_{n,\text{enc}(m),\text{h}(n,m)} \cdot \text{receive}_{\text{h}(n,m+1)}) \right)$$

Similarly, we can give an ACP formula for Bob:

1. A reader familiar with ACP and other process algebras should note that we ignore the complications introduced by recursive formulae. Even though we allow the use of process variables in our specifications, it is seldom necessary to have real recursion when specifying cryptographic protocols. Effectively, this approach simplifies the presentation from both practical and theoretical point of view.

   However, later on, in section "Failures-divergencies semantics" on page 90, we will briefly discuss recursive formulae, divergencies, and how to handle them.

$$BOB = \sum_{m \in M,\, n \in N} \text{receive}_{n,\, \text{enc}(m),\, \text{h}(n,\, m)} \cdot \text{send}_{\text{h}(n,\, m+1)}$$

(Note that each generate, send and receive action in the previous specifications are distinct!).

Ignoring the possibility of an intruder, and to verify that the protocol indeed works if there is no intruders, we can simply define a communication function combining the corresponding send and receive actions:

$$\gamma(\text{send}_{n,\, e,\, h}, \text{receive}_{n,\, e,\, h}) = \text{comm}_{n,\, e,\, h}$$
$$\gamma(\text{send}_{h}, \text{receieve}_{h}) = \text{comm}_{h}$$

If we now denote the set of send actions with S, and the set of receive actions with R, we can specify the whole system as

$$\partial_{S \cup R}(ALICE \,||\, BOB)$$

Discussion. We have seen how a process algebra can be defined by giving an abstract syntax and a number of algebraic axioms that can be used to construct and manipulate process expressions. In the example, we also saw in concrete terms how a simple protocol can be specified in terms of algebraic expressions, and how these specifications can be manipulated using the axioms of the algebra. However, we are still lacking an understanding how the formulae are given meaning, or semantics. This discussion is deferred until we have seen a couple of alternative algebraic approaches.

5.3.2 CCS — Calculus of Communicating Systems

The Calculus of Communicating Systems, or CCS, is a process calculus introduced by Robin Milner in his monologue [79], and later revised in [80]. Our presentation is based on [80] and a number of notes given elsewhere, e.g. [8].

CCS was originally defined from a semantic approach using a couple of different LTS semantics as the basis. This approach is different to that of Bergstra and Klop, where the main interest was to consider algebraic systems and finding semantics corresponding to them. However, in spite of this original approach, we introduce CCS on axiomatic bases and discuss the semantics only later along with the semantics of ACP^{τ} and CSP.

As an algebra, CCS has a signature which consist of

- a set of actions Act, which consist of a set of names A, a corresponding set of co-names $\overline{\text{A}}$, and the silent action τ. In other words, $\text{Act} = \text{A} \cup \overline{\text{A}} \cup \{\tau\}$.
- the terminal process 0, corresponding to the deadlock δ of ACP.
- a prefix operator . (period), which is used to sequentially compose a new process from an action and a process. It the set of processes is denoted as P, the prefix can be considered as a function $. : \text{Act} \times \text{P} \to \text{P}$. This in contrast to ACP and CSP, which define a full sequential composition as a basic operator.
- a summation (or choice) operator +, which is similar to the choice of ACP.

- a communicating parallel composition operator $|$, which is similar to ACP $||$. The difference is in the form of communication. In CCS, a name a and a co-name $\bar{a}$ automatically communicate, always forming a silent action τ. Mainly due to this reason, CCS also includes renaming functions. However, we ignore them since they do not contribute much to the insight of semantics.
- a restriction operation \, which is similar to ACP ∂.

CCS also includes recursive specifications in all of its forms. However, we ignore them since they introduce semantic complications that are unnecessary from our point of view. It is also possible to define a value passing calculus which can be translated in basic CCS. This is straightforward and similar in spirit to our ACP example above, where the possible values of variables are translated into individual actions. This point will become clear in the examples, and we will not pursue it formally.

TABLE 10. Some CCS laws along with corresponding ACP$^\tau$ names

CCS law	ACP$^\tau$ axiom name
$P + Q = Q + P$	A1
$P + (Q + R) = (P + Q) + R$	A2
$P + P = P$	A3
$P + 0 = P$	A6
$\alpha.P \mid \beta.Q = \alpha.(P \mid \beta.Q) + \beta.(\alpha.P \mid Q)$ if $\alpha \neq \bar{\beta}$ [a]	CM1-CM3 + CF2
$\alpha.P \mid \beta.Q = \tau.(P \mid Q)$ if $\alpha = \bar{\beta}$	CM1-CM3 + CF1
$(\alpha.P)\backslash L = \alpha.(P\backslash L)$ if $\alpha \notin L \cup \bar{L}$	D1
$(\alpha.P)\backslash L = 0$ if $\alpha \in L \cup \bar{L}$	D2
$(P + Q)\backslash L = P\backslash L + Q\backslash L$	D3
$\alpha.\tau.P = \alpha.P$	B1
$P + \tau.P = \tau.P$	
$\alpha.(P + \tau.Q) + \alpha.Q = \alpha.(P + \tau.Q)$	
$P \mid Q = Q \mid P$	derivable in ACP
$P \mid (Q \mid R) = (P \mid Q) \mid R$	derivable in ACP
$P \mid 0 = P$	derivable in ACP

a. We have only presented a few special cases of the more general expansion law. This makes the similarities with and differences from ACP more apparent.

Some of the laws of CCS are given in Table 10above. Here, the symbols P, Q and R denote arbitrary processes, while α and β are actions. L is a set of names, similar in function to H in ACP$^\tau$ ∂_H. There are a number of additional laws, mostly concerned with relabelling and recursion. Being more syntactic than semantic in nature, we do not consider them. An other difference between our presentation and [80] is that Milner gives a very general expansion law that simultaneously handles communicating parallel composition and restriction. We have only given a few special cases of this law, to make apparent how the consequences of this law corresponds with some features of ACP. The laws named with CF1, CF2 and D1-D3 are actually (some of the) consequences of the expansion law.

Now, there are apparently three basic differences between ACP$^\tau$ and CCS. Two of these, namely the different way of defining parallel composition and the use of sequential composition vs. prefix are more syntactic in nature, and do not have implications in the expressing power or semantics of the calculi. However, the third difference, i.e. the different laws handling silent actions, do impose a difference at the semantic level. This will be discussed in section 5.4.5, "Branching bisimulation", on page 89.

From our point of view, ACP$^\tau$ and CCS are pretty similar, even though there are both syntactic and semantic differences. However, due to its different initial approach, ACP$^\tau$ and other purely algebraic approaches are more suitable for the analysis of different underlying semantics. Therefore we will mainly use notation based on ACP$^\tau$ in the rest of this study.

5.3.3 CSP — Communicating Sequential Processes

CSP, short for Communicating Sequential Processes, is mainly work of Tony Hoare, originally introduced in [54]. Being the first algebraic approach to the theory of concurrency, it can be considered to suffer from some slight semantic difficulties. In particular, the operations defined are not as orthogonal as in pure process algebra, and therefore axiomatisation is more complex. These are mainly due to historical reasons, and maybe stem from improper understanding of invisible behaviour at that time.

However, CSP has quite strong position on the practical side of using process algebraic notations. For example, CSP has influenced a number of concurrent programming languages such as Occam and specification languages like LOTOS. Furthermore, there is a practical analysis tool FDR [38], with the help of which CSP has been applied to the analysis of cryptographic protocols [68, 98].

Now, given all this, we will mostly ignore the semantic difficulties behind CSP and consider it from a very practical point of view. However, it should be understood that CSP can be given similar kind of semantics as the other process calculi, and that the semantic considerations given in the next section apply to CSP as well as to CCS and ACP$^\tau$.

Notation. In CSP, like the other process algebras, a set of atomic actions A is given. While ACP has a explicit communication function γ, and the names and co-names communicate in CCS, in CSP an action $a \in A$ communicates only with itself, i.e. in ACP terms $\gamma(a, a) = a$ for all $a \in A$. The role of the communication function is achieved by explicitly defining the set of actions communicating in a parallel composition. For example, the notion $P \,|[\, \{a, b\} \,]|\, Q$ denotes that the processes P and Q execute in parallel, synchronizing whenever they are about to perform either of the actions $\{a, b\}$.

In CSP, prefixing sequential composition is expressed $a \rightarrow P$ where a is an action and P is an arbitrary process. Sequential composition of processes is expressed as $P;Q$, and defined in terms of basic operators. The starting process is STOP, similar to CCS 0 and ACP δ. There is no distinction between successful termination and deadlock.

CSP does not have silent actions, i.e. there is no action τ. Instead there are two kinds of alternative composition: external choice $\Box$ and internal choice $\sqcup$. The internal choice is completely nondeterministic, and $P \sqcup Q$ can be represented in process algebra as $\tau P + \tau Q$. The external choice is always decided by the environment, and not directly representable in process algebra [8]. However, for practical purposes CSP $P \Box Q$ is approximately equal to ACP $P + Q$.

CSP is usually used with the so called failure semantics (see below). This and the fact that there are two different choice operators make it possible that silent steps are always removed. However, this also means that τ-loops cannot be presented, and therefore divergencies are lost in abstraction. However, the more prominent modern CSP semantics, including the one used in the FDR tool, do preserve some divergencies.

In practical use, CSP actions are usually expressed in a structural form. In this syntax, a sending action is expressed as e.g. $c.s.!d$, where c is a channel, s is a subchannel or maybe a communication primitive, and d is a variable whose value is sent. Similarly, a receiving action is denoted as $c.s.?d$, where d is a variable receiving a value being received. As in case of other process algebras, the sending notion is understood to mean a specific action uniquely determined by the value of the variable being sent, and a receiving notation means an external choice between all the possible actions of receiving a specific value belonging to the value set (type) of the receiving variable.

Example. Returning to our favourite example, we will present a CSP formulation of it. In CSP, it is natural to consider a separate nonce generator in addition to the protocol parties Alice and Bob. It is an recursive process that selects random elements from a given set. The generator can be expressed in CSP as a formula

$$GEN = gen!0 \rightarrow GEN \sqcup gen!1 \rightarrow GEN \sqcup \ldots \sqcup gen!n \rightarrow GEN$$

Here gen is the only output channel of the process. The process nondeterministically selects a value $\{0, \ldots, n\}$ and outputs it to its output channel, evolving back to its initial state.

In CSP, Alice can be described as a process that receives the message m to be sent from the environment, a nonce n from the environment, that sends a message to the network, and finally receives a reply from the network. Giving the name in to the initial input channel, $trans.a$ to the channel via which the message is transmitted to the network, and $rec.a$ to the channel for the reply, Alice can be defined as

$$ALICE = (in?m \rightarrow gen?n \rightarrow trans.a!(n, \mathrm{enc}(m), \mathrm{h}(n, m)) \rightarrow rec.a?\mathrm{h}(n, m + 1)) \,||\, GEN$$

We can give a similar formula to Bob

$$BOB = rec.b?(n, e, h) \rightarrow (trans.b!\mathrm{h}(n, \mathrm{dec}(e) + 1) \sqcup STOP)$$

Here the internal choice represents Bob's ability to refuse to reply to a bad input, in which case he stops.

In order to derive a complete CSP model, we have to define a formula for Eve, the environment. To make things simple, we assume a faithful environment in this case, i.e. an environment that always delivers a sent message unmodified and promptly. Given this, we can give a formula to Eve.

$$EVE = = trans.a?(n, e, h) \rightarrow rec.b!(n, e, h) \,|||\, trans.b?(h) \rightarrow rec.a!(h)$$

Given these definitions, the whole system can be defined as

$$ALICE \,|[\, \{trans.a, rec.a\} \,]|\, EVE \,|[\, \{rec.b, trans.b\} \,]|\, BOB$$

5.4 Semantics

As already briefly mentioned, an important ability of process algebraic methods is the ability to determine if given two process definitions are equivalent, or if a given definition is a refinement of another. These allow one to check if a more specific formula, i.e. an *implementation*, can be considered to fulfil the requirements introduced by a less complex formula, a *specification*. It is also apparent that the definition of equivalence and refinement is not quite obvious or simple. Indeed, there are literally hundreds of proposed semantics used to define various kinds of equivalence classes and partial orders of process formulae. The study of these are called *comparative concurrency study*, and an excellent though somewhat theoretic introduction to it can be found in [41]. In this study, we only consider finite models. However, most of the discussion can be readily generalized to some infinite cases.

In this study, we only consider a couple of different semantics, and even them only from a process graph point of view. The purpose of this presentation is to give a reader a glimpse of the difficulties involved without actually considering any particular semantics in detail. Instead, we will briefly discuss the merits and problems of some more commonly used semantics from a practical point of view. Furthermore, we only describe the semantic equivalencies based on the definitions. The reader should note that it is usually also possible to define a preorder based on a semantic definition, even though not explicitly stated below. The aim of our approach is to motive an uninitiated reader to follow our semantic considerations later in this study, and to appreciate the criticisms that we impose on some attempts to analyse cryptographic protocols.

Pictorial presentation. To make following easier, we will illustrate the main points of discussion with pictorial presentations of example processes. We hope that this will give the reader some insight to the differences of the semantics without the need of actually pursuing to the detailed formalisms. To give a feeling of the notation, some of the ACP axioms are restated along with the example of section “Parallel composition” on page 78 in Figure 18 on page 86. In the figure, subprocesses are represented as triangles or other filled spaces, and individual actions as arrows.

5.4.1 Graph isomorphism

Graph isomorphism is the strongest semantics considered. In this semantics, two processes are considered equivalent (in fact, equal) if their graphical presentations are isomorphic modulo state and edge names. Of course, edge labelling must be preserved.

Formally, two graphs $g = (S, r, T, s, t, A, \lambda, \surd)$ and $g' = (S', r', T', s', t', A, \lambda', \surd')$ are isomorphic iff there is a bijective relation = between the elements of S and S' such that:

- The roots of the graphs are related, i.e. $r = r'$
- The edges are related; formally, for two pairs of related elements $n = n'$ and $m = m'$, there is an edge e between n and m carrying a label a, i.e. $s(e) = n \wedge t(e) = m \wedge \lambda(e) = a$, if and only if there is a corresponding edge e' from n' to m', i.e. $s'(e') = n' \wedge t'(e') = m' \wedge \lambda'(e') = a$.
- Related nodes have similar terminal conditions, i.e. if $n = n'$, then $\surd(n) = \surd'(n')$.

While graph isomorphism respects the rule $x + y = y + x$, already $x + x = x$ and $(x + y)z = xz + yz$ are not equalities according to graph isomorphism. Thus, graph isomorphism is typically too strong to be useful, not further considered in this study.

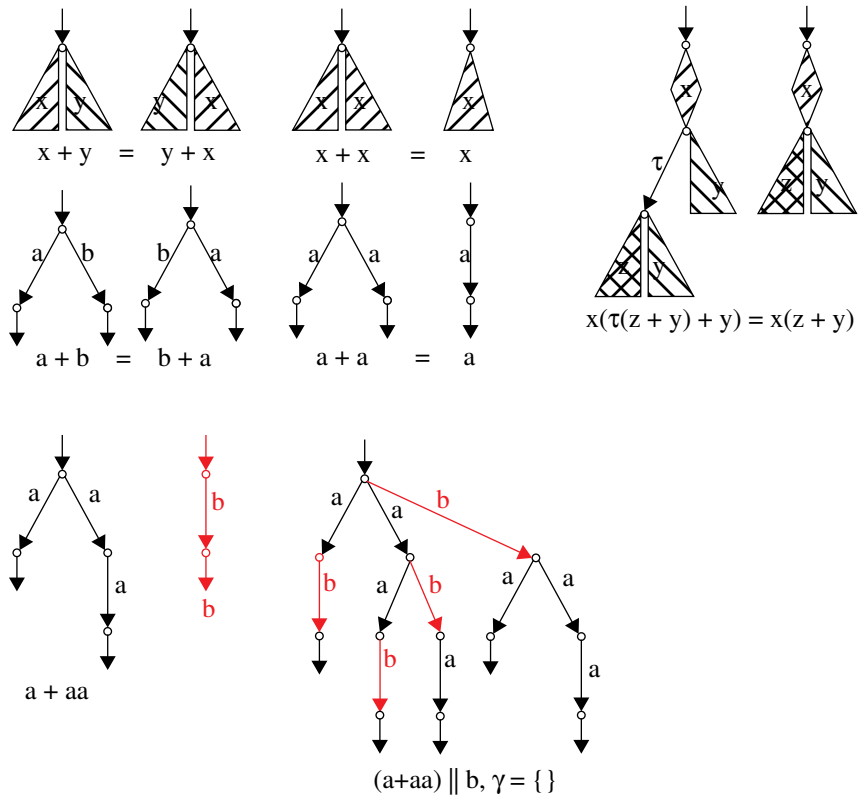


FIGURE 18. Examples of pictorial semantic presentations

5.4.2 Traces

Traces, on the other hand, is the weakest semantics. As already mentioned, in traces semantics two processes are considered equivalent if their trace sets are equal. The trace semantics does not consider the branching structure of a process at all, i.e. there is no information about the timing of decisions made during a protocol run. In addition to the basic trace semantics there are a number other trace based semantics, including completed trace semantics, failure trace semantics and ready trace semantics, to mention but a few. These are beyond the scope of this study; the interested reader would find e.g. Chapter 1 of [41] useful.

Trace semantics are a good solution for some purposes, especially when only safety properties are considered (cf. section “Safety properties” on page 23). However, for most purposes a stronger semantics, i.e. finer set of equivalence classes, is desired. We now turn our attention to some of them.

5.4.3 Strong and weak bisimulation

Bisimulation, or strong bisimulation, between graphs g and g' is defined as a relation $=_{bs}$ between the nodes of the graphs such that:

- The roots of the graphs are related, as in the case of isomorphism
- For each edge of g, there is a corresponding edge in g'; however, this corresponding does not need to be one-to-one, but there may be several edges in g related to a single edge of g'. Formally, for each $e \in T, s(e) = n, t(e) = m, \lambda(e) = a$, denoted

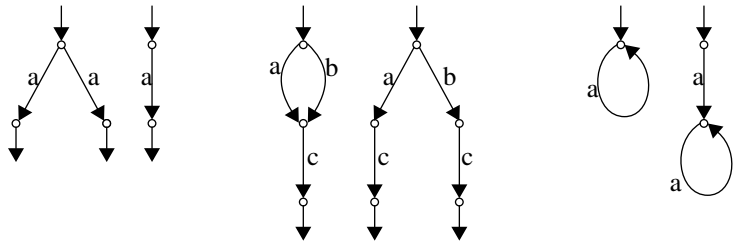


FIGURE 20. Pairs of graphs bisimilar but not isomorphic

$n \xrightarrow{a} m$, for which there is a corresponding starting node $n' =_{bs} n$, there is an edge $e' \in T'$, $s'(e') = n'$, $t'(e) = m'$, $\lambda'(e) = a$, i.e. $n' \xrightarrow{a} m'$ (not necessarily unique) such that $t(e) = m =_{bs} m' = t'(e')$

- Vice versa: for each edge of g', there is a (not necessarily unique) corresponding edge of g. The edge correspondence of this and the previous condition is given in pictorial form in Figure 19 on page 87.
- Finally, related nodes must be similar in terms of termination, as in the case of isomorphism.

Examples of graphs that are bisimilar but not isomorphic are given in Figure 20 on page 87.

