INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

UMI

A Bell & Howell Information Company
300 North Zeeb Road, Ann Arbor MI 48106-1346 USA
313/761-4700 800/521-0600

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

An Adversary-Centric Logic of Security and Authenticity

by

Scott David Dexter

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Computer Science and Engineering)
in The University of Michigan
1998

Doctoral Committee:
Associate Professor Kevin J. Compton, Chair
Professor Andreas Blass
Professor Yuri Gurevich
Adjunct Professor Peter Honeyman
Associate Professor Atul Prakash

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

UMI Number: 9840522

UMI Microform 9840522
Copyright 1998, by UMI Company. All rights reserved.

This microform edition is protected against unauthorized copying under Title 17, United States Code.

UMI
300 North Zeeb Road
Ann Arbor, MI 48103

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

ACKNOWLEDGEMENTS

The success of a project like this is possible only with the support of many friends and colleagues . . . .

Thanks to Kevin Compton, for striking out in a new direction with me, for teaching by example the methodology of research, and for giving so generously of his time and insight.

Thanks to Yuri Gurevich, for introducing me to the concept of mathematical maturity.

Thanks to Peter Honeyman, for summer research support and his enthusiasm for broadening my areas of inquiry.

Thanks to Ann Ford, for her mentorship in the art of teaching and for her support and friendship during moments of difficulty.

Thanks to my fellow students in the department, for their willingness to exchange ideas about the graduate student experience and to talk about things outside that experience; especially Phil Haar, Patrick Doyle, Jon Atkins, Chuck Wallace, Paul Jensen, Viviane Crestana, Eric Klavins, and Jim Eng.

Thanks to my fellow workers in the Graduate Employees Organization, for helping me develop an analysis of the academy and the world and for nurturing the talents they insisted on seeing in me; especially Tamara Joseph, Rachel Barish, Michelle Mueller, and Eric Fink.

Thanks to my friends outside the department, for always being willing to complain and hear complaints and for sharing with me a much larger perspective; especially Alex Marchevsky, Jason Elias, Brenda Cárdenas, and Graciela Hernandez.

Thanks to my parents, for making getting to this point seem natural but never necessary.

And finally, to Jeanne, whose support, insight, intellectual rigor, and companionship have sustained me along the way; she has shown me that in all endeavors we make the road only by walking.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS ii

LIST OF FIGURES v

CHAPTERS

1 The Need for Formal Analysis 1
- 1.1 Approaches to Formal Analysis 2
  - 1.1.1 Logics of Knowledge and Belief 3
  - 1.1.2 Algebraic Approaches 6
  - 1.1.3 Logic-Based Approaches 10
  - 1.1.4 Complexity-Theoretic Approaches 12

2 Informal Description of the Method 15
- 2.1 Protocols and the Environment 15
  - 2.1.1 Agents 15
  - 2.1.2 Messages 16
- 2.2 The Adversary 17
- 2.3 Defining Attack 18
- 2.4 Modeling Protocols 21
  - 2.4.1 Example: Modeling the Needham-Schroeder Protocol 22
- 2.5 Analyzing Protocols 23
  - 2.5.1 Example: Analyzing the Needham-Schroeder Protocol . . . 25
- 2.6 Timestamps 25

3 A Formal Semantics 29
- 3.1 A Model of Messages 29
- 3.2 Linear Logic 30
  - 3.2.1 Introduction to Linear Logic 31
  - 3.2.2 Logic Programming for Linear and Affine Logic 33
  - 3.2.3 Affine Logic and Protocol Analysis 35
- 3.3 Toward Resolution in Affine Logic 37
  - 3.3.1 Resolution 38
  - 3.3.2 *LLH*-Soundness 39
  - 3.3.3 *LLH*-Completeness 39
- 3.4 An Intended Model 42

4 Implementation and Results 45

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

4.1 Implementation 45
4.1.1 The Preprocessor 45
4.1.2 The Proof Generator 46
4.2 Results 48
4.2.1 Needham-Schroeder 48
4.2.2 Otway-Rees 50
4.2.3 "Optimized" Otway-Rees 50
4.2.4 Bellare-Rogaway 3PKD 52
4.2.5 Yahalom 53
4.2.6 Wide-Mouthed Frog 53
4.2.7 Modified Wide-Mouthed Frog 54
4.2.8 Kerberos 55

5 Discussion and Future Work 58
5.1 Attack Defined 58
5.2 Formal Semantics 58
5.3 Automation 59
5.4 Intuition 59
5.5 Future Work 59

BIBLIOGRAPHY 61

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

LIST OF FIGURES

Figure

2.1 The Needham-Schroeder protocol 22
2.2 Message-structure rules for the Needham-Schroeder protocol 23
2.3 Needham-Schroeder protocol-specific inference rules 24
2.4 Partial analysis of the Needham-Schroeder protocol 26
2.5 The Wide-Mouthed Frog protocol 27
2.6 Wide-Mouthed Frog protocol-specific inference rules 27
3.1 Message-structure rules 30
3.2 Linear logic proof system LL 34
3.3 A relevant fragment of affine logic 37
3.4 Resolution rules for restricted linear logic 38
4.1 Specification of the Needham-Schroeder protocol 46
4.2 The Needham-Schroeder proof tree for Q_i 49
4.3 The Needham-Schroeder protocol 50
4.4 The Otway-Rees protocol 50
4.5 The "optimized" Otway-Rees protocol 50
4.6 Analysis of Otway-Rees protocol 51
4.7 The Bellare-Rogaway 3PKD protocol 52
4.8 The Yahalom protocol 53
4.9 The Wide-Mouthed Frog protocol 54
4.10 Analysis of the Wide-Mouthed Frog protocol 54
4.11 The Modified Wide-Mouthed Frog protocol 54
4.12 The Kerberos protocol 55

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

CHAPTER 1

The Need for Formal Analysis

The growing prevalence of distributed computing is driving a demand for security and privacy in distributed communication. While informal understandings of "security and privacy" abound, the task of provably ensuring that these properties hold necessitates a more systematic approach. Much work has been done in the mathematical foundations of cryptography; methods of combining these basic elements into a particular application are often subtle and are not yet well understood.

In their 1978 paper, Needham and Schroeder [NS78] presented one of the first cryptographic key distribution protocols. They gave an intuitive argument for its correctness, but called for more rigorous methods of reasoning about protocol correctness. Since then, cryptographic protocols have been subjected to a variety of formal analytic techniques, which have exposed a variety of flaws (in the Needham-Schroeder protocol as well as many others) and yielded some information about properties of correct protocols.

Because cryptographic protocols are so subtle, the project of developing a formal methodology for their analysis is fraught with particular challenges. To be effective as well as practically useful, a formal method for the analysis of cryptographic protocols must be *expressive*, it must have a *formal semantics*, it must be *intuitive* in its application, and it must be *automatable*.

An *expressive* methodology is one which provides mechanisms to capture not only the operation of a variety of protocols but also the variety of goals which these protocols may be intended to meet and the variety of attacks to which these protocols may be subjected.

A methodology must have a *formal semantics* in order for us to make rigorous statements

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

about the properties of the methodology itself. In particular, we desire assurance that a protocol proved secure (under some set of assumptions) within a particular methodology is indeed secure. We also wish to know that the methodology is sufficiently powerful to prove a secure protocol secure.

The utilization of a methodology must be *intuitive* so that a protocol designer may apply the methodology to a proposed protocol while needing little to no extra knowledge (beyond that required to design the protocol). A methodology that requires somewhat esoteric knowledge in an area such