Strong bisimulation is quite useful and relatively much used semantics. In fact, if the system under consideration contains no silent moves, and therefore also no divergencies, strong bisimulation is usually the most appropriate semantics if the branching structure of the processes is under consideration. However, the introduction of silent actions, or τ, changes the situation. Weak bisimulation [80], also called τ-bisimula-

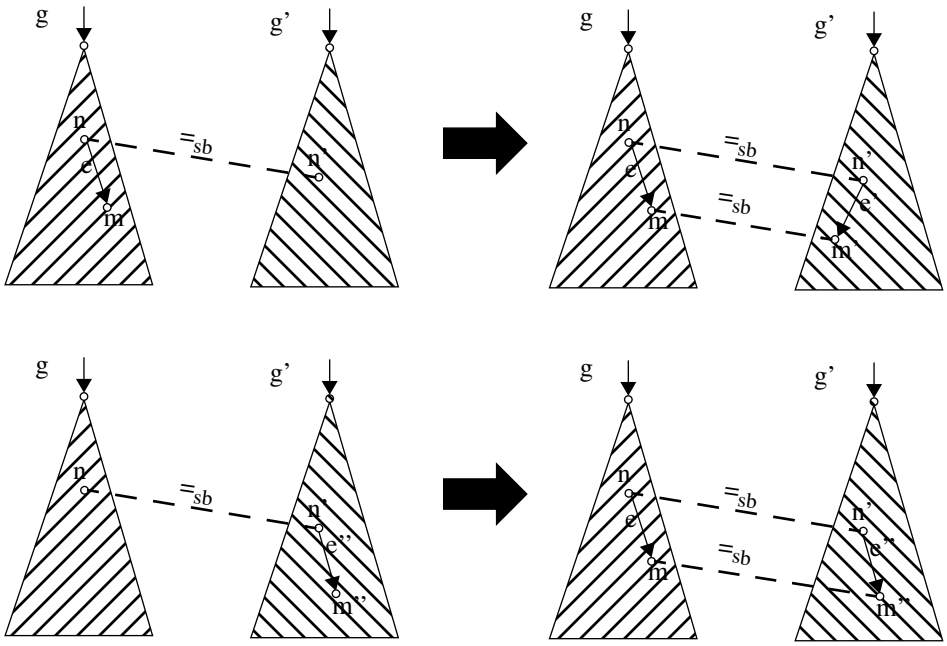


FIGURE 19. Bisimulation condition presented pictorially (based on [8, page 47]

tion and observational equivalence [41, page 122], is one of the earliest attempts to introduce a bisimulation notion in the presence of silent moves. Unfortunately, it has later been argued to be too coarse, i.e. to identify too many processes, as we will see in a moment.

Formally, two graphs g and g' are considered to be weakly bisimilar if there exists a weak bisimulation between them. Now, in order to define weak bisimulation, we have to first introduce the concept of *generalized step*. A single action, or a step, i.e. an edge of a graph g between two nodes n and m carrying the label a, can be denoted as $n \xrightarrow{a} m$. Now, a generalized a-step between two nodes n and m, denoted $n \stackrel{a}{\Rightarrow} m$, is a sequence of steps $n \xrightarrow{\tau} \ldots \xrightarrow{\tau} n' \xrightarrow{a} m' \xrightarrow{\tau} \ldots \xrightarrow{\tau} m$, i.e. a (possibly empty) sequence of silent steps followed by an a-step, followed by another (possibly empty) sequence of silent steps.

With the help of this new definition, weak bisimulation can be defined as a relation $\approx$ between the nodes of two graphs such that:

- The roots of the graphs are related, i.e. $r \approx r'$
- If $n \approx n'$, whenever there is a path $n \stackrel{a}{\Rightarrow} m$ in g, there is a corresponding path $n' \stackrel{a}{\Rightarrow} m'$ in g' such that $m \approx m'$
- If $n \approx n'$, whenever there is a path $n' \stackrel{a}{\Rightarrow} m'$ in g', there is a corresponding path $n \stackrel{a}{\Rightarrow} m$ in g such that $m \approx m'$. [41, page 122; 80, page 108]

This definition is identical to the definition of (strong) bisimulation other than that whereas in the bisimulation case there is a requirement that there is a corresponding *step* in the other graph for each step in the first graph, for weak bisimulation it is enough that there is a corresponding *path* for each path. Weak bisimulation identifies much larger sets of process graphs than bisimulation.

5.4.4 Observational congruence, or rooted τ-bisimilarity

The first, and maybe the worst, problem with weak bisimulation is that it is not a congruence with respect to the choice operator +. That is, given two weakly bisimilar processes P and Q (or, equivalently, two corresponding process graphs), i.e. $P \approx Q$, it is not necessarily the case that $P + R \approx Q + R$. There is a stronger notion, called observational congruence by Milner [80, page 153] and rooted τ-bisimulation by van Glabbeek [41, page 132], which fixes this problem.

It is easiest to define observational congruence by adding a root condition to the definition of observational equivalence, i.e. weak bisimulation. The root condition simply requires that the root of a process graph is only related with the root node of the other graph in either direction. It turns out that this is enough to yield a congruence, and the resulting algebraic laws for τ-elimination can be defined as following:

T1 $x\tau = x$

T2 $\tau x = \tau x + x$

T3 $a(\tau x + y) = a(\tau x + y) + ax$

5.4.5 Branching bisimulation

van Glabbeek argues [41, Chapter 3] that weak bisimulation, or even observational equivalence, is a too coarse notion to really preserve the branching structure of processes. He introduces a new type of equivalence semantics called (rooted) branching bisimulation. It can be axiomatised with the τ-laws

B1 $x\tau = x$

B2 $x(\tau(y + a) + y) = x(y + z)$

As the reader must have noted, these are the τ-laws given the axiomatisation of the algebraic system ACP^{τ} given in Table 9 on page 79. Furthermore, T1 = B1 and B2 can be easily derived from T1 and T2. This is no coincidence. Indeed, based on claims presented by van Glabbeek in his dissertation, (rooted) branching bisimulation equivalence seems to be the most preferable notion of equivalence preserving the branching structure of a process, but simultaneously allowing one to remove a maximum number of silent moves.

An example of the difference between weak bisimulation and branching bisimulation is show in Figure 21 on page 89, where two processes that are weakly bisimilar but not branching bisimilar are given. The reason for the difference is that the right graph is immediately able to perform b, but the left graph must perform τ before being able to perform b. Branching bisimulation makes the difference, but weak bisimulation does not.

Other reasons that van Glabbeek gives to prefer branching bisimulation include the following:

- A version of branching bisimulation called divergence sensitive branching bisimulation corresponds to a variant of CTL* logic without the nexttime operator (cf. section “Syntax and semantics” on page 61).
- No other abstract semantic equivalence is as easy to decide as branching bisimulation.
- Rooted branching bisimulation equivalence is preserved under refinement of actions, whereas weak bisimulation (or observational congruence) is not.

Similar kinds of claims can probably be given for a number of other semantics as well. However, we will not pursue them further, but note the fact that branching bisimulation and branching time temporal logic CTL-X can be considered corresponding.

5.4.6 Handling divergencies

Until now, we have mostly ignored the question of divergencies in our study of semantics. The notion of divergencies is important in practice, since divergencies are easily introduced by abstraction. That is, while no designer usually explicitly constructs divergencies when constructing a specification or an implementation level

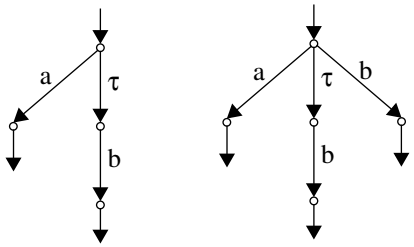


FIGURE 21. Processes that are weakly but not branching bisimilar [41, page 169]

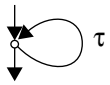


FIGURE 22. Delay, or simple τ-loop

design, the need of comparing processes may lead to their existence. In concrete terms, when an implementation level description is compared with a specification, those actions not present in the specification are abstracted away, or hidden. If the implementation contains loops of actions, e.g. to compensate nondeterminism introduces by random errors, these loops may turn into divergencies by the abstraction process.

The original semantic equivalencies abstracted from all divergencies; for example, the original CSP considered a divergent behaviour as chaos, and claims that no observations are possible past this point. That is, once a process encounters a divergency, nothing can be said about its future behaviour.

A second possibility is to assume that every divergence will be exited sooner or later, if possible. This notion is called *fair abstraction*. Under fair abstraction, τ-loops with an exit point are abstracted away. An alternative way to describe fair abstraction is to note that under fair abstraction deadlock and livelock are considered equal. [8, page 154].

Divergencies can be expressed in algebraic terms by introducing a new symbol, Δ, or delay, which denotes a simple τ-loop (see Figure 22 on page 90). Since loops in the first place can only be introduced by recursive definition (which we have mostly ignored until now), it is not possible to give a simple equational rule of how to handle τ-loops introduced by abstraction. Instead, one can give a number of axioms of how one can simplify recursive formulae. The simplest of these axioms, called DE_1, can be written as follows:

$$\mathrm{DE}_1 \ \frac{x = ix + y}{\tau_i(x) = \Delta \cdot \tau_i(y)}$$

Using Δ, it is possible to describe fair abstraction with the principle $\tau\Delta = \tau$, which is called fairness principle. Using this, one can derive $\Delta = \tau + \varepsilon$, where ε denotes successful termination, and $\Delta\delta = \tau\delta$, which expresses the above mentioned property of fair abstraction of considering deadlock and livelock equivalent. [8]

5.4.7 Failures-divergencies semantics

As discussed above, divergencies are typically introduced in abstraction process. Now, while it is often perfectly reasonable to make the fairness assumption, and thereby ignore non-livelock divergencies (and identify livelocks with deadlocks), the methods certainly ignore some vital information about the processes. Therefore a number of semantics explicitly considering divergencies have been introduced. Handling of these are beyond the scope of this study, and therefore we just enumerate a few:

- The FDR model checking tool introduces by Formal Systems, London, uses a CSP semantics explicitly modelling divergencies. The FDR semantics are trace based, and thus coarser than most bisimulation based semantics [38].

- The CFFD semantics promoted by Valmari et al. preserves divergency information, but still allows a fairly large class of processes to be identified. CFFD is finer than FDR semantics, but still trace based and coarser than the bisimulation semantics. However, is not comparable to weak bisimulation or other bisimulation semantics that do not preserve divergencies [110].
- The divergence sensitive branching bisimulation briefly mentioned in the section of branching bisimulation can be considered as a third example [41].

5.5 Model checking

Having the notation and some insight of the semantic theory of process algebra in our hands, we now turn our attention to model checking, or methods of proving that a process or protocol model has (or has not) a number of desired (or undesired) properties. There are a number of approaches to model checking, but we will present only a few. The model checking approaches we will cover include the following:

- Equivalence and preorder relations are used to determine the above mentioned correspondence between a specification and an implementation.
- Minimization and visualization can be used to find a minimal model equal to one given, and to visualize the minimal model in order to bring insight to the behaviour of the original model.
- Direct property checking can be used to figure out if a model fulfils any number of properties given in some other form, e.g. with temporal logic. Property checking can be combined with minimization if there are equality semantics that we know to preserve the properties.

5.5.1 Tackling state space explosion

Before going to the approaches themselves, we will first briefly describe the so called state space explosion problem and a number of approaches to alleviate it. Typically, a detailed model of a communication protocol consists of a number of processes acting in parallel. In our favourite example Alice, Bob and Eve are all parallel processes. As we have already seen in some of the examples, given fairly general constraints that basically restrict a number of completely deterministic systems outside the consideration, parallel composition causes a well-known *state explosion* problem. That is, the number of states in the composite model tends to grow exponentially in the number of processes. This easily leads to models having billions or even larger number of states.

There are a number of methods designed to alleviate the state space explosion problem in the process algebraic approach. Those we consider here all try to reach the same goal: cut down the number of states to be considered. However, they are applicable in different situations.

First, there are relatively fast (cubic worst case complexity) minimization algorithms based on weak bisimulation or some relative of it. These algorithms take a model representation (i.e. an LTS) of a process and produce another process that is minimal in number of states and equivalent to the first one in terms of used bisimulation semantics. The technique can be combined with symbolic representation allowing different values of variables to be folded together.

Another possibility is to use failure- and/or divergence preserving semantics such as CFFD to produce a smaller model. These methods use both heuristic methods and algorithms such as stubborn sets, and typically produce models that are smaller than

the original one, but not necessarily minimal. In fact, it has been shown that minimization of a process with respect to a failure-divergencies semantics has a worst case complexity in PSPACE. However, this does not mean that the semantics are inapplicable; a result less than optimal is often good enough [111].

It is also possible to apply an equivalence (or a preorder relation) on the fly, i.e. during a testing process, on the model considered. For example, it is possible to apply the stubborn sets method during the verification and thereby lessen the number of states that need to be checked. In the area of model checking based verification of cryptographic protocols, most methods seem to be based on the idea of limiting the number of states during the verification. However, it seems to be more usual to apply some kind of problem specific heuristics instead of process algebraic equivalence theories.

Finally, a method which seems to be surprisingly easy to apply but apparently unused in the area of cryptographic protocols, is the reduction of state spaces at the level of compositional construction. This method takes advantaged of the fact that the protocol models, like most concurrent processes, are constructed from several components. Now, if we have a equational semantics that preserves the properties we are interested in and that is a congruence with respect to the composition operators (typically parallel composition and hiding), it is possible to first reduce the models to be combined, then apply the construction operator, and reduce the model again. [111]

5.5.2 Comparing process models

If the properties a protocol needs to fulfil can be defined in terms of a process model, it is natural to perform model checking by comparing protocol models. In this setting, we have two distinct models of behaviour. The first one, a specification, is typically fairly simple, given directly as a single model describing only an external view of a system, and attempts to describe how an implementation should behave. The other process model, an implementation, is more detailed, contains some kind of representation of the communication environment the protocol is meant to operate in, and is typically given as a parallel composition of several process models.

The aim of the verification is to check if the implementation is equal to the specification with respect to some semantic model, or, alternatively, to determine if the implementation is an refinement of the specification, i.e. a more detailed model in the preorder defined by the semantics. In this kind of verification it is possible to use compositional LTS reduction as well as bisimulation and/or failures-divergencies based minimization algorithms, depending on the properties represented by the specification.

5.5.3 Visualisation approaches

Another possibility, promoted user friendly by Valmari et al., is based on the idea of reducing an implementation model small enough so that human inspection is feasible. The basic idea is to hide all but a few actions of the model, produce an reduced model with respect to this hiding, and to inspect the implementation form this point of view. It is possible to use different sets of actions left visible, thereby checking that the implementation fulfils the mental requirements from several points of view. This method seems to be promising in the sense that it may bring new insight to the possible execution paths of the implementation level mode.

5.5.4 Checking validity of formulae

Typically protocol specification, or some part of it, is not given as a process model but as a number of logical formulae or other type of queries. The aim of verification is to check if these properties hold in all possible states the model can reach. In this case, semantic equivalencies and algorithms based on them must be seen as tools: the aim of the equivalence is to diminish the number of states to be check. Typically one of the problems in this setting is to ensure that the state space reduction algorithms does preserve the properties expressed by the formulae. That is, any model modification performed in the name of semantic equivalence may not remove nor introduce any phenomena that could affect to the properties one is trying to check.

Again, it is enlightening to notice that there is no need to require that any reduction algorithm would produce a minimal equivalent process model; it is enough to reduce enough to bring the model within the limits that can be checked in reasonable time. Now, it should be clear by now that the weaker the used semantics, the coarser the equivalence classes are, and thereby the size of the smallest models within an equivalence class diminish as the semantics gets weakened. Thus, the weakest semantics that preserves the interesting properties is ideal, since the set of potential reduction tools is largest. Therefore it is important to analyse the desired properties in terms of the semantic theories, and, if compositional construction is taken advantage of, combine the analysis results with congruence requirements.

5.6 Summary

In this chapter, we have briefly discussed the properties of concurrency in general, thereafter concentrating on process algebraic models and methods. We first introduced the basic concepts of states, actions and events, thereafter briefly discussing how to model communication, what are deadlocks and livelocks in terms of actions and events, and what is the purpose of abstraction and hiding. The basic model behind all our discussion is that of process graphs, or, labelled transition systems. Of the existing process algebraic formalisms we briefly covered ACP$^{\tau}$, CCS and CSP.

In the second part of the chapter, we concentrated on semantic issues. The semantics were viewed from the point of view of graph equivalence classes: a strong semantics considers few graphs equivalent, a weak semantics larger groups. Graph isomorphism is clearly the strongest semantics, while plain trace semantics is the weakest practical one. The problem with trace semantics is that it does not preserve the branching structure of a process. Various kinds of bisimulations have been proposed to take care of the branch preserving problem, including strong bisimulation, weak bisimulation (aka observational equivalence), rooted τ-bisimulation (aka observational congruence), and branching bisimulation. However, these do not handle divergencies very well. The problem of handling divergencies was touched only lightly, mentioning a few semantics, since most cryptographic protocols tend to behave pretty regularly in respect to divergent behaviour.

Finally, we introduced the concept of model checking. First, the problem of state space explosion, and some possible ways to alleviate it, was discussed. Thereafter three different model checking approaches were mentioned: model comparison vrt. some semantics, human inspection based on visualisation, and checking of the validity of logical or other formulae.

PART III

CHAPTER 6 *Comparison of some BAN-based approaches* *97*

Introduction to the selected papers 97
Comparison of syntactic approaches 104
Differences in the semantic approaches 106
Summary 110

CHAPTER 7 *Future directions* *111*

Process algebras and protocol models 111
Temporal and modal interpretation 112
Summary 114

CHAPTER 8 *Conclusions* *115*

CHAPTER 6

Comparison of some BAN-based approaches

In this chapter we refer and analyse a number of protocol analysis papers. The papers represent advancements in the area of approaches based on epistemic and doxatic logic. We have selected the papers partly based on the number of citations and partly by our own intuitive feeling of what seems to be more and what less important. Based on this criteria, the papers include the well known BAN [20], CKT5 [14], GNY [46], AT [2], and SvO [108] approaches as well as a lesser know papers by Paul Syverson [106], and Wedel and Kessler [113]. Paper [106] represents Paul Syverson's approach in adding time to a BAN like logic, while the [113] paper represents a fairly complicated approach with some good and some undesirable properties. Some of the discussion has been influenced by [77, 97].

The main points that we will concentrate on are protocol idealization vs. explicit message recognition, monotonicity of beliefs, and the differences in underlying semantic models. In general, the real problem is how to interpret the meaning of a message.

The rest of this chapter is organized as follows. In the first section, the papers are described in rough historical order, pointing out the major contributions in each paper, the most important differences between them, and also discussing some criticism. This discussion is purposefully kept at somewhat superficial level, without going into details; the next two sections attempt to go in greater depth. In the second section of this chapter we have colleted together the syntactic reasoning rules of the various logics. This perspective lets us to see the differences between the approaches in a very concrete way. The section after this, on the other hand, concentrates on the semantic models of the logics, and discusses, among other things, the deep differences between protocol idealization versus explicit message recognition combined with initial beliefs. A summary concludes the chapter.

6.1 Introduction to the selected papers

In this section a number of modal logic based cryptographic protocol analysis techniques are considered in some detail. The selected papers come in two flavours: the work of Bieber (CKT5) represents a separate direction; the rest of the techniques belong to the BAN family.

We will first consider the original BAN approach. However, since it is generally considered outdated, the consideration is brief and sketchy, concentrating only on the main points. CKT5 is discussed next to give perspective. The rest of the section describes GNY, AT, SvO, adding time by Syverson, and AUTLOG by Wedel and Kessler.

6.1.1 The original BAN logic by Burrows, Abadi and Needham

The paper titled "A Logic of Authentication" by Michael Burrows, Martín Abadi and Roger Needham [20] is usually considered to be the seminal work in the field of modal logic based analysis of cryptographic protocols. The simplicity of the application of the technique, as well as the wide publicity due to the multiple publication of the paper, quickly led to several applications of the technique. However, the technique itself has both theoretical and practical drawbacks, and therefore the derivatives are favoured over the original BAN in practical use.

Figure 23 below depicts the analysis process of the BAN approach. The outlined box contains the formalized part of the process: turning a protocol specification into an idealized protocol, and identifying the beliefs the parties initially possess are left for informal reasoning.

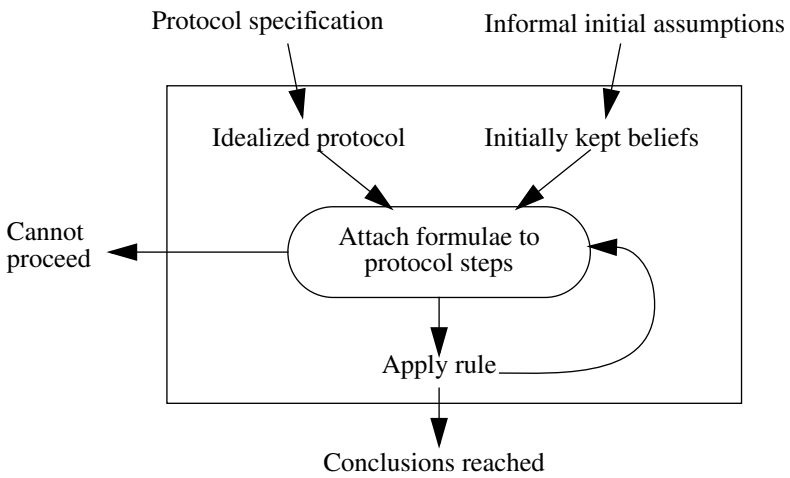


FIGURE 23. BAN-logic analysis process (Adapted from [97, page 15])

The BAN-approach makes a number of external assumptions that have been later relaxed in other approaches, e.g. BAN assumes that all parties can remember what messages they have sent, ever. Furthermore, the designers of BAN made a conscious decision to avoid explicit notion of time, and to base the belief analysis on operational notation of beliefs, as opposed to the intensional notion of beliefs used elsewhere this study.

Having been very successful in practical terms, later research has found several deficiencies in the BAN approach. For example, Mao and Boyd [71] describe four weaknesses: the protocol idealization is too informal and flexible, the logic mixes both messages and formulae in the belief system, the system of deriving the initial beliefs (assumptions) is flawed and does not give the weakest set of initial beliefs possible, and BAN does not consider confidentiality. The last one, i.e. handling confidentiality, was not a goal in BAN. In our opinion, however, the remaining three points are well founded.

According to our experience, there seems to be some problems involved with the operational approach to beliefs as well. In particular, the usage of an explicit, monotonously growing set of believes for each agent as the semantics seems to be somewhat arbitrary. That kind of semantics gives relatively little backing theory to determine whether a belief held by a party actually holds at a particular moment of time. Furthermore, determining and handling beliefs that are initially true but turn later false is problematic. We will return to the issue of semantics in section 6.3, "Differences in the semantic approaches", on page 106.

6.1.2 CKT5 by Bieber

CKT5 by Pierre Bieber [14] represents a quite different approach to protocol modelling. In fact, Bieber's approach resembles somewhat to the approach we have outlined in chapter 7. The approach concentrates on giving a reasonable semantics for a modal logic describing cryptographic protocols, and discusses how the resulting logic can be used to describe security properties of the logics.

Bieber considers the cryptosystems used to be ideal, as usual. He assumes that there is a separate unreliable but honest network, a number of honest agents and a number of malicious agents. In his communication model, each message sent is received by someone (a honest or malicious agent), and each message received was sent by someone. He assumes that messages are constructed from basic messages in an unambiguous way; we will return to this point below. Basically this means that he considers it impossible to decrypt or encrypt messages without the appropriate key.

The basic model consists of a set of agents AGT, a set of actions, which can be sending, receiving or internal actions, and the local histories of agents. A local history consists of some initial state I_a of the corresponding agent, a set of messages $S(a, 0, t)$ the agent has sent between the start of time and a given point of time t, and a set of messages $R(a, 0, t)$ the agent has received before t. It should be noted that what Bieber calls a local *history* is actually a local *state*, i.e. a history up to a moment of time. A global history (i.e. state) is a set of local histories. From now on, we will deviate from Bieber's terminology and speak about local and global states instead of histories.

As usual, there may be several global states corresponding to a single local state. This forms a number of state equivalence classes per agent, giving a usual multimodal Kripke structure (cf. section 4.4.2, "Knowledge of different agents", on page 57). However, in the Bieber's model, there is a global time (even though the agents do not necessarily "know" this), and the knowledge relations are considered only among local states at a given time. In our opinion, this would complicate practical analysis considerably, since for each local state (of a given agent) all states of the other agents must be explicitly considered, figuring out which of them are possible and which of them are not, given the causal relationships of the sending and receiving actions. The approach leads also to another complication: the fact whether a message has been sent (or received) or not, is considered a modal connective; we will ignore this detail in the following.

Comprehension of messages. Bieber's approach to recognizing, or comprehending, messages is interesting. The idea is based on two parallel term algebras: a free algebra, similar to the term rewriting systems used by Dolev and Yao, and a cryptoalgebra, representing the algebra of the actual bit patterns carried in the protocol messages. A party is said to *know* the meaning of a bit pattern (e.g. a term of the cryptoalgebra) iff its interpretation in the terms of the free algebra is identical in all worlds the party considers currently possible. For a basic term, this property is defined as $K_{a,t}\mathrm{clear}(x)$, meaning that the interpretation of x is clear to a at time t.

Formally, Bieber defines a predicate univoque [1] to denote a party's ability to fully comprehend the contents (and therefore the meaning) of a message. In other words, an univoque message has to be constructed of clear text messages using concatenation and encryption with keys that are known to the party. Using a notation resembling the original, the fact that a message m is comprehensible to an agent a at time t can be expressed as

$$\begin{aligned}&\text{univoque}(a, t, m) \leftrightarrow \\ &\quad K_{a,t}\text{clear}(m) \\ &\vee (\exists k : K_{a,t}\text{key}(k) \wedge \text{univoque}(a, t, d(k, m))) \\ &\vee (\exists m_1, m_2 : m = \langle m_1, m_2 \rangle \wedge \text{univoque}(a, t, m_1) \wedge \text{univoque}(a, t, m_2))\end{aligned}$$

where $d(k, m)$ denotes decryption, $\text{clear}(m)$ means that m is a plain text message, and $\text{key}(k)$ means that k is a valid key. $K_{a,t}\varphi$ means that the party a *knows* at time t that φ is true, i.e. φ holds in all states a considers possible at the current state.

Bieber's partial clearness is a notion corresponding to *recognition* of messages in GNY and AUTLOG. He defines a predicate p_clear :

$$\text{p_clear}(m) \leftrightarrow \exists m' : \text{belongs}(m', m) \wedge \text{clear}(m')$$

where

$$\begin{aligned}&\text{belongs}(m', m) \leftrightarrow m = m' \vee (\exists m_0, k : \text{key}(k) \wedge m' = d(k, m_0) \wedge \text{belongs}(m_0, m)) \\ &\vee (\exists m_0, m_1 : (m_0 = \langle m_1, m' \rangle \vee m_0 = \langle m', m_1 \rangle) \wedge \text{belongs}(m_0, m))\end{aligned}$$

This basically expresses that a message is partially clear (i.e. recognizable) iff there is a submessage of it that is clear, i.e. plain text. A message is a submessage, on the other hand, if it is either the containing message itself, or the containing message has the submessage either due to concatenation or as an encrypted part of it. The reader should note how the definition of belongs differs in this last respect from the notion of submsg used in AUTLOG.

Secrecy, integrity, and authenticity. Using notations introduced above, Bieber gives formal definitions for secrecy, authenticity and integrity, among other things. We repeat these definitions (or slight modifications of them) here using a different language based on the usual notation used in this study.

The definition of secrecy is relatively simple:

$$\begin{aligned}\text{private}(\{a, b\}, x) \leftrightarrow \Box\{&a \text{ has } x \wedge b \text{ has } x) \\ &\wedge \Box (a \text{ knows } (c \text{ has } x \rightarrow c \in \{a, b\}) \\ &\quad \wedge b \text{ knows } (c \text{ has } x \rightarrow c \in \{a, b\}) \ )\end{aligned}$$

Basically this states that an item x (e.g. a key) is known only to a and b if and only if they (originally) know it, and they can be always (at all future times) sure that if there is a party c that knows the item, then c must be either a or b. This requirement seems to be too strict, and in fact it stems from Bieber's opinion that secret sharing requires *common* knowledge about the possession of the secret and that no-one outside the group knows (possesses) the secret.

1. Using the terminology of AUTLOG, a similar concept is the *localization* of a message, cf. section 6.1.7, "Yet another approach: AUTLOG by Wedel and Kessler", on page 104.

Bieber's definition for integrity does not seem to consider message integrity as we know it, and remains somewhat obscure. The lemmas he gives express various facts what a party can infer from an encrypted message it cannot decrypt. He seems to intend to show that a key is necessary to create a message that contains an encrypted component.

Bieber's notation for authenticity of a message, translated in our language, can be expressed as a formula

$$a \text{ has } k \wedge a \text{ knows } (a \text{ recognizes } \{m_0\}_k) \wedge a \text{ knows } \text{submsg}(m_0, m_2) \wedge a \text{ reads } m_2 \\ \rightarrow \exists m_1, b : b \text{ haswritten } m_1 \wedge \text{submsg}(m_0, m_1) \wedge b \text{ has } k \wedge b \text{ has } \{m_0\}_k$$

This states that if a receives a message m_2 that contains an encrypted submessage m_0 that can be decrypted with some key k known to a, then there is some party b that has sent a message m_1 containing the submessage m_0, and b knows k. Combined with the definition of privacy of keys this allows one to detect when a message has been sent by a particular party, and thereby assure its authenticity.

Proving security. In addition to defining the security properties in the model of computation provided, Bieber shows how confidentiality (and timeliness) of a message can be protected with proper usage of nonce and encryption. This proof is not particularly interesting.

The fundamental value of Bieber's work is in showing that security properties can be defined in terms of low level communication primitives. He is also among the first authors applying Kripke structure based semantics to the knowledge formulae in the analysis of cryptographic protocols. However, his definitions and semantics seem to be unnecessarily complicated, and simpler approaches have later emerged (e.g. AT, AUTLOG, and the approach given in chapter 7 of this work).

Furthermore, Snekkenes [104] applies CKT5 to a protocol known to have a flaw, and is able to use CKT5 to prove that the protocol is secure. This indicates that a strictly epistemic logic is probably not sufficient for analysing the security of authentication protocols [97].

6.1.3 The GNY logic of Gong, Needham and Yahalom

The GNY approach is the first one to introduce the notion of *recognizability* and to properly distinguish between possessing a message and believing in a statement. Basically, instead of performing an informal protocol idealization step, the actual protocol messages carrying actual data are analysed. Only if a message can be recognized, i.e. it contains some data or structure already known and expected by the receiver, the meaning conveyed by the message can be accepted and added to the set of beliefs.

Another advancement in GNY is the introduction of the rationality rule, i.e. if it is universally true that $\varphi \rightarrow \psi$, then $(a \text{ believes } \varphi) \rightarrow (a \text{ believes } \psi)$. This corresponds to the reasoning rule modus ponens and the modal axiom K used in the later logics.

The handling of beliefs is still somewhat arbitrary. A person analysing the protocol will explicitly attach the beliefs a sender want to express to the protocol messages sent. If a receiver of a message is able to recognized the message, and can believe in its authenticity of its integrity and origin, the receiver may deduce that the sender of the message actually believed the meaning conveyed. In GNY the attachment of

beliefs is performed more mechanically and with less human judgment than in BAN. However, the process can still be considered to be an informal idealization process.

The GNY approach leads to relatively complicated deduction formulae, and impoliticly requires that the sender and the originator agree on the meaning of messages. However, GNY allows the receiver to determine the level of trust it has to the sender, and explicitly decide whether it accepts the beliefs of the sender as its own beliefs.

6.1.4 The Abadi-Tuttle (AT) logic

The AT logic is one of the more successful attempts to give better than original semantics for a BAN-like logic. AT is also the first paper describing possible-world semantics at the level of reasoning rules — the earlier attempts (e.g. Bieber) have more concentrated on the lower level details of security. The semantics are considered in more detail in section 6.3, “Differences in the semantic approaches”, on page 106.

In addition to the new semantics, AT makes a distinction between a protocol party sending some message during the current protocol run versus having sent the message at all i.e. maybe during an earlier protocol run. It also introduces a distinction between messages that have been constructed by a protocol party and those that merely have been forwarded by it. In our opinion, this latter distinction is not so useful in practice, since it is hard to determine. A better approach is to clearly distinguish actual messages from the meanings they convey, as in GNY or AUTLOG.

The state model of AT consists of send, receive and key generation actions. Information (i.e. received messages and generated keys) cannot be lost; thus, the possession set of a protocol party is always monotonic. The possessions of parties and local histories combined give straightforward rules to determine which histories, i.e. interleavings of actions, are possible and which are not. To distinguish between originally generated and forwarded messages, a protocol party’s information (possession) set is divided into messages and submessages *seen*, and messages and submessages *sent*. The difference between them determines what has only been forwarded, and what is originally generated.

Abadi and Tuttle do not consider negative beliefs, thereby retaining the monotonicity of the beliefs in addition to the monotonicity of the information.

6.1.5 Towards unified semantics: SvO

A more recent paper [108] by Paul Syverson and Paul C. van Oorschot attempts to unify the approaches of BAN, GNY, AT and VO [87], and succeeds in defining a simple logic (only 20 rules) with a good model theoretic, possible-worlds based semantics. From our point of view, the only technical deficiency in this approach is the mixing of actual messages and logic formulae, thereby requiring an idealization phase along the lines of BAN and AT. The full syntax and semantics of SvO will be described below and compared to those of the earlier approaches as well as the one used in AUTLOG.

The SvO approach starts from an idealized version of a protocol, along the line of BAN and AT. However, their language also considers public keys, key agreement (i.e. DH public key), functions and message comprehensibility. Their language contains a set T_o of primitive terms (i.e. principals, nonces, keys, constants etc.), and a set T of terms that can be recursively constructed using functions. They also allow messages even in the idealized protocol to contain both terms (i.e. information) and formulae (i.e. beliefs and statements).

Formally, the SvO language of messages consists of any term $x, x \in T$, any formula $\varphi, \varphi \in F_T$, and any composition of messages $F(x_1, \ldots, x_n)$ where $x_1, \ldots, x_n$ are messages and F is any function. This is fundamentally different from GNY and AUTLOG. In GNY, a formula can only appear as an extension of a term in a message, and in AUTLOG formulae cannot appear within messages at all.

The set of formulae is quite similar to other doxatic logics used. It is constructed of

- primitive propositions $p, p \in \Phi$,
- propositional connectives $\neg\varphi$ and $\varphi \wedge \psi$,
- modal connectives a believes φ and a controls φ,
- message predicates a sees x, a received x, a says x, a said x, and fresh x, whose truth value can be determined by studying the local history of a (with fresh x being an exception),
- key validity predicates $a \overset{k}{\leftrightarrow} b$ and $\mathrm{PK}(a, k)$, and key possession predicate a has k.

They also adopt a convention $\tilde{k}$ for complement keys, and $\tilde{F}$ for complements of bijective functions. The full set of SvO reasoning rules and axioms, using the syntax of this paper, is given in Appendix C, “Rules in the modal approaches”, on page 133. On the side of reasoning rules, Syverson and van Oorschot explicitly note that the necessitation inference rules may only be applied to theorems. The model theoretic semantics makes it possible to determine the validity of initial assumptions in each given setting.

SvO does not attempt to address time or message ordering. In particular, SvO is not able to detect some replay attacks that can be detected with some other logics. In fact, they explicitly note that “the (in [106]) introduced temporal operators are necessary if one is to even express such criteria [i.e. interleaving or certain kind of replay attacks] in a BAN-like logic.” [108, Section 5].

However, in whole SvO is clearly one of the better BAN-derivatives, even though it does not attempt to directly address the protocol idealization step and its drawbacks.

6.1.6 Adding time by Paul Syverson

In [106] Paul Syverson extends the AT logic with temporal operators. The new logic is sound with respect to the model semantics defined for AT. Thus, using the same model of computation, Syverson adds a number of standard temporal axioms:

K $\quad \Box(\varphi \rightarrow \psi) \rightarrow (\Box\varphi \rightarrow \Box\psi)$

4 $\quad \Box\varphi \rightarrow \Box\Box\varphi$

D $\quad \Box\varphi \rightarrow \Diamond\varphi$

L $\quad \Box(\varphi \wedge \Box\varphi \rightarrow \psi) \vee \Box(\psi \wedge \Box\psi \rightarrow \varphi)$

Z $\quad \Box(\Box\varphi \rightarrow \varphi) \rightarrow (\Box\Diamond\varphi \rightarrow \Box\varphi)$

In addition to the axioms, the necessitation rule applies to the past time operator $\Box$.

Within the example given in the paper, the temporal operators are used to define a number of causal requirements and faithfulness assumptions. The causal require-

ments states the requirement that in a correctly functioning protocol a message received is preceded with a corresponding message sent by the assumed, legitimate protocol party. Similarly, the faithfulness assumptions state that a protocol party sending a message is in a legitimate state, i.e. it actually has received a message that is required to trigger the sending of the message. These formulae are used to define "criterion for causal consistency", which states that if the protocol parties are faithful, the system will fulfil the causal requirements. Failing to fulfil the causal consistency criterion is an indication that the protocol has some kind of a problem.

6.1.7 Yet another approach: AUTLOG by Wedel and Kessler

Wedel and Kessler describe in their paper [113] yet another extension of BAN. The logic is based on a semantic model similar to AT, but contains explicit message recognition rules somewhat similar to GNY. This different initial approach makes it possible to analyse protocols without first idealising them. In our opinion, this is a clear advantage compared to the earlier approaches.

The important new notion Wedel and Kessler introduce are inherently connected to the notions of message comprehension and recognition. These concepts, though in different format and in different setting, are already familiar from e.g. CKT5 [14]. However, [113] is the first to introduce them in the setting of a BAN-like logic. The idea introduced is based on the concept of *localization.* All messages received from a network by a party are first localized. This means that the party tries to determine the structure and information content of the message using the information it already has. For example, if a party receives an encrypted message, i.e. a reads $\{x\}_k$, and has the relevant key k , it is able to decrypt the message into a meaningful plaintext version. This notion is expressed with the formula a believes $(a$ reads $\{x\}_k)$, which means that in addition to *having* the message $\{x\}_k$, it also knows the structure of the message, i.e. that it is x encrypted with k , or at least believes so. If, on the other hand, a does not possess k , we can only deduce a believes $(a$ reads $(\{x\}_k)_a)$ where $(\{x\}_k)_a$ is the message *localized towards* a , and in the absence of k effectively identical to garbage.

The actual mechanism used to localize messages is based on a function $sight_k^{(r,k)}$, whose definition and application is somewhat complex. A minor further confusion is caused by using the symbol $\equiv$ to denote the localization process within a proof. Despite of these minor difficulties, the notion of message comprehension, and the idea of attaching beliefs to comprehended messages via explicit initial assumptions is novel and beautiful.

6.2 Comparison of syntactic approaches

The different approaches in different logics have led to different set of axioms and reasoning rules in the logics. In the first logics (BAN and GNY), there was only reasoning rules. Replacing the operational notion of belief, based on explicit beliefs sets, with an intensional notion of belief, based on modal logic, replaced this setting by introducing two reasoning rules: modus ponens and necessitation, and giving the rest of rules as axioms. This simultaneously led to a cleaner set of rules by shifting most of the reasoning from belief level to non-belief level, i.e. by leaving out the explicit modal operator from the rules.

We have collected *some* of the reasoning rules of the logics to Table 11 below. The table is by no means complete. Also, in most cases we have ignored minor differences between rules carrying essentially the same intuitive meaning. However, occasionally these differences have been interesting enough so that we have given two

version of an essentially same rule. These rule pairs are indicated with a large curly brace at the right hand side of the table. However, due to our ignoring of minor details, the reader is advised to study, in addition to the table, appendix C, "Rules in the modal approaches", on page 133, and preferably the original papers as well.

TABLE 11. Comparison of the rules of the different logics

Rule or axiom	BAN	GNY	AT	SvO	WK
If $\vdash\varphi$ and $\vdash\varphi \rightarrow \psi$, then $\vdash\psi$	-	-	R1	MP	MP
If $\vdash\varphi$, then $\vdash a$ believes φ	-	M	R2	N	M
a believes $\varphi \wedge a$ believes $(\varphi \rightarrow \psi)$ $\rightarrow a$ believes ψ	-	-	A1	1	K
a believes $\varphi \rightarrow a$ believes (a believes φ)	-	-	A2	2	4
$\neg$(a believes φ) $\rightarrow a$ believes $\neg$(a believes φ)	-	-	A3	-	5
(a believes $\varphi \wedge a$ believes ψ) $\leftrightarrow a$ believes $(\varphi \wedge \psi)$	✓	-	A4	der?	der
(a controls $\varphi \wedge a$ believes φ) $\rightarrow \varphi$	✓	J1	-	-	J1
(a controls $\varphi \wedge a$ writes φ) $\rightarrow \varphi$	-	-	A15	15	-
fresh $x_i \rightarrow$ fresh $\langle x_1, \ldots, x_n \rangle$	✓	F1	A16	16	F1
fresh $x \rightarrow$ fresh $f(x)$	-	F1–F6	A17	17	F2
a haswritten $x \wedge$ fresh $x \rightarrow a$ writes x	✓	J2	A20	18	NV
a reads $x \rightarrow a$ has x	-	P1	-	8	H1
a has $x_1 \wedge \ldots \wedge a$ has x_n $\rightarrow a$ has $f(x_1, \ldots, x_n)$	-	P2,P4, P6–P8	-	10	H2+ H3
a has $\langle x_1, \ldots, x_n \rangle \rightarrow a$ has x_i	-	P3	-	9	SE1 +H1
a reads $\langle x_1, \ldots, x_n \rangle \rightarrow a$ reads x_i	✓	T2	A7	6	SE1
a reads $\{x\}_k \wedge a$ has $\tilde{k} \rightarrow a$ reads x	✓	T3,T4, T6	A8	7	SE2
a haswritten $\langle x_1, \ldots, x_n \rangle \rightarrow a$ haswritten x_i	-	I7	A12	13	SA1
a writes $\langle x_1, \ldots, x_n \rangle \rightarrow a$ writes x_i	-	I7	A12	14	SA2
a haswritten $h(x) \wedge \neg$(a reads $h(x)$) $\rightarrow a$ haswritten x	-	I3	A13	-	SA3
a writes $h(x) \wedge \neg$(a reads $h(x)$) $\rightarrow a$ writes x	-	I3	A13	-	SA4
$a \overset{k}{\leftrightarrow} b \wedge a$ reads $\{x\}_k \rightarrow a$ haswritten x	✓	I1	A5	3	-
$\left(a \overset{k}{\leftrightarrow} b \wedge c \text{ reads } \{x\}_k \wedge \neg(a \text{ haswritten } \{x\}_k)\right)$ $\rightarrow (b$ haswritten $\{x\}_k)$	-	-	-	-	A1
$\overset{k}{\rightarrow} b \wedge a$ reads $\{x\}_{k^{-1}} \rightarrow b$ haswritten x	✓	I4	-	4	-
$\overset{k}{\rightarrow} b \wedge a$ reads $\{x\}_{k^{-1}} \rightarrow b$ haswritten $\{x\}_{k^{-1}}$	-	-	-	-	A2
$\left(a \overset{k}{\leftrightarrow} b\right) \leftrightarrow \left(b \overset{k}{\leftrightarrow} a\right)$	✓	impl- icit	A21	19	S

TABLE 11. Comparison of the rules of the different logics

Rule or axiom	BAN	GNY	AT	SvO	WK
$\alpha \xrightarrow{x} a \wedge \alpha \xrightarrow{y} b \rightarrow a \overset{g^{xy}}{\leftrightarrow} b$	-	-	-	5	KA
$a \overset{k}{\leftrightarrow} b \wedge \text{good}_f(x) \rightarrow a \overset{f(k,x)}{\leftrightarrow} b$	-	-	-	-	KD
a believes (a has $h(x)$) $\rightarrow a$ believes (a has x)	-	-	-	11+ 20	-
a has $h(x) \rightarrow a$ believes (a recognizes x)	-	R6	-	-	-
a believes (a has x) $\wedge$ a reads $h(x)$ $\rightarrow a$ believes (a reads $h(x)$)	-	-	-	12+ 20	C3+ C
a has $x \rightarrow a$ recognizes $h(x)$	-	R5	-	-	R3
a recognizes $x_i \rightarrow a$ recognizes $\langle x_1, \ldots, x_n \rangle$	-	R1	-	-	R1
a recognizes $x \wedge a$ has $\tilde{k}$ $\rightarrow a$ recognizes $\{x\}_k$	-	R2-R4	-	-	R2
a has $x \wedge a$ has $k \rightarrow P$ recognizes $\{z\}_{k^{-1}}$	-	-	-	-	R4

Maybe the most interesting point in the table is the last section. There we can see how the GNY recognition rules R1–R6 can effectively be covered by a number of simpler rules. The table indicates clearly how BAN and AT do not consider message recognition at all and how the SvO approach seems to be somewhat limited, while WK attempts to cover the same are GNY covers, and somewhat more. Other than this, the table may speak for itself.

6.3 Differences in the semantic approaches

A deeper difference between the discussed approaches can be found in their semantic model, and model of computation.

6.3.1 Models of computation

In the original BAN approach, a local state of a party a consists of two sets: a set of formulae M_a and a set of beliefs B_a. These sets are transitively closed, including all the formulae (messages) and beliefs that can be deduced from the contents of the sets. A global state is a tuple of local states, as usual. A run, on the other hand, is simply a finite sequence of global states where the sets M_a and B_a monotonously grow for all a. A run is a run for a protocol, if all the messages described in the protocol appear in the message sets M_a of the receiving parties at an appropriate moment.

The discussion about the validity and truth of beliefs formulae is vague in BAN, and based on the direct construction of beliefs sets using the annotation rules. Thus, effectively, the beliefs in the BAN sense are semantically meaningless formulae attached to the points of a protocol run, giving no real information about the state of affairs. The intuitive meanings of the beliefs are distinct from the semantics, i.e. the semantics give no base to actually accept the intuitive meanings attached to the beliefs.

In GNY, the semantics given are pretty similar. Basically, a local state of an agent consists of a set of possessed messages P_a and a set of believes B_a, having closure

properties similar to BAN. A global state is a tuple of local states. Given this, a believes φ iff $\varphi \in B_a$ and a has x iff $x \in P_a$. A run is a sequence of global states, requiring monotonicity of message and belief sets as above. The only difference is that we now have explicit rules for including new messages and beliefs in the set of messages and beliefs:

- For a protocol step $a \rightarrow b : x \propto \varphi$, denoting a sending b a message x with the attached meaning φ, occurring between the states s_i and s_{i+1},

  $x \in P_a(s_i) \cap P_b(s_{i+1})$ and $\varphi \in B_a(s_i)$.

Kripke-structure based semantics. AT takes a totally different approach. A local state includes a local history, i.e. the sequence of actions the party has performed, and a set of keys the party possesses. The environment state includes a global history, a set of keys known to the adversaries, and a set of messages sent but not yet delivered. The actions can be send, receive or key generation actions, each having a clearly defined effect on the histories of the performing party and the environment. The set of messages potentially possessed by a party (or the environment) can be determined from the local history and the set of keys known using syntactic (algebraic) concatenation, encryption and decryption rules. This notion is determined using the functions $seen - submsgs_K(x)$ which determine what parts of x can be seen using the keys in the key set K, and $said - submsgs_{K,M}(x)$, which determines the parts of x the sender of x can be kept responsible for, given that it knows the keys in the keyset K and the messages in the message set M.

The set of global states can be seen as points in a multimodal Kripke-structure. To define the reachability relations, Abadi and Tuttle define a function $hide$ that takes a local state of a party, and hides all messages the party cannot understand. Effectively, all messages and parts of messages that cannot be comprehended are replaced by a special symbol $\perp$ denoting an encrypted message that cannot be decrypted.[1] Given the hiding function, two global states s_i and s_j are indistinguishable with respect to a party a iff $hide(s_{a,i}) = hide(s_{a,j})$ where $s_{a,i}$ is the local state of a at the global state s_i.

A disturbing feature of the AT semantics is that the handling of the $hide$ function is not clearly visible at the logic level. That is, there is no notion for expressing that a has received $\{x\}_k$ and *does* comprehend its structure vs. it *does not* comprehend the structure. A rule corresponding to this situation, A11, entitles the party to believe it has received the encrypted message. However, this does not grab the essence of the distinction. Furthermore, it has been later shown that A11 is not sound with respect to the semantics [113].

Towards unification: SvO. The basic model of computation in SvO is very similar to AT. However, there are a few differences. First, in AT the first global state is seen to represent the beginning of the current run, in SvO there may be states before the start of the current run (i.e. state with $t < 0$). This seems also to mean that there is no need for the set of initial keys. Second, SvO makes a difference between receiving a message and seeing it, not available in AT.

The main difference concerns the cryptographic (and algebraic) capabilities of the parties, and how this point of view reflects to the possibility relations within the Kripke-structure. While in all the other logics discussed in this chapter, the parties are thought to have a fixed set of algorithms at their capability, the semantics of SvO includes the possibility of having variation. This may lead to a situation where a party considers possible a world that looks similar to the current one, but where it actually

1. This is somewhat similar to the *-notion of incomprehensible messages used in WK.

would have greater algorithmic capabilities, and thus could comprehend more messages.

Formally, $comprehension(a, (r, t), (r', t'))$ denotes the set of messages the party a is able to comprehend, i.e. fully understand, given the algorithmic capabilities of the point (r, t) and the seen messages of the point (r', t'). Thus, $comprehension(a, (r, t), (r, t))$ denotes the comprehended messages in a run r at time t. Now, the possibility relation $\sim$ (for each party) is defined by

$$(r, t) \sim (r', t') \quad \text{(for } a\text{) iff}$$
$$comprehension(a, (r, t), (r, t)) = comprehension(a, (r, t), (r', t')) \quad .$$

In our opinion, this approach does not make much sense in most practical situations. Furthermore, it complicates semantics and makes the possibility relations non-euclidean.

Other than these, and the fact that SvO can handle a larger set of cryptosystems, the semantics of SvO and AT are almost identical.

Minor differences: WK. The semantics given in [113] are based on SvO. Runs are infinite sequence of states, with states both before the current era and during the current era. The current era (or run) starts at $t = 0$. In addition to sending, receiving and generating messages, a party can *name* a message, i.e. give a new name to a received message. A state $\bar{S}$ is defined to contain the reflexive transitive closure of messages a party can construct from the set of message generated, received or named. This is similar to $seen - submsg$ of AT, and the similar concept of SvO, but handles all messages possessed by a party, not just those that can be generated from a given message. As usual, WK requires that only computable message can be sent, only sent message can be received, and only basic messages (i.e. keys and nonces) can be generated. Furthermore, only messages known to a party, i.e. messages that belong to $\bar{S}$, can be given a new name.

The possibility relations of the Kripke-structure are defined with the help of a function $sight$. This is similar in function to $comprehension$ in SvO, but defined in much more detail. A world (r, t) is considered possible by a party iff it belongs to a set of so called *good worlds*, and if $sight$ applied to the messages of current history is identical to sight applied to the messages in the history of (r, t). In addition to this, WK adds new semantics for the formulae of type a recognizes x and $x_a \equiv y$. These will be discussed in section 6.3.4, “Idealization vs. explicit recognition of messages” on the next page.

6.3.2 Adding time

Adding time to a possible world semantics can be quite straightforward, as can be seen in [106]. The model of computation remains the same, and the temporal aspect is created using either local of global time. However, proving the resulting logic sound can be tedious as there may be surprising connections between temporal and doxatic operators.

Further handling of semantics of time is beyond the scope of this study.

6.3.3 A set of beliefs vs. beliefs based on possible worlds relations

A fundamental difference between the earlier approaches, i.e. BAN and GNY, and the later developments, e.g. AT, SvO and WK, is in the definition and handling of beliefs. In the former case, a belief of a party is just a formula belonging to the set of the party’s beliefs. All beliefs held by a party at a given state are explicitly enumerated.

Whether any of these beliefs is true or not, is not directly apparent from the semantics; the semantics only tell whether a party holds the belief or not. Thus, the beliefs in this sense are operational, and can be seen as bits of information directing the behaviour of the agent.

The latter approach is a nearly complete opposite in some sense. In a possible-worlds based approach, the possibility relationships implicitly define a (potentially infinite) set of beliefs for a party. The reasoning rules and axioms of a logic can be used to infer some of the beliefs — preferably interesting ones. If the logic is sound, all of these beliefs are true if the premises are true. Thus, if a party's belief about an initial state of affairs is reasonable, and it doesn't do any unreasonable assumptions during a protocol run, the set of beliefs it can be inferred to hold remains valid.

It is worth to note that the beliefs handled even in latter the logics are really beliefs, not knowledge. That is, there is no assurance that the set of worlds a party initially considers possible actually contains the real state of affairs. In other words, one or more of a party's initial beliefs may be false. Starting from such an initial state, the real state of affairs may remain outside the set of worlds considered possible during all of the protocol run. For example, if a party, say Alice, initially beliefs that a key k is only known to itself and to single known trusted party, say Bob, i.e. $\text{Alice} \overset{k}{\leftrightarrow} \text{Bob}$, but the key is known to someone else, e.g. Carol, then all beliefs based on the assumption $\text{Alice} \overset{k}{\leftrightarrow} \text{Bob}$ that happens to be false.

Thus, in the first approaches the reasoning rules define how beliefs evolve during a protocol run, and there is no direct semantic support to determine whether the rules are reasonable or not. In the latter approaches the definition of possibility relations implicitly define the belief axioms of the logic, and it is relatively easy to determine whether the axioms are sound with respect to the semantics. In other words, while earlier the designer of a logic had to use one's intuition on designing reasoning rules, a possible worlds based semantics allows the logic designer to concentrate on the base of the beliefs: what a protocol party can determine about its environment and the messages seen, and what it cannot.

One important issue not immediately apparent from the papers referred, is the difficulty of defining the possibility relations. All the possible worlds based semantics use a slightly different construct for this: *hide* in AT, *comprehension* in SvO, or *sight* in WK. The basic idea in all of these is the same: to make a party to consider possible a world which is similar to the current one, but where the real meaning of some incomprehensible message is different from the current state. In other words, a party should not make distinction between the possible states of affairs based on information not available to it. In the case of cryptographic protocols, the lack of information may be due to lack of decryption keys in addition to the usual lack of a proper message received.

6.3.4 Idealization vs. explicit recognition of messages

In BAN, AT and SvO, protocol idealization is a fundamental step in the protocol analysis process. In this process, all or some of the actual protocol messages or their constructs are replaced or augmented with formulae. The logic itself does not make much distinction between messages and logical formulae. GNY and WK take a different approach. In GNY, (some of) the protocol messages are explicitly annotated with formulae, denoting that a message is supposed to communicate that the sender of the message believes in the formula. In WK, all parties have some initial assumptions about the meaning of formulae, i.e. Bob believes that if Alice sends a message m, Bob is entitled to believe that Alice believes in some formula φ.

At the semantic level, the most apparent distinction based on this can be seen in typing. Only GNY and WK define clearly distinct languages for messages and logical formulae. All the rest blur the distinction in a way or another, in the case of AT even to the point that it is possible to believe in some meaningless datum, e.g. a nonce. At the other end, in WK messages and formulae are distinct different sorts that cannot be mixed anywhere.

In GNY, and to a larger extend in WK, the comprehension and recognition of messages is explicitly brought to the language level. In the original BAN approach, this step was only implicitly available within the protocol idealization process. AT and SvO try to be more explicit in this sense, and define the possibility relations on the base of this concept. However, they fail to fully bring this phenomenon to the level of the formal language.

To date, WK is most explicit in this direction. First, it has a number of recognition rules that tell when a protocol party is able to partially comprehend (or verify) the contents of a message (see section C.5.5, “Recognition axioms”, on page 140). In some sense, this corresponds to the Bieber’s notion of partially clear messages. Second, all messages received are *“localized”* before they may be used within beliefs. To put it simply, the message localization process replaces all incomprehensible submessages of a message with a special symbol $*$, denoting an arbitrary bit string. For example, if a message contains both plain text parts and encrypted parts, all the plain text parts are immediately comprehensible, but the encrypted parts are only comprehensible if the party holds the necessary key(s) to decrypt them. Again, this is clearly related to the notion of the explicit cryptoalgebra and free algebra used by Bieber.

6.4 Summary

We have discussed seven approaches to the modelling and analysing cryptographic protocols using epistemic and doxatic logics and explicitly compared five of these approaches. Making a gross simplification, the differences between the five approaches can be summarized as follows:

- BAN and GNY are based on an operational, set based, concept of beliefs. All the rest use Kripke–structure based possible worlds semantics, where the difference in beliefs is due to differences in the definition of the possibility relations.
- BAN, AT and SvO have a protocol idealization step, where the protocol messages are replaced or augmented with logical formulae. GNY and WK rely on explicit message recognition in this respect.
- BAN does not explicitly consider message comprehension. GNY takes a rule based approach. AT and SvO do handle message comprehension at semantic level, but this is not fully utilized at the level of the formal logic. WK make message comprehension explicit, and gives explicit logical rules to handle this. However, its approach may be unnecessarily complicated.

Bieber’s earlier epistemic approach can be used as a valuable background to reflect the other approaches to. Especially Bieber’s notion of partially clear messages corresponds beautifully to message recognition in GNY and WK, and Bieber’s use of two different algebras may give more insight into the message comprehension process.

CHAPTER 7

Future directions

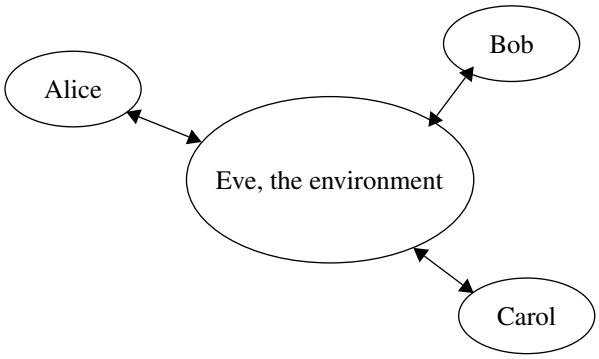


FIGURE 24. Underlying communication model

In this short chapter we outline one possible direction for future work. The basic idea is to combine, at the level of modelling, approaches based on process algebra and modal logic. Using process algebra one can relatively easily build abstract but faithful enough models of the protocol being studied. Giving a suitable mapping, the resulting process graph can be interpreted as a model for a multi modal epistemic logic. Such a logic could easily also encompass temporal operators.

7.1 Process algebras and protocol models

The set of actual messages that can be sent within any protocol is finite, though large. In addition, all well designed protocols have a maximum bound for the number of messages. Therefore, all realistic protocol sequences, or models, are finite. Having the parallel composition operation, process algebra suits pretty well to define such a model. Such a model consists of a number of protocol parties connected to the environment (see Figure 24). The environment models *both* the communication media conveying messages between the protocol parties *and* an adversary capable of deleting, duplicating, modifying and creating messages (cf. e.g. [7]). The protocol parties and the environment can be represented as fairly simple process graphs. Using the parallel composition these graphs can be combined into a single model.

The worst practical problem in this approach lies in the modelling of the computational capabilities of the environment. As the environment represents also the adversaries, this means that the environment is capable of *generating* new messages based on the information it initially has, and that it receives during a protocol run. The approach of having two algebras — a free algebra representing information and another algebra representing the actual bit patterns — seems to be a promising possibility.

The number of *possible* protocol messages (or well-formatted messages) is finite. Therefore it is at least theoretically possible to define a function on the environment states giving the messages the environment is able to generate at a given state. Furthermore, since the computational capabilities are essentially binary: either a message is accepted or it is not, it seems to be possible to represent enough of possible values with a fairly small number of samples. That is, superimposing or folding key and nonce values with certain properties onto representative values allows one to use vastly smaller models than otherwise.

The beauty of the process algebraic approach lies in its relative simplicity. The details of the model are easily understandable, yielding reasonable assurance that the model actually represents the desired properties of the protocol under study.

7.1.1 Action vs event based models

An action based algebraic model is easier than an event based model for a human to understand. The behaviour of the parties and the environment can be studied in isolation and combined. Visualization [112] may give more insight to the behaviour. However, to understand the temporal relationships within a protocol run it is desirable to transform the model into a at least partially event based model. Taking into account the natural real parallelism that is part of the model, a non-interleaving semantics is desirable. Such a transformation seems to naturally lead to concurrent Kripke structures [49] or Chu spaces [50, 93]. These formalisms have the advantage that they support the time-knowledge dualism [92] and seem to be more compact than pure event based approaches. Due to the finiteness of actions and the finiteness of protocol run length, even such an event based model is finite.

7.2 Temporal and modal interpretation

Temporal interpretation. Given an non-interleaving event based model there seems to be a fairly natural temporal interpretation. In such a model, each state may be represented as a set of events occurred so far. That is, given a set of unique events E, a state s is a the set of events that have occurred, $s \subset E$, and the set of possible states S is a subset of the powerset of events, $S \subseteq 2^E$. [49]

Given such a model, the temporal relationship state precedes-in-time $\leq$ can be defined very simply:

$$\text{for all } s_1, s_2 \in S,\ s_1 \leq s_2 \text{ iff } s_1 \subseteq s_2$$

That is, a state precedes in time another state iff the all the events that have occurred in the earlier have also occurred in the later state. The structure $(S, \leq)$ is clearly transitive, reflexive, serial, and antisymmetric, and therefore it is straightforward to define a linear temporal logic for the model.

However, the model is not a fully branching tree, but may contain areas of computation that are temporally unrelated, but have a common future. That is, it is possible

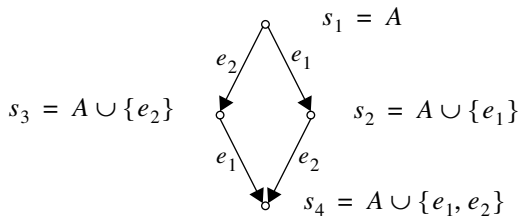


FIGURE 25. Parallel execution of events e_1 and e_2

that there are four states s_1, s_2, s_3, s_4 such that $s_1 \leq s_2$, $s_1 \leq s_3$, $s_2 \not\leq s_3$, $s_3 \not\leq s_2$, but $s_2 \leq s_4$ and $s_3 \leq s_4$. An example about this situation is given in Figure 25 on page 113. It is also possible that the events e_1 and e_2 in the figure are actually more complex processes, or that there are more than two processes executing in parallel, creating an n–dimensional cube. Developing a temporal logic that respects the branching structure of such a model is beyond the scope of this paper.

Epistemic interpretation. From the information theoretical point of view, the set of events a protocol party has participated determines the intensional knowledge the party has. That is, if E_a is the set of events observable by a protocol party a, the set $s_i \cap E_a$ represents the local state of a in a given global state s_i. Now, a relationship R_a defining the possible worlds in the view of a can be defined:

For all states $s_1, s_2 \in S$, $s_1 R_a s_2$ iff $s_1 \cap E_a = s_2 \cap E_a$.

That is, a party considers all such states possible where its local states are similar.

For a non-cryptographic protocol such a definition would suffice for most purposes. However, given the cryptographic setting, the situation gets considerably more complex. It is well possible — or actually natural in cryptographic setting — that there is a set of events $E_i = e_{i_1}, e_{i_2}, \ldots, e_{i_k}$ that all represent a party sending similarly formatted messages at a particular situation. Each of the events represents a different message, carrying different information. However, due to the receiving party not having a necessary cryptographic key, it cannot distinguish between the messages. That is, $(A \cup \{e_{i_i}\}) \cap E_a \neq (A \cup \{e_{i_j}\}) \cap E_a$ but a cannot tell the difference.

A deep difference between the BAN-based advanced authentication logics lies in this very point. (This difference was discussed on a somewhat superficial level in section "Kripke-structure based semantics" on page 107.) The crucial point lies in dividing the event set E_i into disjoint sections, each representing a set of messages the party considered cannot distinguish from each other. Such a division immediately defines the possibility relationship R_a for each party a, leading directly to a well-defined semantic model for a multimodal epistemic logic.

Unfortunately the sectioning of a given event set from the point of view of a protocol party seems to be fairly complex. A promising approach seem to lie in the usage of a separate free message algebra (or term rewriting system) long the way of Dolev and Yao [34, 35], Bieber [14], and Wedel and Kessler [113]. In these approaches, a message is modelled as a composition of basic messages. The parts of the message whose information is not available to a party due to lack of cryptographic keys is turned into a special symbol denoting incomprehensible information. Now, all events that have the same representation in this message algebra are indistinguishable by the protocol party.

It is important to notice that the set of events that a party cannot distinguishes depends on time. That is, if the party later on learns a cryptographic key allowing it to comprehend some more parts of the message, it can refine the sectioning of the event set. Using this new information, it can further revise its view of the possible histories, narrowing the possibility relationship. However, such resectioning is possible only when the party receives a message, thereby revealing some previously unknown key.

To us, making this sectioning approach more explicit seems to be the only really missing piece from the puzzle. However, pursuing this point is beyond the scope of this study.

7.3 Summary

We have briefly suggested that it might be fruitful to consider the modal logic approaches from a process algebraic point of view. The relatively recently developed non-interleaving semantics for process algebras [50] seem to yield relative straightforward temporal and epistemic interpretations of a protocol model in general. However, in the case of cryptographic protocols a further complication is caused by the nature of cryptographic information, forcing one to consider some event sets indistinguishable by a protocol party. However, every such a event sectioning must be reconsidered whenever the party learns more information due to the possibility of the party being able to decrypt some previously incomprehensible part of the already received messages.

CHAPTER 8

Conclusions

In this study we have considered a number of formalisms and their applicability to analysing cryptographic protocols. However, our concern has been more on the level of the security of the whole, distributed system than on the considerations pertaining to a protocol alone. We have studied protocol models based on action semantics, event semantics and knowledge semantics. The tools that have been used are modal logics of knowledge and belief, temporal logics and process algebraic specifications in general.

The focus of the study is on analysing and designing protocols in order to find and prevent protocol failures. We have tried to be independent of the cryptosystems used. This may lead to models that are insecure when used with a particular cryptosystem while being secure if used with another one. These kinds of considerations can be seen as protocol cryptanalysis (cf. e.g. [102]), and are beyond the scope of this study.

The three parts of this work are relatively independent. Part I contains the necessary background information, including an introduction to distributed systems security, cryptology, protocol modelling, cryptanalysis and protocol flaws in Chapter 2. After that, in Chapter 3 the goals of cryptographic protocols are analysed in some depth.

Part II contains introductions to modal logic and process algebra. These introductions are specific in nature, tailored towards the reader interested in analysing cryptographic protocols. This approach is mostly visible in examples and some omissions with respect to general theory in order to simplify the presentation.

Part III contains the real research results. However, their thorough understanding requires the previous introductory material, as well as familiarity with at least some of the actual research papers referred. In Chapter 6 we have compared, both on syntactic and semantic level, the original BAN logic as well as a number of developments based on it. Bieber's logic has been covered, too. In Chapter 7 we have outlined some initial ideas how combining logical approaches and algebraic approaches might further deepen the understanding of the actual behaviour and underlying assumptions of cryptographic protocols.

Based on the work it is clear that there is still quite much to be done in the area of modelling and verification of cryptographic protocols. Both the formalisms themselves need to be developed, and especially the automatic verification tools need

much more improvement. Furthermore, it appears that the area of combining both epistemic (or doxatic) and temporal operators within a multimodal logical framework is an area where even basic research is needed. Unfortunately, in order to be really able to model the intrinsic phenomena involved in protocol failures, such a formal theory would be needed. It might even be that such a theory would bring more light in the inherent connections between modal logics and process algebraic models.

Until recently, the logical approaches in protocol analysis have been more successful. This is natural in the sense that the logical formulae present human thoughts more clearly than algebraic constraints. That is, less experience is needed in order to understand protocol properties when expressed with logical formulae than with algebraic formulae. However, recent development by Kewin Lowe and others at Oxford University, as well as elsewhere, has shown that the automatic verification tools based on process algebra have recently advanced to a level where better results can be achieved than before. This has led to a situation where the current progress based on process algebra (especially CSP) faster than the corresponding progress based on modal logic. Nevertheless, my personal feeling is that be there a similar technical leap at the logical analysers, the situation may well turn back to the favour of modal logic.

Anyhow, the use of the various formalisms has shown that designing and implementation of cryptographic protocols is a very challenging task. Even the application of the current methods, however crude they are, have shown that several protocols designed by leading experts have contained subtle but disastrous design flaws. Given the importance of the Internet and distributed systems in general in the future, both more research and more engineering experience is needed to face the challenges of the future.

APPENDIX A

A protocol example

This appendix describes the lengthy protocol example of Section 2.5, “A protocol example”, on page 26, in full detail. The example system consists of three parties: Alice, Bob and Eve, Eve denoting the combined network and eavesdropper. For simplicity, the range and domain of both encryption and hash functions has been kept unrealistically small. The reason for the security of our example lies in the unrealistically small computational capabilities of the parties.

It is assumed that all the parties are only able to remember three digits in the range {0...9} and nothing else. That makes it impossible for the parties to store more than one earlier protocol run in order to use that information for cryptanalysis. It also makes it impossible for the parties to store the representations of encryption and hash functions in memory as tables. Furthermore, it is assumed that a random numbers in the range (0…9) are unpredictably enough and that it is infeasible to try them all. It is also assumed that the encryption function $\mathrm{enc} : \{0\ldots9\} \rightarrow \{0\ldots9\}$ and hash function $\mathrm{h} : \{0\ldots99\} \rightarrow \{0\ldots9\}$ cannot be cryptanalysed or inverted (not even by exhaustive search). We also suppose that only Alice is able to encrypt anything and only Bob able to decrypt, even though we do not explicitly denote any keys, i.e. that only Alice has the proper encryption key, and that only Bob has the proper decryption key.

These assumptions that we make, being unrealistic, are important. They clearly indicate the kind of assumptions one often implicitly makes when dealing with cryptographic protocols. An unrealistic small example like this sometimes forces one to pay attention to details that might otherwise go unnoticed.

A.1 The protocol

The purpose of the example protocol is to send one digit of information from Alice to Bob without revealing it to Eve. The protocol will function as follows. Let m be the digit to be sent.

1. Alice *generates* a random nonce n
2. Alice calculates $\mathrm{enc}(m)$ and $\mathrm{h}(n, m)$, and *sends* the message $n, \mathrm{enc}(m), \mathrm{h}(n, m)$ to Bob (via Eve).

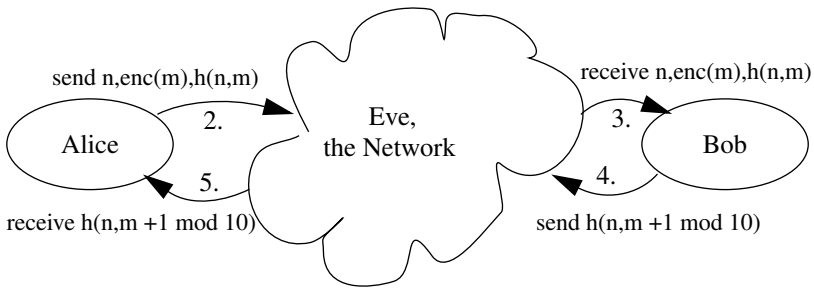


FIGURE 26. An example protocol flow

3. Upon *receiving* the message, Bob decrypts $\mathrm{enc}(m)$ revealing m and calculates $\mathrm{h}(n, m)$ to make sure that the message was not tampered with.
4. To inform Alice that he has got the message, Bob calculates $\mathrm{h}(n, (m+1) \bmod 10)$ and *sends* it to Alice (again via Eve).
5. When Alice *receives* Bob's response, she is able to convince herself that Bob has got the digit.

The flow of messages is illustrated in Figure 26.

A.1.1 Actions

To model the protocol the following actions are defined:

- send(Alice…Bob, 0…9, 0…9, 0…9)

  Alice or Bob sends a message consisting three digits (n , $\mathrm{e}(m)$, and $\mathrm{h}(n, m)$) to the network. (2*10*10*10 = 2000 different actions).
- receive(Alice…Bob, 0…9, 0…9, 0…9)

  Alice or Bob receives a message consisting of three digits.
- send(Alice…Bob, 0..9)

  Alice or Bob sends a digit ($\mathrm{h}(n, (m+1) \bmod 10)$).
- receive(Alice…Bob, 0…9)

  Alice or Bob receives a digit.
- generate(0…9)

  Alice or Bob generates a random number.

Thus, altogether, there are 4420 different possible actions. Therefore there is one run of length zero (the empty run), 4420 runs of length one, 4420 * 4420 = 19536400 runs of length two, and over 10^{18} runs of length five. Restricting the runs considered to those where the actions appear in the right order and are performed by the right party, there are only 10+1000+1000+100+100 = 2210 runs to consider, all of length five. Now, given an m , the digit to transfer, there are only 10 legal runs: one for each possible nonce value. Since there are only 10 possible values for m , there are only 10*10 = 100 legal runs in these sets of 2210 runs with legal action order, or of about 10^{18} total runs. All the rest represent a failure of one kind or another.

To illustrate the protocol in real terms, the functions $\mathrm{h}(x)$ and $\mathrm{enc}(x)$ are given in Tables 12 and 13. Using the tables we can enumerate all the legal protocol runs for

e.g. $m = 0$. If $m = 0$, enc(0) = 5, h(50) = 2, h(51) = 9, …, and h(51) = 4. Thus, the legal runs are given in table Table 14 on page 119.

TABLE 12. Values of encryption function enc(x)

x	0	1	2	3	4	5	6	7	8	9
enc(x)	5	9	8	0	3	1	4	6	7	2

TABLE 13. Values of hash function h(x)

	0_	1_	2_	3_	4_	5_	6_	7_	8_	9_
0	1	1	6	8	2	2	1	8	6	6
1	9	7	0	5	2	9	4	3	1	7
2	8	1	2	5	1	6	6	2	7	5
3	4	9	5	3	9	3	0	5	4	7
4	8	5	7	8	8	5	5	7	4	1
5	5	9	3	6	8	8	1	3	6	4
6	0	5	7	9	9	1	2	4	9	6
7	7	5	1	7	4	9	1	4	0	4
8	1	0	2	0	3	2	9	4	7	6
9	7	3	3	4	6	4	8	5	3	5

TABLE 14. Legal protocol runs for $m = 0$

N	Alice sends	Bob receives	Bob sends	Alice receives
g(0)	s(A, 0, 5, 2)	r(B, 0, 5, 2)	s(B, 9)	r(A, 9)
g(1)	s(A, 1, 5, 9)	r(B, 1, 5, 9)	s(B, 6)	r(A, 6)
g(2)	s(A, 2, 5, 6)	r(B, 2, 5, 6)	s(B, 3)	r(A, 3)
g(3)	s(A, 3, 5, 3)	r(B, 3, 5, 3)	s(B, 5)	r(A, 5)
g(4)	s(A, 4, 5, 5)	r(B, 4, 5, 5)	s(B, 9)	r(A, 9)
g(5)	s(A, 5, 5, 8)	r(B, 5, 5, 8)	s(B, 1)	r(A, 1)
g(6)	s(A, 6, 5, 1)	r(B, 6, 5, 1)	s(B, 8)	r(A, 8)
g(7)	s(A, 7, 5, 9)	r(B 7, 5, 9)	s(B, 2)	r(A, 2)
g(8)	s(A, 8, 5, 2)	r(B, 8, 5, 2)	s(B, 4)	r(A, 4)
g(9)	s(A, 9, 5, 4)	r(B, 9, 5, 4)	s(B, 2)	r(A, 2)

A.2 Run based protocol specification

Given the protocol descriptions, we can formally specify the set of possible actions Σ (the alphabet), which is finite (whose size is 4410 elements), the set of all runs $\Re_{all}$, which is infinite but enumerable, the set of possible runs $\Re$, which is also enumerable, a subset of $\Re$ containing only the runs of length 5 (a finite set), and finally the set of legal runs.

The set of possible actions:

$$\begin{aligned}\Sigma = {} & \{\text{generate}(0), \ldots, \text{generate}(9)\} \cup \\ & \{\text{send}(\text{Alice}, 0, 0, 0), \ldots, \text{send}(\text{Alice}, 0, 0, 9), \\ & \quad \text{send}(\text{Alice}, 0, 1, 0), \ldots, \text{send}(\text{Alice}, 9, 9, 9), \\ & \quad \text{send}(\text{Bob}, 0, 0, 0), \ldots, \text{send}(\text{Bob}, 9, 9, 9)\} \cup \\ & \{\text{receive}(\text{Alice}, 0, 0, 0), \ldots, \text{receive}(\text{Alice}, 9, 9, 9), \\ & \quad \text{receive}(\text{Bob}, 0, 0, 0), \ldots, \text{receive}(\text{Bob}, 9, 9, 9)\} \cup \\ & \{\text{send}(\text{Alice}, 0), \ldots, \text{send}(\text{Bob}, 9)\} \cup \\ & \{\text{receive}(\text{Alice}, 0), \ldots, \text{receive}(\text{Bob}, 9)\}\end{aligned}$$

All runs $\Re_{all}$, possible runs $\Re$, and possible runs of length 5 $\Re_5$:

$$\begin{aligned}\Re_{all} &= \Sigma^* = \varnothing \cup \Sigma \cup \Sigma \times \Sigma \cup \ldots \\ \Re_{all} \supseteq \Re &= \langle s_0, s_1, \ldots, s_n \rangle \qquad n = 0, 1, \ldots \\ & \quad s_k = \text{receive}(a_1, x_1, x_2, x_3) \Rightarrow \exists a_2, s_i, i < k : s_i = \text{send}(a_2, x_1, x_2, x_3) \\ & \quad s_k = \text{receive}(a_1, x_1) \Rightarrow \exists a_2, s_i, i < k : s_i = \text{send}(a_2, x_1) \\ \Re_5 &= \{r \in \Re : |r| = 5\}\end{aligned}$$

The set of successful runs:

$$\begin{aligned}\Re_{succesfull} = \{ & \langle s_0, s_1, \ldots, s_4 \rangle \in \Re_5 : \text{P}(s_0, s_1, \ldots, s_4)\} \\ & \text{P}(s_0, s_1, \ldots, s_4) = \exists n, m : \\ & s_0 = \text{generate}(n) \wedge \\ & s_1 = \text{send}(\text{Alice}, n, \text{enc}(m), \text{h}(n, m)) \wedge \\ & s_2 = \text{receive}(\text{Bob}, n, \text{enc}(m), \text{h}(n, m)) \wedge \\ & s_3 = \text{send}(\text{Bob}, \text{h}(n, (m + 1) \bmod 10)) \wedge \\ & s_4 = \text{receive}(\text{Alice}, \text{h}(n, (m + 1) \bmod 10))\end{aligned}$$

This completes the external, trace based specification.

A.3 Protocol model

Next we develop a protocol model consisting of a local state machine for Alice and Bob. Thereafter we present a (partial) state for Eve or the environment. This state includes all the information Eve is able to gather during a single protocol run. To keep things simple in this example, we assume that none of the parties are able to remember information from earlier protocol runs.

The local state of Alice consists of the message to be sent m_{Alice}, a nonce n_{Alice}, and a state variable s_{Alice} denoting the actions Alice has performed so far. The latter also dictates exactly what Alice is ready to perform next:

$$m_{\text{Alice}} \in \{0, \ldots, 9\}$$
$$n_{\text{Alice}} \in \{\varepsilon, 0, \ldots, 9\} \qquad \text{where } \varepsilon \text{ denotes the empty value}$$
$$s_{\text{Alice}} \in \{\text{init}, \text{generated}, \text{sent}, \text{received}\}$$

The symbols init, generated, sent, received denote the initial state of Alice, the state where Alice has generated the nonce n_{Alice}, the state where Alice has sent the first message and is waiting for a reply fro Bob, and the state where she has received his reply, respectively.

In the same way, Bob's local state consists of m_{Bob}, n_{Bob} and s_{Bob}:

$$m_{\text{Bob}} \in \{\varepsilon, 0, \ldots, 9\}$$
$$n_{\text{Bob}} \in \{\varepsilon, 0, \ldots, 9\}$$
$$s_{\text{Bob}} \in \{\text{init}, \text{received}, \text{sent}\}$$

Using the state variables, we can define the set of legal states for Alice and Bob:

$$S_{\text{Alice}} = \{\langle s, m, n\rangle : s = \text{init} \rightarrow n = \varepsilon, s \neq \text{init} \rightarrow n \neq \varepsilon\}$$
$$S_{\text{Bob}} = \{\langle s, m, n\rangle : s = \text{init} \rightarrow n = m = \varepsilon, s \neq \text{init} \rightarrow n \neq \varepsilon \wedge m \neq \varepsilon\}$$

It is quite straightforward to define the sets of acceptable actions in each state (there are no legal actions in Alice' state received and Bob's state sent):

$$A_{\text{Alice}}(\langle \text{init}, m_a, \varepsilon\rangle) = \{\text{generate}(0), \ldots, \text{generate}(9)\}$$
$$A_{\text{Alice}}(\langle \text{generated}, m_a, n_a\rangle) = \{\text{send}(\text{Alice}, n_a, \text{enc}(m_a), \text{h}(n_a, m_a))\}$$
$$A_{\text{Alice}}(\langle \text{sent}, m_a, n_a\rangle) = \{\text{receive}(\text{Alice}, \text{h}(n_a, (m+1) \bmod 10))\}$$
$$A_{\text{Bob}}(\langle \text{init}, \varepsilon, \varepsilon\rangle) = \{\text{receive}(\text{Bob}, 0, 0, \text{h}(0, \text{dec}(0))), \ldots, \text{receive}(b, 9, 9, \text{h}(9, \text{dec}(9)))\}$$
$$A_{\text{Bob}}(\langle \text{received}, m_b, n_b\rangle) = \{\text{send}(\text{Bob}, \text{h}(n_b, (m_b+1) \bmod 10))\}$$

No other actions are allowed in any state. The states where Alice and Bob enter after each acceptable actions is quite straightforward and not spelled out explicitly.

A successful protocol run is when Alice and Bob both end in their final states.
All other sequences shall lead to a deadlock.

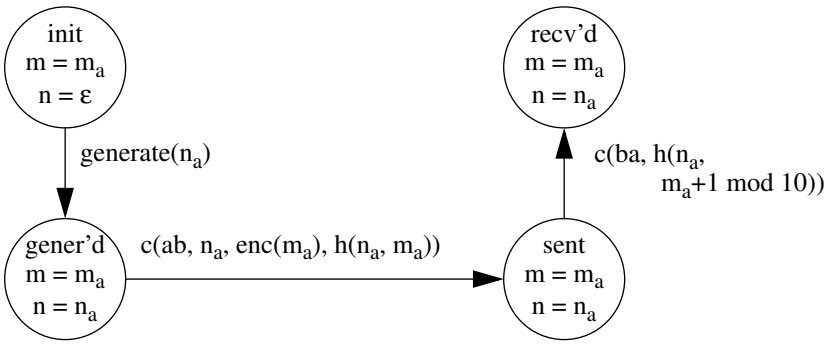


FIGURE 27. An LTS for Alice with states for different m, n combined.

A.4 LTS, ACP and CSP specifications

The individual LTS models for Alice and Bob are given in Figure 27 and Figure 28. Note that we have changed both send(Alice, x) and receive(Bob, x) to c(ab, x) in order to alleviate the need for action renaming before joining the individual models. It is also noteworthy to realize that most "states" in the figures actually denote a number of states (typically 10 or 100, one for each different values of m or m, n. To make this explicit, the first possible states of Alice are given in full in Figure 29 on page 123. Figure 30 on page 123 gives the combined LTS without any explicit network model, i.e. $A|\{\mathrm{c}(ab, x), \mathrm{c}(ba, x)\}|B$.

The behaviour of Alice and Bob can be described as ACP formulae. Table 15 explicitly gives the behaviour of Alice at different states using ACP specifications. Given the definitions in the table, the total behaviour of Alice can be defined as

$$ALICE = \sum_{0 \le m < 10} INIT_m$$

TABLE 15. ACP formulae for Alice' behaviour in different states

State	ACP formula
init	$INIT_m = \sum_{0 \le n < 10} (\mathrm{generate}(n) GENERATED_{m,n})$
generated	$GENERATED_{m,n} = \mathrm{send}(\mathrm{Alice}, n, \mathrm{enc}(m), \mathrm{h}(n, m)) SENT_{m,n}$
sent	$SENT_{m,n} = \mathrm{receive}(\mathrm{Alice}, \mathrm{h}(n, (m+1) \bmod 10)) RECEIVED_{m,n}$
received	$RECEIVED_{m,n} = \varepsilon$

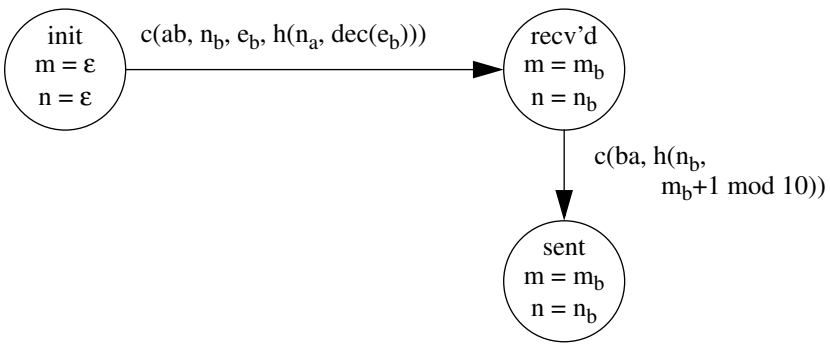


FIGURE 28. An LTS for Bob with states for different m, n combined.

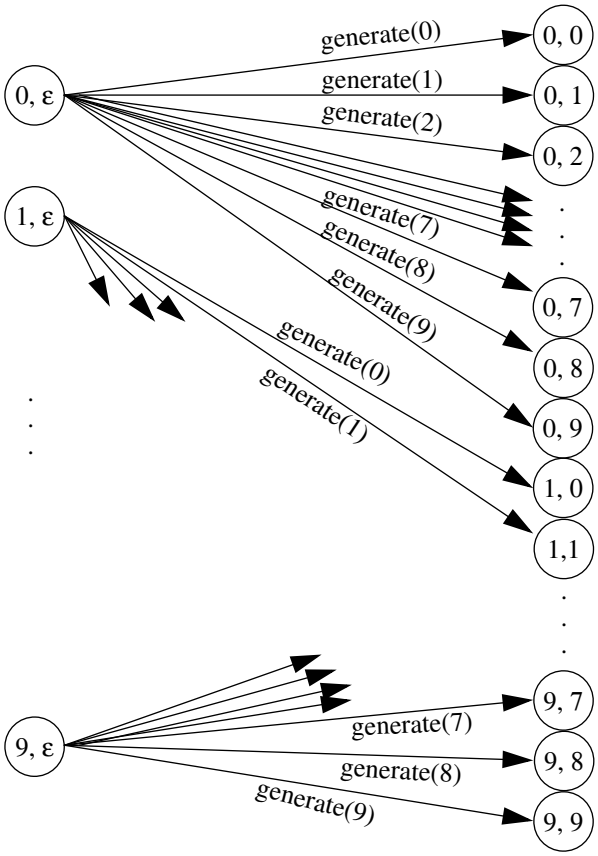


FIGURE 29. Alice' states init and generated in full.

Global protocol states. The combined LTS based protocol model given in Figure 30 is based on synchronous communication and therefore unrealistic. That is, the sending of a message appears *simultaneously* with the receipt of the message. This is unrealistic and does not conform to the environment model described earlier.

In order to produce a more realistic protocol model, we keep the models for Alice and Bob similar, but add a third component, Eve the environment. Eve is always able to receive messages from both Alice and Bob, and it is able to send any of the messages she has received so far to Alice or Bob. However, communication will only happen if a party is ready to send or receive a message. In this sense, communication between Alice and Eve as well as between Bob and Eve is synchronous. The presence and behaviour of Eve makes the communication between Alice and Bob asynchronous. The most important consequence of this is that if Alice has sent a message to Bob, she does not know if he has received it before she got a response back from him.

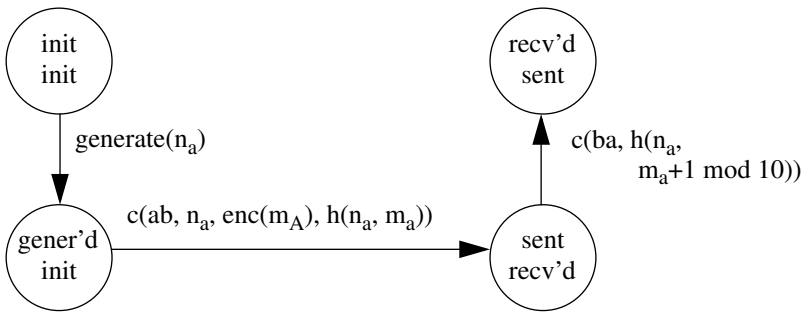


FIGURE 30. The combined LTS representing both Alice' and Bob's actions

Thus, Eve is always able to receive anything, and she is able to send anything she has earlier received. This can be achieved by defining the environment state, or the local state of Eve, as the collection of all messages received by the environment so far. We accomplish this by defining s_e as the sequence of messages received so far. Thus, the set of all possible environment states S_e can be defined as

$$S_e = \Re / \text{generate}(x)$$

or the set of possible protocol runs with all generate actions removed.

The behaviour of Eve can be defined as $\sum EVE_r$ where $r \in \Re$ is a protocol run and EVE_r denotes a state that is able to receive anything and send any of those messages that have been sent during r. Now, if $r = \langle \text{generate}(0), \text{send}(\text{Alice}, 0, 5, 9) \rangle$, EVE_r would be

$$\{\text{receive}(\text{Eve}, 0, 0, 0), \ldots, \text{receive}(\text{Eve}, 9, 9, 9)\} \cup \{\text{send}(\text{Eve}, 0, 5, 9)\}$$

Now we can define a communication function γ which lets Eve to receive anything Alice or Bob sends, i.e.

$$\gamma(\text{send}(a_x, x_1, x_2, x_3), \text{receive}(\text{Eve}, x_1, x_2, x_3)) = \text{communicate}(a_x e, x_1, x_2, x_3)$$

and Alice and Bob to receive anything Eve decides to send.

Given these preliminaries, the behaviour of the overall system can be defined as the ACP formula $ALICE||EVE||BOB$. Analysing this model it can be shown that the only possible successful protocol run is the one intended. In fact, even if we have added the environment model, there is still only 1 (initial) + 10 (generated) + 100 (sent, not received) + 100 (sent and received) + 100 (reply sent, not received) + 100 (reply received) = 411 possible states. The primary reason for this is that Eve cannot generate any new messages but only delay the delivery of sent messages.

The previously described environment model allows the environment to send only messages that it has already learned. However, if we consider a real world situation, an adversary can send absolutely anything to the protocol parties. It relies on the responsibility of the parties to decide whether the message received is acceptable or not. In our toy model, Eve has 1/10 probability of generating a message Bob will accept as a genuine one from Alice, and 1/10 probability of responding to Alice' message in such a way that Alice believes it coming from Bob. In real life, where the ranges and domains of cryptoalgorithms are much larger, these probabilities will be nearly negligible.

Using the alternative environment model where Eve is able to both send and receive anything, the total model of the system becomes much more complex. In addition to the initial action of Alice generating a nonce, Eve can decide to send a message to Bob. Of the 1000 alternatives Eve has, 100 will be accepted by Bob. Thus, the 10 secondary states (where Alice has generated a nonce) is replaced by a union of the legal states and 100 alternative states where Eve has succeeded to send Bob a message that looks legal. In real life the number of states would grow so huge that there would be no possibilities to handle all states explicitly.

Yet another environment model would allow Eve to generate some messages, but in such a way that all these messages would be "illegal". In the case of our toy model, this is easy by deliberately choosing a number of messages where the hash function checking does not succeed. Another direction is to allow Eve to remember messages from previous protocol runs. Using this approach, it can be shown that Eve is able to launch a reply attack against Bob but not against Alice. In other words, Eve is able to

make Bob believe that Bob has received a message from Alice, even though Alice has not sent any messages during the protocol run, but Eve is not able to convince Alice that she has succeeded sending a message to Bob if Bob has not received the message.

This completes our discussion of protocol models in the light of the example. The issues are covered in more detail in chapter “Model checking and Process Algebra” on page 71.

A.5 Knowledge and beliefs

There is one issue left to be discussed using the example: the question of knowledge. The apparent purpose of the protocol seems to be transfer the one digit of information, m, from Alice to Bob. Unfortunately it can be shown that since sufficiently powerful Eve (one that remembers earlier messages) can make Bob believe that he has received a message even when he has not, this primary goal is not succeeded. Instead, it can be shown that after a successful protocol run, Alice knows that Bob knows m, and that Bob indeed does know m, but Bob does not know whether the m he knows was actually sent by Alice during the protocol run, or if it is a reply attack sent by Eve.

The issues of knowledge and beliefs are covered in more detail in Section 4.4, “Logics of knowledge and belief”, on page 55.

APPENDIX B

ISAKMP / Oakley — A real world example

ISAKMP, the Internet Security Association and Key Management Protocol, is a framework for cryptographic protocols developed by the National Security Agency, NSA, and others. The purpose of the framework is to facilitate Internet protocols that create, redefine and detach Internet security associations. Oakley is a Diffie-Hellman based key agreement protocol whose purpose is to create an initial security association between two parties. Oakley is based on the Photuris protocol, originally developed by Phil Karn, and is modified to work within the ISAKMP framework.

As a cryptographic protocol, the ISAKMP + Oakley is relatively complex; however, some of the other real world protocols, e.g. Kerberos, are even more complex. However, both the ISAKMP framework and the Oakley protocol are still under development, and the current specifications lack in detail, contain some mistakes and will most probably change considerably before their final release.

The discussion below is based on the situation in spring 1997. The ISAKMP framework itself begins to be relatively mature, but the adaptation of Oakley to work within the framework requires considerably more work.

B.1 ISAKMP framework

ISAKMP provides a framework for combined authentication, key exchange, and management of security association. ISAKMP itself tries to be free of security policy decisions, cryptographic algorithms, and authentication protocols. In theory, it should be able to include any public or private key authentication and/or key exchange protocol within the framework. In practice, Oakley is currently the only protocol publicly specified to function within the framework.

ISAKMP is intended to function in two phases. Initially the communicating hosts (or other communication entities) representing one or more communicating parties establish an ISAKMP security association. The actual daemon process or other programme actually responsible for the negotiation is called the *negotiation server.* The ISAKMP security association is, in turn, used to protect negotiation of other, protocol specific, security associations. Initially, the only protocol specific security associa-

tions are Internet IPSEC AH (Authentication Header) and ESP (Encapsulated Security Payload) security associations.

The kind of operation performed by the ISAKMP server requires, of course, that the actual communication protocols can trust in the local negotiation server, and that they have a trusted path to communicate with the negotiation server.

To alleviate denial of service (DOS) attacks, ISAKMP uses anti-clogging tokens, or *cookies*, to identify security associations. When a party initiates a negotiation of a security association, it generates a random number, called cookie, and associates it with the association being negotiated. Whenever response packets are received, they are checked to contain a valid cookie. This prevents anyone not able to eavesdrop to send valid-looking packets, thereby alluding a party to reserve resources or perform lengthy cryptographic operations.

All ISAKMP operations appear within an *domain of interpretation* (DOI). The domain of interpretation defines exact packet formats, exchange types, naming conventions and other such details. Initially there are only one public DOIs: the Internet Domain of Interpretation, which is designed for generic Internet traffic between any organizations.

Within a DOI, a situation contains all of the security-relevant information a party needs to make a network level access control decision and to determine the required security services and functions used to protect a session being negotiated. For example, in the Internet DOI, the situation defines a subnet, an individual IP address, a Domain Name Service domain, or an individual user on behalf of which the security association is being established for.

The actual ISAKMP packets consists of a fixed ISAKMP header, followed by one or more explicitly identified payloads. The actual number and type of payloads depends on the exchange being performed, the authentication and key exchange protocol used, and possibly also on the DOI. In any case, the DOI defines the format of some fields within the payloads. Currently there are 12 payloads defined, see Table 16 on page 129.

B.2 Establishing the initial association: base exchange

Typically the initial ISAKMP security association is created using the *base exchange*. It is also possible to use the *identity protection exchange*. The other types of exchanges currently defined include the authentication only exchange, and the Oakley proposal suggest a couple of more. We will only consider the base exchange here.

In the base exchange, the key exchange and authentication of the identity of the parties are combined. However, it transmits the identity of the negotiating parties in clear, as opposed to the identity protection exchange, where the identity of the negotiating party is only released to the peer. The protocol requires four messages: the first two are used to convey the cookies and to negotiate the parameters for the security associations; the second pair is used to establish keying material for the association, and to authenticate the identity of the parties and the contents of the newly established association.

TABLE 16. ISAKMP payload types

ID	Abbr.	Name	Purpose
1	SA	Security Association	Proposes or selects security association parameters
2	P	Proposal	A single proposed set of parameters, used within a SA
3	T	Transform	Parameters for a specific transformation, used with an P
4	KE	Key Exchange	Carries key exchange information, e.g. a DH public component
5	ID	Identification	Identifies a user an association is established for
6	CERT	Certificate	Conveys a certificate associating a user identity to a key. This payload is for convenience only, to alleviate need to communicate with a public key infrastructure.
7	CR	Certificate Request	A request for certificates.
8	HASH	Hash	Contains result of a hash function. Usage of this field is determined by the ISAKMP protocol exchange. Typically used to ensure integrity of a ISAKMP message or association data
9	SIG	Signature	Contains result of a digital signature function. The usage is similar to the hash field, but may be used for non-repudiation purposes also.
10	NONCE	Nonce	Contains random data generated by the sender. Used to ensure freshness of protocol messages.
11	N	Notification	Used to convey error messages and other informational data.
12	D	Delete	Informs the recipient that the sender has removed a security association. Care must be taken to alleviate DOS attacks.

The message flow in the base exchange can be illustrated as follows:

$A \rightarrow B$: HDR, SA, NONCE
$B \rightarrow A$: HDR, SA, NONCE
$A \rightarrow B$: HDR, KE, ID, SIG
$B \rightarrow A$: HDR, KE, ID, SIG

In the diagram, the initiator (Alice) first sends the responder (Bob) an ISAKMP packet containing the header, an security association payload and a nonce. The security association payload defines a situation and one or more proposals for the parameters of the security association. The responder (Bob) replies to this packet with one that selects the proposal of the ones suggested by Alice that Bob considers best from his point of view, and another nonce.

In the third message, Alice sends her identification information, key exchange data and a signature that authenticates this message and the security association parameters. As a reply, Bob sends his identity, his part of key exchange data, and a signature. With this information, both parties can calculate common key material, and authenticate the identity of the peer. The nonces are used to ensure the freshness of the signature and association parameters.

B.3 Using Oakley to establish the initial association

Since the current ISAKMP and Oakley documents do not contain enough of information to unambiguously describe their combined operation, Harkins and Carrel of Cisco systems have given their proposal for resolution. The proposal does not implement the entire Oakley protocol nor the complete ISAKMP framework, but only three Oakley specific exchanges: Oakley *Main Mode*, Oakley *Aggressive Mode*, and Oakley *Quick Mode*. The Oakley main mode exchange roughly corresponds to the ISAKMP identity protection exchange; the aggressive and quick modes are distinct from ISAKMP exchanges. We will only consider the main mode in the this section; the next section will consider Oakley quick mode.

The message flow can be illustrated as follows:

$A \rightarrow B$: HDR, SA
$B \rightarrow A$: HDR, SA
$A \rightarrow B$: HDR, KE, NONCE
$B \rightarrow A$: HDR, KE, NONCE
$A \rightarrow B$: HDR, {ID, SIG}
$B \rightarrow A$: HDR, {ID, SIG}

The first two messages carry the same purpose as in the case of the ISAKMP base exchange: negotiate the security association parameters. In the next two messages, the parties exchange keying material and explicit nonces; including the nonces as explicit information may allow the parties to use the same keying material to communicate with several peers, thereby saving one exponentiation per ISAKMP association. However, the security implications of such a practice are dubious.

With the third pair of messages, the parties authenticate the identity of each other and all the material previously negotiated. The identity information is protected with the keying material negotiated earlier, protecting the identify from passive eavesdroppers.

B.4 Defining an Internet AH/ESP association: Oakley Quick Mode

Establishing the initial ISAKMP association alone does not accomplish anything from the user's point of view. For the user, the forthcoming associations are important. These are negotiated in the protection of the ISAKMP association, allowing a lighter protocol.

When using Oakley, the quick mode may be used to establish a protocol specific security association. In the quick mode, the identities of the client protocol peers are authenticated by the negotiation servers, and the negotiation servers generate new keying material from nonces and their own keying material. This may or may not be acceptable, depending on the level of trust the client protocol parties can place upon the negotiation server.

Schematically, the messages flow as follows:

$A \rightarrow B$: HDR, {HASH, SA, NONCE}
$B \rightarrow A$: HDR, {HASH, SA, NONCE}
$A \rightarrow B$: HDR, {HASH}

In the first message, Alice sends Bob proposals for the new association, a fresh random number, a hash value protecting the nonce from cut & paste attacks, and optionally identities of the client protocol peers. Bob replies with the selected proposal, another random number, a hash calculated over the ISAKMP SA keying material and the new nonces, and the and possibly the same or restricted identities. The third message carries a new hash value calculated over the ISAKMP SA keying material and the nonces, but calculated differently from the previous hash value. This ensures Bob that the exchange is fresh.

The integrity and confidentiality of the messages are protected by the ISAKMP security association. This effectively means that the client protocol parties must consider the negotiation parties to function as trustworthy proxies for their peers. As already noted, this may or may not be appropriate depending on the situation.

APPENDIX C

Rules in the modal approaches

C.1 BAN-logic (Burrows, Abadi, Needham)

C.1.1 Beliefs

$$\frac{a \text{ believes } \varphi \wedge a \text{ believes } \psi}{a \text{ believes } (\varphi \wedge \psi)}$$

$$\frac{a \text{ believes } (\varphi \wedge \psi)}{a \text{ believes } \varphi}$$

$$\frac{a \text{ believes } (b \text{ believes } (\varphi \wedge \psi))}{a \text{ believes } (b \text{ believes } \varphi)}$$

C.1.2 Saying (writing, sending)

$$\frac{a \text{ believes } (b \text{ haswritten } (\varphi \wedge \psi))}{a \text{ believes } (b \text{ haswritten } \varphi)}$$

C.1.3 Seeing (receiving, reading)

$$\frac{a \text{ reads } (\varphi \wedge \psi)}{a \text{ reads } \varphi}$$

$$\frac{a \text{ believes } a \overset{k}{\leftrightarrow} b \wedge a \text{ reads } \{\varphi\}_k}{a \text{ reads } \varphi}$$

$$\frac{a \text{ believes } \overset{k}{\rightarrow} a \wedge a \text{ reads } \{\varphi\}_k}{a \text{ reads } \varphi}$$

$$\frac{a \text{ believes } \overset{k}{\rightarrow} b \wedge a \text{ reads } \{\varphi\}_{k^{-1}}}{a \text{ reads } \varphi}$$

C.1.4 Message authentication

$$\frac{a \text{ believes } a \overset{k}{\leftrightarrow} b \wedge a \text{ reads } \{\varphi\}_k}{a \text{ believes } (b \text{ haswritten } \varphi)}$$

$$\frac{a \text{ believes } \overset{k}{\rightarrow} b \wedge a \text{ reads } \{\varphi\}_{k^{-1}}}{a \text{ believes } (b \text{ haswritten } \varphi)}$$

C.1.5 Freshness

$$\frac{a \text{ believes fresh } \varphi}{a \text{ believes fresh } (\varphi \wedge \psi)}$$

$$\frac{a \text{ believes fresh } \varphi \wedge a \text{ believes } (b \text{ haswritten } \varphi)}{a \text{ believes } (b \text{ believes } \varphi)}$$

C.1.6 Jurisidiction

$$\frac{a \text{ believes } (b \text{ controls } \varphi) \wedge a \text{ believes } (b \text{ believes } \varphi)}{a \text{ believes } \varphi}$$

C.1.7 Key derivation and generation

$$\frac{a \text{ believes } b \overset{k}{\leftrightarrow} c}{a \text{ believes } c \overset{k}{\leftrightarrow} b}$$

$$\frac{d \text{ believes } \left(a \text{ believes } b \overset{k}{\leftrightarrow} c\right)}{d \text{ believes } \left(a \text{ believes } c \overset{k}{\leftrightarrow} b\right)}$$

C.2 GNY logic(Gong, Needham, Yahalom)

C.2.1 Reasoning rules

M If $\vdash \varphi$ then $\vdash a$ believes φ (given differently)

C.2.2 Seeing (reading, receiving) axioms

T1 $$\frac{a \text{ reads } x^*}{a \text{ reads } x}$$

T2 $$\frac{a \text{ reads } \langle x, y \rangle}{a \text{ reads } x}$$

T3, T4, T6 $$\frac{a \text{ reads } \{x\}_k \wedge a \text{ has } \tilde{k}}{a \text{ reads } x}$$ (given separately for symmetric and public keys)

T5 $$\frac{a \text{ reads } f(x, y) \wedge a \text{ has } x}{a \text{ reads } y}$$

C.2.3 Possession axioms

P1 $$\frac{a \text{ reads } x}{a \text{ has } x}$$

P2.1 $$\frac{a \text{ has } x \wedge a \text{ has } y}{a \text{ has } \langle x, y \rangle}$$

P2.2 $$\frac{a \text{ has } x \wedge a \text{ has } y}{a \text{ has } f(x, y)}$$

P3 $$\frac{a \text{ has } \langle x, y \rangle}{a \text{ has } x}$$

P4 $$\frac{a \text{ has } x}{a \text{ has } h(x)}$$

P5 $$\frac{a \text{ has } f(x, y) \wedge a \text{ has } x}{a \text{ has } y}$$

P6.1, P7 $$\frac{a \text{ has } k \wedge a \text{ has } x}{a \text{ has } \{x\}_k}$$

P6.2, P8 $$\frac{a \text{ has } \tilde{k} \wedge P \text{ has } \{x\}_k}{a \text{ has } x}$$

C.2.4 Freshness axioms

F1.1 $$\frac{a \text{ believes fresh } x_i}{a \text{ believes fresh } \langle x_1, \ldots, x_i, \ldots, x_n \rangle}$$

F1.2 $$\frac{a \text{ believes fresh } x}{a \text{ believes fresh } f(x)}$$

F2.1, F3 $$\frac{a \text{ believes fresh } x \wedge a \text{ has } k}{a \text{ believes fresh } \{x\}_k}$$

F2.2, F4 $$\frac{a \text{ believes fresh } \{x\}_k \wedge a \text{ has } \tilde{k}}{a \text{ believes fresh } x}$$

F5, F6 $$\frac{a \text{ believes fresh } k}{a \text{ believes fresh } \tilde{k}}$$

F7.1, F8 $$\frac{(a \text{ believes } (a \text{ recognizes } x)) \wedge (a \text{ believes fresh } k) \wedge (a \text{ has } k)}{a \text{ believes fresh } \{x\}_k}$$

F7.2, F9 $$\frac{(a \text{ believes } (a \text{ recognizes } \{x\}_k)) \wedge (a \text{ believes fresh } k) \wedge (a \text{ has } \tilde{k})}{a \text{ believes fresh } x}$$

F10 $$\frac{a \text{ believes fresh } x \wedge a \text{ has } x}{a \text{ believes fresh } h(x)}$$

F11 $$\frac{a \text{ believes fresh } h(x) \wedge a \text{ has } h(x)}{a \text{ believes fresh } x}$$

C.2.5 Recognition axioms

R1.1 $$\frac{a \text{ believes } a \text{ believes } x_i}{a \text{ believes } a \text{ believes } \langle x_1, \ldots, x_n \rangle}$$

R1.2 $$\frac{a \text{ believes } a \text{ believes } x}{a \text{ believes } a \text{ believes } f(x)}$$

R2.1, R3
$$\frac{a \text{ believes } (a \text{ believes } x) \wedge a \text{ has } k}{a \text{ believes } a \text{ believes } x_k}$$

R2.2, R4
$$\frac{a \text{ believes } a \text{ believes } x_k \wedge a \text{ has } \tilde{k}}{a \text{ believes } a \text{ believes } x}$$

R5
$$\frac{a \text{ believes } a \text{ believes } x \wedge a \text{ has } x}{a \text{ believes } a \text{ believes } h(x)}$$

R6
$$\frac{a \text{ has } h(x)}{a \text{ believes } (a \text{ recognizes } x)}$$

C.2.6 Interpretation axioms

I1.1
$$\frac{\begin{array}{c} a \text{ reads } \{x\}_k^* \wedge a \text{ has } k \wedge a \text{ believes } a \overset{k}{\leftrightarrow} b \\ \wedge a \text{ believes } (a \text{ recognizes } x) \wedge (a \text{ believes fresh } x \vee a \text{ believes fresh } k) \end{array}}{a \text{ believes } (b \text{ haswritten } x)}$$

I1.2
$$\frac{\begin{array}{c} a \text{ reads } \{x\}_k^* \wedge a \text{ has } k \wedge a \text{ believes } a \overset{k}{\leftrightarrow} b \\ \wedge a \text{ believes } (a \text{ recognizes } x) \wedge (a \text{ believes fresh } x \vee a \text{ believes fresh } k) \end{array}}{a \text{ believes } (b \text{ haswritten } \{x\}_k)}$$

I1.3
$$\frac{\begin{array}{c} a \text{ reads } \{x\}_k^* \wedge a \text{ has } k \wedge a \text{ believes } a \overset{k}{\leftrightarrow} b \\ \wedge a \text{ believes } (a \text{ recognizes } x) \wedge (a \text{ believes fresh } x \vee a \text{ believes fresh } k) \end{array}}{a \text{ believes } (b \text{ has } k)}$$

I2 Skipped (concerns shared secrets)

I3.1
$$\frac{\begin{array}{c} a \text{ reads } h(x,k)^* \wedge a \text{ has } \langle x,k \rangle \wedge a \text{ believes } a \overset{k}{\leftrightarrow} b \\ \wedge (a \text{ believes fresh } x \vee a \text{ believes fresh } k) \end{array}}{a \text{ believes } (b \text{ haswritten } \langle x,k \rangle)}$$

I3.2
$$\frac{\begin{array}{c} a \text{ reads } h(x,k)^* \wedge a \text{ has } \langle x,k \rangle \wedge a \text{ believes } a \overset{k}{\leftrightarrow} b \\ \wedge (a \text{ believes fresh } x \vee a \text{ believes fresh } k) \end{array}}{a \text{ believes } (b \text{ haswritten } h(x,k))}$$

I4.1
$$\frac{a \text{ reads } \{x\}_{k^{-1}} \wedge a \text{ has } k \wedge a \text{ believes } \overset{k}{\rightarrow} b \wedge a \text{ believes } (a \text{ recognizes } x)}{a \text{ believes } (b \text{ haswritten } x)}$$

I4.2
$$\frac{a \text{ reads } \{x\}_{k^{-1}} \wedge a \text{ has } k \wedge a \text{ believes } \overset{k}{\rightarrow} b \wedge a \text{ believes } (a \text{ recognizes } x)}{a \text{ believes } (b \text{ haswritten } \{x\}_{k^{-1}})}$$

I5
$$\frac{\begin{array}{c} a \text{ reads } \{x\}_{k^{-1}} \wedge a \text{ has } k \wedge a \text{ believes } \overset{k}{\rightarrow} b \wedge a \text{ believes } (a \text{ recognizes } x) \\ \wedge (a \text{ believes fresh } x \vee a \text{ believes fresh } k) \end{array}}{a \text{ believes } (b \text{ has } \langle k^{-1}, x \rangle)}$$

I6
$$\frac{a \text{ believes } (b \text{ haswritten } x) \wedge a \text{ believes fresh } x}{a \text{ believes } (b \text{ has } x)}$$

I7
$$\frac{a \text{ believes } (b \text{ haswritten } \langle x_1, \ldots, x_n \rangle)}{a \text{ believes } (b \text{ haswritten } x_i)}$$

C.2.7 Jurisdiction axioms

J1 $$\frac{a \text{ believes } (b \text{ controls } \varphi) \wedge a \text{ believes } (b \text{ believes } \varphi)}{a \text{ believes } \varphi}$$

J2 $$\frac{a \text{ believes } honest(b) \wedge a \text{ believes } (b \text{ haswritten } x) \wedge a \text{ believes } conveys(x, \varphi) \wedge a \text{ believes fresh } x}{a \text{ believes } (b \text{ believes } \varphi)}$$

J3 $$\frac{a \text{ believes } honest(b) \wedge a \text{ believes } (b \text{ believes } (b \text{ believes } \varphi))}{a \text{ believes } (b \text{ believes } \varphi)}$$

C.3 AT-logic (Abadi and Tutle)

C.3.1 Reasoning rules

R1 If $\vdash \varphi$ and $\vdash (\varphi \rightarrow \psi)$ then $\vdash \psi$

R2 If $\vdash \varphi$ then $\vdash a$ believes φ

C.3.2 Belief axioms (modalities)

A1 a believes $\varphi \wedge a$ believes $(\varphi \rightarrow \psi) \rightarrow a$ believes ψ

A2 a believes $\varphi \rightarrow a$ believes $(a$ believes $\varphi)$

A3 $\neg(a$ believes $\varphi) \rightarrow a$ believes $\neg(a$ believes $\varphi)$

A4 a believes $\varphi \wedge a$ believes $\psi \leftrightarrow a$ believes $(\varphi \wedge \psi)$

C.3.3 Message authentication axioms

A5 $a \overset{k}{\leftrightarrow} b \wedge c$ reads $\{x\}_k \rightarrow b$ haswritten x (a excluded as sender)

A6 Skipped (concerns shared secrets).

C.3.4 Seeing (reading, receiving)

A7 a reads $\langle x_1, \ldots, x_n \rangle \rightarrow a$ reads x_i

A8 a reads $\{x\}_k \wedge a$ has $k \rightarrow a$ reads x

A9-A10 Skipped (concerns shared secrets and message forwarding).

A11 a reads $\{x\}_k \wedge a$ has $k \rightarrow a$ believes $(a$ reads $\{x\}_k)$ (cf. R2 above)

C.3.5 Saying (writing, sending, meaning)

A12.1 a haswritten $\langle x_1, \ldots, x_n \rangle \rightarrow a$ haswritten x_i

A12.2 a writes $\langle x_1, \ldots, x_n \rangle \rightarrow a$ writes x_i

A13 Skipped (concerns shared secrets).

A14 a haswritten $x \wedge \neg a$ reads $x \rightarrow a$ haswritten x

C.3.6 Jurisdiction

A15 $(a$ controls $\varphi \wedge a$ writes $\varphi) \rightarrow \varphi$

C.3.7 Freshness

A16 $\mathsf{fresh}\ x_i \rightarrow \mathsf{fresh}\ \langle x_1, \ldots, x_n \rangle$

A17 $\mathsf{fresh}\ x \rightarrow \mathsf{fresh}\ \{x\}_K$

A18-A19 Skipped (concers shared secret and message orwarding).

A20 $a\ \mathsf{haswritten}\ x \wedge \mathsf{fresh}\ x \rightarrow a\ \mathsf{writes}\ x$

C.3.8 Key derivation and generation

A21 $a \overset{k}{\leftrightarrow} b \rightarrow b \overset{k}{\leftrightarrow} a$

C.4 SvO-logic (Syverson and van Oorshot)

C.4.1 Reasoning rules

MP If φ and $\varphi \rightarrow \psi$ then ψ

N If $\vdash \varphi$ then $\vdash a\ \mathsf{believes}\ \varphi$

C.4.2 Believing

1 $a\ \mathsf{believes}\ \varphi \wedge a\ \mathsf{believes}\ (\varphi \rightarrow \psi) \rightarrow a\ \mathsf{believes}\ \psi$

2 $a\ \mathsf{believes}\ \varphi \rightarrow a\ \mathsf{believes}\ (a\ \mathsf{believes}\ \varphi)$

C.4.3 Message authentication

3 $a \overset{k}{\leftrightarrow} b \wedge a\ \mathsf{reads}\ \{x\}_k \rightarrow b\ \mathsf{haswritten}\ x$

4 $\overset{k}{\rightarrow} b \wedge a\ \mathsf{reads}\ \{x\}_{k^{-1}} \rightarrow b\ \mathsf{haswritten}\ x$

C.4.4 Key agreement

5 $\alpha \overset{x}{\rightarrow} a \wedge \alpha \overset{y}{\rightarrow} b \rightarrow a \overset{g^{xy}}{\leftrightarrow} b$

C.4.5 Receiving (seeing, reading)

6 $a\ \mathsf{reads}\ \langle x_1, \ldots, x_n \rangle \rightarrow a\ \mathsf{reads}\ x_i$

7 $a\ \mathsf{reads}\ \{x\}_k \wedge a\ \mathsf{has}\ \tilde{k} \rightarrow a\ \mathsf{reads}\ x$

C.4.6 Seeing

8 $a\ \mathsf{reads}\ x \rightarrow P\ \mathsf{sees}\ x$

9 $a\ \mathsf{sees}\ \langle x_1, \ldots, x_n \rangle \rightarrow a\ \mathsf{sees}\ x_i$

10 $a\ \mathsf{sees}\ x_1 \wedge \ldots \wedge a\ \mathsf{sees}\ x_n \rightarrow a\ \mathsf{sees}\ f(x_1, \ldots, x_n)$

C.4.7 Comprehending

11 a believes $(a$ sees $f(x)) \rightarrow a$ believes $(a$ sees $x)$

12 a reads $f(x) \wedge a$ believes $(a$ sees $x) \rightarrow a$ believes $(a$ reads $f(x))$

C.4.8 Saying (writing, sending, meaning)

13 a haswritten $\langle x_1, \ldots, x_n \rangle \rightarrow a$ haswritten $x_i \wedge a$ reads x_i

14 a writes $\langle x_1, \ldots, x_n \rangle \rightarrow a$ haswritten $\langle x_1, \ldots, x_n \rangle \wedge a$ writes x_i

C.4.9 Jurisdiction

15 $(a$ controls $\varphi \wedge a$ writes $\varphi) \rightarrow \varphi$

C.4.10 Freshness

16 fresh $x_i \rightarrow$ fresh $\langle x_1, \ldots, x_n \rangle$

17 fresh $\langle x_1, \ldots, x_n \rangle \rightarrow$ fresh $f(x_1, \ldots, x_n)$

C.4.11 Nonce verification

18 fresh $x \wedge a$ haswritten $x \rightarrow a$ writes x

C.4.12 Goodness of keys

19 $a \overset{k}{\leftrightarrow} b \rightarrow b \overset{k}{\leftrightarrow} a$

C.4.13 Having

20 a has $x \leftrightarrow a$ sees x (x must be a key)

C.5 Wedel-Kessler logic (AUTLOG)

C.5.1 Reasoning rules

MP If φ and $\varphi \rightarrow \psi$ then ψ

M If $\vdash \varphi$ then $\vdash a$ believes φ

C.5.2 Modalities

K a believes $\varphi \wedge a$ believes $(\varphi \rightarrow \psi) \rightarrow a$ believes ψ

4 a believes $\varphi \rightarrow a$ believes $(a$ believes $\varphi)$

5 $\neg(a$ believes $\varphi) \rightarrow a$ believes $\neg(a$ believes $\varphi)$

C.5.3 Jurisdiction axioms

J a controls $\varphi \wedge a$ believes $\varphi \rightarrow \varphi$

C.5.4 Posession axioms

H1 a reads $x \rightarrow a$ has x

H2 a has $x_1 \wedge \ldots \wedge a$ has $x_n \rightarrow a$ has $\langle x_1, \ldots, x_n \rangle$

H3 a has $x \rightarrow a$ has $f(x)$

H4 a has $g^x \wedge a$ has $y \rightarrow a$ has g^{xy}

C.5.5 Recognition axioms

R1 a recognizes $x_i \rightarrow a$ recognizes $\langle x_1, \ldots, x_n \rangle$

R2 a recognizes $x \wedge a$ has $\tilde{k} \rightarrow a$ recognizes $\{x\}_k$

R3 a has $x \rightarrow a$ recognizes $h(x)$

R4 a has $k \wedge a$ has $x \rightarrow a$ recognizes $\{x\}_{k^{-1}}$

R5 a has $x \wedge a$ has $g^y \rightarrow a$ recognizes g^{xy}

C.5.6 Freshness axioms

F1 fresh $x_i \rightarrow$ fresh $\langle x_1, \ldots, x_n \rangle$

F2 fresh $x \rightarrow$ fresh $f(x)$

F3 fresh $x \rightarrow$ fresh g^{xy}

C.5.7 Seeing (receiving, reading)

SE1 a reads $\langle x_1, \ldots, x_n \rangle \rightarrow a$ reads x_i

SE2 a reads $\{x\}_k \wedge a$ has $\tilde{k} \rightarrow a$ reads x

C.5.8 Nonce verification

NV a haswritten $x \wedge$ fresh $x \rightarrow a$ writes x

C.5.9 Saying (sending, writing, meaning)

SA1 a haswritten $\langle x_1, \ldots, x_n \rangle \rightarrow a$ haswritten x_i

SA2 a writes $\langle x_1, \ldots, x_n \rangle \rightarrow a$ writes x_i

SA3 a haswritten $h(x) \wedge \neg(a$ reads $h(x)) \rightarrow a$ haswritten x

SA4 a writes $h(x) \wedge \neg(a$ reads $h(x)) \rightarrow a$ writes x

C.5.10 Authentication and key confirmation axioms

A1 c reads $f(k, x) \wedge a \overset{k}{\leftrightarrow} b \wedge \neg a$ haswritten $f(k, x) \rightarrow b$ haswritten $f(k, x)$

A2 c reads $f(k^{-1}, x) \wedge \overset{k}{\rightarrow} b \rightarrow b$ haswritten $f(k^{-1}, x)$

C.5.11 Comprehension axioms and localization equivalences

C $\quad a \text{ reads } x \wedge x_a \equiv y \rightarrow a \text{ believes } (a \text{ reads } y)$

C1 $\quad a \text{ recognizes } x_i \rightarrow (\langle x_1, \ldots, x_i, \ldots, x_n \rangle_a \equiv \langle (x_1)_a, \ldots, (x_i)_a, \ldots, (x_n)_a \rangle)$

C2 $\quad a \text{ recognizes } x \wedge a \text{ has } k^{-1} \rightarrow (f(k, x))_a \equiv f(k, x_a)$

C3 $\quad a \text{ has } x \rightarrow h(x)_a \equiv h(x_a)$

C4 $\quad a \text{ has } x \wedge a \text{ has } g^y \rightarrow g^{xy}{}_a \equiv g^{xy}$

C5 $\quad a \text{ has } k \wedge a \text{ has } x \rightarrow \sigma(k^{-1}, x)_a \equiv \sigma(k^{-1}, x_a)$

C.5.12 Localization equivalence axioms

E1 $\quad x \equiv x$

E2 $\quad x \equiv y \wedge y \equiv x \rightarrow x \equiv z$

E3 $\quad x \equiv y \rightarrow f(x) \equiv f(y)$

E4 $\quad x_1 \equiv y_1 \wedge \ldots \wedge x_n \equiv y_n \rightarrow \langle x_1, \ldots, x_n \rangle \equiv \langle y_1, \ldots, y_n \rangle$

C.5.13 Key derivation and generation axioms

S $\quad a \overset{k}{\leftrightarrow} b \rightarrow b \overset{k}{\leftrightarrow} a$

KD $\quad a \overset{k}{\leftrightarrow} b \wedge \text{good}_f(x) \rightarrow a \overset{f(k, x)}{\leftrightarrow} b$

KA $\quad \alpha \overset{x}{\rightarrow} a \wedge \alpha \overset{y}{\rightarrow} b \rightarrow a \overset{g^{xy}}{\leftrightarrow} b$

APPENDIX D

References

[1] M. Abadi, R. Needham, *Prudent engineering practice for cryptographic protocols*, Technical Report N:o 125, Digital Equipment Corporation, Systems Research Center, June 1994.

[2] M. Abadi, M. R. Tuttle, "A Semantics for a logic of authentication", *Proceedings of the 10th ACM Symposium on Principles of Distributed Computing*, ACM Press, August 1991, pp. 201-216.

[3] O. Amyay, G. Juanole, S. Zwecker, "An epistemic logic based synthesis of communication services and protocols", *Proceedings of the 12th International Conference on Distributed Computing Systems*, Yokohama, Japan, IEEE Computer Society Press, June 1992, pp. 674-681.

[4] R. J. Anderson, "Why cryptosystems fail", *Proceedings of the 1st ACM Conference on Computer and Communications Security*, ACM Press, 1993, pp. 215-227.

[5] R. J. Anderson, "Liability and computer security: nine principles", *Computer Security--ESORICS'94*, Springer-Verlag, 1994, pp. 231-245.

[6] R. Atkinson, *Security architecture for the internet protocol*, RFC1825, Internet Engineering Task Force, 1995.

[7] T. Aura, *Modelling the Needham-Schroder authentication protocol with high level Petri nets*, Technical Report B14, Digital Systems Laboratory, Helsinki University of Technology, Septemper 1995.

[8] J. C. M. Baeten, W. Weijland, *Process Algebra*, Cambridge Tracts in Theoretical Computer Science 18. , Cambridge, UK, 1990.

[9] M. Barr, C. Wells, *Category theory of computing science, 2nd edition*, Prentice Hall International, 1995.

[10] D. E. Bell, L. J. LaPadula, *Secure computer system: unified exposition and multics interpretation*, Technical Report N:o MTR-2997ESD-TR-75-306, MITRE Corp, Bedford, Ma, March 1976.

[11] S. M. Bellowin, "Security problem in the TCP/IP protocol suite", *ACM Computer Communication Review*, 19(2), April 1989, pp. 32-48.

[12] J. van Benthem, J. van Eijck, V. Stebletsova, "Modal Logic, Transition Systems and Processes", *Journal of Computational Logic* , 4(5), Oxford University Press, 1994, pp. 1-50.

[13] J. A. Bergstra, J. W. Klop, “Process Algebra for Synchronous Communication”, *Information and Control*, 1984, pp. 109-137.

[14] P. Bieber, “A logic of communication in a hostile environment”, *Proceedings of IEEE Computer Security Foundations Workshop III*, Los Alamitos, CA, IEEE Computer Society Press, 12-14 June 1990, pp. 14-22.

[15] M. Blaze, J. Feigmenbaum, J. Lacy, “Decentralized trust management”, *Proceedings of the 1996 IEEE Computer Society Symposium on Research in Security and Privacy*, Oakland, CA, May 1996.

[16] C. A. Boyd, “A formal framework for authentication”, *Computer Security--ESORICS’92*, Springer-Verlag, 1992, pp. 273-292.

[17] C. A. Boyd, “Security architectures using formal methods”, *IEEE Journal on Selected Areas in Communications*, 11(5), June 1993, pp. 694-701.

[18] C. A. Boyd, W. Mao, “On a limitation of BAN logic”, *Advances in Cryptology--EUROCRYPT’93 Proceedings*, Springer-Verlag, 24-26 May 1993, pp. 240-247.

[19] C. A. Boyd, W. Mao, “Designing secure key exchange protocols”, *Computer Security--ESORICS’94*, Springer-Verlag, 1994, pp. 93-106.

[20] M. Burrows, M. Abadi, R. Needham, *A logic of authentication*, Technical Report N:o 39, Digital Equipment Corporation, Systems Research Center, 130 Lytton Avenue, Palo Alto, CA 94301, 22 February 1989.

[21] M. Burrows, M. Abadi, R. Needham, “Rejoinder to Nessett”, *Operating Systems Review*, 24(2), April 1990, pp. 39-40.

[22] M. Burrows, M. Abadi, R. Needham, *The scope of a logic of Authentication*, Technical Report N:o 39-appendix, DEC Systems Research Center, 1994.

[23] U. Carlsen, “Cryptographic protocol flaws”, *Proceedings of IEEE Computer Security Foundations Workshop VII*, Franconia, New Hampshire, IEEE Computer Society Press, 14-16 June 1994, pp. 192-200.

[24] Chellas, *Modal logic: an introduction*, Cambridge University Press, Cambridge, UK, 1980.

[25] D. Clark, D. Wilson, “A comparison of commercial and military security policies”, *Proceedings of the 1987 IEEE Symposium on Security and Privacy*, IEEE Computer Society Press, April 1987, pp. 184-194.

[26] J. Clark, J. Jacob, *Security protocols: an annotated bibliography*, Department of Computer Science, University of York, YORK, YO1 5DD, England.

[27] E. M. Clarke, E. A. Emerson, A. P. Sistla, “Automatic verification of finite-state concurrent systems using temporal logic specifications”, *ACM Transactions on Programming Languages and Systems*, 8(2), April 1986, pp. 244-263.

[28] D. E. Denning, G. M. Sacco, “Timestamps in key distribution protocols”, *Communications of the ACM*, 24(8), August 1981, pp. 533-536.

[29] D. E. Denning, *Cryptography and data security*, Addison-Wesley, Reading, MA, 1982.

[30] W. Diffie, M. E. Hellman, “New directions in cryptography”, *IEEE Transactions on Information Theory*, November 1976, pp. 644-654.

[31] W. Diffie, M. E. Hellman, “Privacy and authentication: an introduction to cryptography”, *Proceedings of IEEE*, 67(3), 1979.

[32] W. Diffie, “The first ten years in public key cryptography”, *Proceedings of IEEE*, 76(5), May 1988, pp. 560-577.

[33] W. Diffie, P. C. van Oorschot, M. J. Wiener, “Authentication and authenticated key exchanges”, *Design, Codes, and Cryptography*, Kluwer Academic Publisher, 1992, pp. 107-125.

[34] D. Dolev, A. C. Yao, “On the security of public key protocols (Extended Abstract)”, *Proceedings of the 22nd IEEE Symposium on Foundations in Computer Science*, Nashville, Tennessee, 28--30 October 1993, pp. 350-357.

[35] D. Dolev, A. Yao, “On the security of public-key protocols”, *IEEE Transactions on Information Theory*, IT-29(2), March 1983, pp. 198-208.

[36] C. Ellison, B. Frantz, B. M. Thomas, *Simple Public Key Certificate*, Internet Draft draft-ietf-spki-cert-structure-01.txt (Work in progress), 25 March 1997.

[37] R. Fagin, J. Y. Halpern, Y. Moses, M. Y. Vardi, *Reasoning about knowledge*, MIT Press, Cambirdge, MA, 1995, 477 p..

[38] Formal Systems Design, *Failures divergence refinement: A verification tool for finite state systems*, Formal Systems Design and Development Inc, Auburn, AL, pp. 2.

[39] R. van Glabbeek, *What is branching time and why to use it?*, Technical Report N:o STAN-CS-93-1486, Stanford University, .

[40] R. J. van Glabbeek, *History Preserving Process Graphs*, Unpublished manuscript, Stanford University, 1995.

[41] R. J. van Glabbeek, *Comparative concurrencey semantics and refinement of actions*, Centrum voor Wiskunde en Informatica, 1996.

[42] I. Goldberg, D. Wagner, “Randomness and the Netscape Browser”, *Dr. Dobb’s Journal*, Miller Freeman Inc., January 1996.

[43] R. Goldblatt, *Logics of time and computation*, CSLI Publications, Stanford, 1992.

[44] L. Gong, “Using one-way function for authentication”, *Computer Communication Review*, 19(5), July 1989, pp. 8-11.

[45] L. Gong, *Cryptographic protocols for distributed systems*, Ph.D. Thesis, University of Cambridge, April 1990.

[46] L. Gong, R. Needham, R. Yahalom, “Reasoning about belief in cryptographic protocols”, *Proceedings of the 1990 IEEE Computer Society Symposium on Research in Security and Privacy*, Oakland, CA, IEEE Computer Society Press, 7-9 May 1990, pp. 234-248.

[47] L. Gong, “Handling infeasible specifications of cryptographic protocols”, *Proceedings of IEEE Computer Security Foundations Workshop IV*, IEEE Computer Society Press, June 1991, pp. 99-102.

[48] L. Gong, “A security risk of depending on synchronized clocks”, *ACM Operating System Review*, 26(1), January 1992, pp. 49-53.

[49] V. Gupta, “Concurrent Kripke structures”, *Proceedings of North American Process Algebra Workshop*, Cornell University, August 1993.

[50] V. Gupta, *Chu Spaces: A Model of Concurrency*, Ph.D. Thesis, Stanford University, Septemper 1994.

[51] J. Y. Halpern, *Reasoning about knowledge*, Kaufmann, 1986.

[52] N. Heintze, J. D. Tygar, “A model for secure protocols and their compositions”, *Proceedings of the 1994 IEEE Computer Society Symposium on Research in Security and Privacy*, IEEE Computer Society Press, May 1994, pp. 2-13.

[53] J. Hintikka, *Knowledge and belief*, Cornell University Press, 1962.

[54] C. A. R. Hoare, “Communicating sequential processes”, *Communications of the ACM*, 21(8), August 1978, pp. 666-672.

[55] C. A. R. Hoare, *A model for communicating sequential processes*, Campridge University Press, 1980, pp. 229-243.

[56] C. A. R. Hoare, *Communicating sequential processes*, Prentice-Hall, 1985, 256 p..

[57] R. Housley, W. Ford, W. Polk, D. Solo, *Internet Public Key Infrastructure: Part I: X.509 Certificat and CRL Profile*, Internet Draft draft-ietf-pkix-ipki-part1-04.txt (Work in progress), 26 March 1996, 75 p..

[58] ISO 7498-2:1989, *OSI Basic Reference Model -- Part 2: Security Architecture*, International Organization for Standardization, 1989.

[59] J. Jacob, "Security specifications", *Proceedings of the 1988 IEEE Symposium on Security and Privacy*, Oakland, CA, IEEE Computer Society Press, April 18-21 1988, pp. 14-23.

[60] R. Kailar, V. D. Gilgor, "On belief evolution in authentication protocols", *Proceedings of the 1991 IEEE Computer Society Symposium on Research in Security and Privacy* , IEEE Computer Society Press, June 1991, pp. 103-116.

[61] A. T. Karila, *Open Systems Security -- an Architectural Framework*, Ph.D. Thesis, Helsinki University of Technology, 1991.

[62] R. A. Kemmerer, C. A. Meadows, J. K. Millen, "Three systems for cryptographic protocol analysis", *Journal of Cryptology*, 7(2), 1994, pp. 79-130.

[63] V. Kessler, G. Wedel, "AUTLOG--an advanced logic of authentication", *Proceedings of IEEE Computer Security Foundations Workshop VII*, IEEE Computer Society Press, 1994, pp. 90-99.

[64] P. Kijser, T. Parker, D. Pinkas, "SESAME: the solution to security for open distributed systems", *Journal of Computer Communications*, 17(4), July 1994, pp. 501-518.

[65] B. Lampson, M. Abadi, M. Burrows, E. Wobber, "Authentication in distributed systems: theory and practice", *Proceedings of the 13th ACM Symposium on Operating System Principles*, ACM Press, October 1991.

[66] A. Liebl, "Authentication in distributed systems: a bibliography", *Operating Systems Review*, 27(4), October 1993, pp. 122-136.

[67] G. Lowe, "An attack on the Needham-Schroeder public key authentication protocol", *Information Processing Letters*, 56(3), November 1995, pp. 131-136.

[68] G. Lowe, "Breaking and fixing the Needham-Schroeder public-key protocol using CSP and FDR", *Proceedings of the 2nd International Workshop on Tools and Algorithms for the Construction and Analysis of Systems*, Passau, Germany, 27-29 March 1996.

[69] W. Mao, C. A. Boyd, "Towards formal analysis of security protocols", *Proceedings of IEEE Computer Security Foundations Workshop VI*, IEEE Computer Society Press, June 1993, pp. 147-158.

[70] W. Mao, C. A. Boyd, "On the use of encryption in cryptographic protocols", *Codes and Cyphers*, 1993, pp. 251-262.

[71] W. Mao, C. A. Boyd, "Development of authentication protocols: some misconceptions and a new approach", *Proceedings of IEEE Computer Security Foundations Workshop VII*, IEEE Computer Society Press, 1994, pp. 178-186.

[72] D. Maughan, M. Schertler, M. Schneider, J. Turner, *Internet security association and key management protocol (ISAKMP)*, Internet Draft draft-ietf-ipsec-isakmp-07.txt, .ps (Work in progress), Internet Engineering Task Force, 21 February 1997, 79 p..

[73] P. McMahon, "SESAME V2 public key and authorization extensions to Kerberos", *Proceedings of the 1995 Network and Distributed Systems Security*, SNDSS'95, IEEE Computer Society Press, 1995, pp. 114-131.

[74] C. A. Meadows, "A system for the specification and analysis of key management protocols", *Proceedings of the 1991 IEEE Computer Society Symposium on Research in Security and Privacy*, Los Alamitos, CA, IEEE Computer Society Press, 1991, pp. 182-195.

[75] C. A. Meadows, "The NRL protocol analyzer: an overview", *2nd Conference on the Practical Applications of Logic Programming*, 1994.

[76] C. A. Meadows, "A model of computation for the NRL protocol analyzer", *Proceedings of IEEE Computer Security Foundations Workshop VII*, Franconia, New Hampshire, IEEE Computer Society Press, 14-16 June 1994.

[77] C. A. Meadows, "Formal verification of cryptographic protocols: a survey", *Advances in Cryptology--ASIACRYPT'94 Proceedings*, Springer-Verlag, 1995, pp. 133-150.

[78] J. K. Millen, S. C. Clark, S. B. Freedman, "The Interrogator: protocol security analysis", *IEEE Transactions on Software Engineering*, SE-13(2), February 1987, pp. 274-288.

[79] R. Milner, *A Calculus of Communicating Systems*, Springer-Verlag, 1980.

[80] R. Milner, *Communication and Concurrency*, Prentice-Hall, 1989.

[81] R. Molva, G. Tsudik, E. van Herreweghen, S. Zatti, "KryptoKnight authentication and key distribution system", *Computer Security--ESORICS'92*, ESORICS'92, 1993.

[82] R. Needham, M. D. Schroeder, "Using encryption for authentication in large networks of computers", *Communications of the ACM*, 21(12), December 1978, pp. 993-999.

[83] R. Needham, M. D. Schroeder, "Authentication revisited", *Operating Systems Review*, 21(7), January 1987.

[84] D. M. Nessett, "A critique of the Burrows, Abadi and Needham logic", *Operating Systems Review*, 24(2), April 1990, pp. 35-38.

[85] B. C. Neuman, "Proxy-based authorization and accounting for distributed systems", *Proceedings of the 13th International Conference on Distributed Computing Systems*, May 1993, pp. 283-291.

[86] Object Management Group, *CORBAservices: Common Object Services Specification, Revised Edition*, Object Management Group, Farmingham, MA, March 1997.

[87] P. C. van Oorschot, "Extending cryptographic logics of belief to key agreement protocols", *Proceedings of the 1st ACM Conference on Computer and Communications Security*, Fairfax, Virginia, 3-5 November 1993, pp. 232-243.

[88] H. K. Orman, *The Oakley key determination protocol*, Internet Draft draft-ietf-ipsec-oakley-01.txt (Work in progress), Internet Engineering Task Force, 1996.

[89] D. Otway, O. Rees, "Efficient and timely mutual authentication", *Operating Systems Review*, 21(1), 1987, pp. 8-10.

[90] C. A. Petri, "Fundamentals of a theory of asynchronous information flow", *Proceedings of IFIP Congress 62*, Munich, North-Holland, Amsterdam, 1962, pp. 386-390.

[91] B. Pfitzmann, *Digital Signature Schemes: General Framework and Fail-Stop Signatures*, Springer-Verlag, Heidelberg, 1996.

[92] V. R. Pratt, "Time and Information in Sequential and Concurrent Computation", *Proceedings of Theory and Practice of Parallel Programming (TPPP'94)*, Sendai, Japan, November 1994, pp. 1-24.

[93] V. R. Pratt, *Chu Spaces and their Interpretation as Concurrent Objects*, Springer-Verlag, 1995, pp. 392-405.

[94] R. L. Rivest, A. Shamir, L. M. Adleman, “A method for obtaining digital signatures and public key cryptosystems”, *Communications of the ACM*, 1978, pp. 120-126.

[95] R. L. Rivest, B. Lampson, “SDSI -- a simple distributed security infrastructure”, *Proceedings of the 1996 Usenix Security Symposium*, 1996.

[96] A. W. Roscoe, “Modelling and verifying key-exchange protocols using CSP and FDR”, *Proceedings of the 8th IEEE Computer Security Foundations Workshop*, Ireland, 13-15 June 1995.

[97] A. D. Rubin, P. Honeyman, *Formal methods for the analysis of authentication protocols*, Technical Report N:o 93-7, Center for Information Technology Integration, Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI, 8 November 1993.

[98] S. A. Schneider, “Security properties and CSP”, *Proceedings of the 1996 IEEE Computer Society Symposium on Research in Security and Privacy*, Oakland, CA, May 1996.

[99] B. Schneier, *Applied cryptography -- protocols, algorithms and source code in C*, John Wiley & Sons, Inc., 1996, 758 p..

[100] M. Setala, A. Valmari, “Validation and verification with weak process semantics”, *Proceedings of Nordic Seminar on Dependable Computing Systems 1994*, Lyngby, Denmark, August 1994, pp. 15-26.

[101] C. E. Shannon, “Communication theory of secrecy systems”, *Bell Systems Technical Journal*, 1949, pp. 656-715.

[102] G. J. Simmons, “Cryptanalysis and protocol failures”, *Communications of the ACM*, 37(11), November 1994, pp. 56-65.

[103] E. Snekkenes, “Exploring the BAN approach to protocol analysis”, *Proceedings of the 1991 IEEE Computer Society Symposium on Research in Security and Privacy*, 1991, pp. 171-181.

[104] E. Snekkenes, “Roles in cryptographic protocols”, *Proceedings of the 1992 IEEE Computer Society Symposium on Research in Security and Privacy*, Los Alamitos, CA, IEEE Computer Society Press, May 1992, pp. 105-119.

[105] J. G. Steiner, B. C. Neuman, J. I. Schiller, “Kerberos: an authentication service for open network systems”, *Proceedings of Winter 1988 USENIX Conference*, Dallas, Texas, USENIX Association, February 1988, pp. 191-202.

[106] P. Syverson, “Adding time to a logic authentication”, *Proceedings of the 1st ACM Conference on Computer and Communications Security*, Fairvax, VA, ACM Press, 3-5 November 1993, pp. 97-101.

[107] P. Syverson, C. A. Meadows, “Formal requirements for key distribution protocols”, *Advances in Cryptology--EUROCRYPT’94 Proceedings*, Springer-Verlag, 1995.

[108] P. Syverson, P. C. van Oorschot, “On unifying some cryptographic protocol logics”, *Proceedings of the 1994 IEEE Computer Society Symposium on Research in Security and Privacy*, May 1994, pp. 14-28.

[109] M. Tienari, *Formal Specification of Computer Communication Protocols*, Technical Report N:o D363, Department of Computer Science, University of Helsinki, 21 August 1995.

[110] A. Valmari, M. Tienari, “An Improved Failures Equivalence for Finite-State Systems with a Reduction Algorithm”, *Proceedings of Protocol Specification, Testing and Verification XI*, North-Holland, 1991, pp. 3-18.

[111] A. Valmari, “Failure-based equivalences are faster than many believe”, *Proceedings of Structures in Concurrency Theory (STRICT) 95*, 1995.

[112] A. Valmari, M. Setala, *Visual verification of safety and liveness*, Software Systems Laboratory Report, Tampere University of Technology, 1995, 20 p..

[113] G. Wedel, V. Kessler, “Formal semantics for authentication logics”, *Computer Security -- ESORICS’96*, Rome, Italy, Springer-Verlag, Septemper 1996, pp. 219-241.

[114] T. Y. C. Woo, S. S. Lam, “A semantic model for authentication protocols”, *Proceedings of the 1993 IEEE Computer Society Symposium on Research in Security and Privacy*, May 1993, pp. 178-194.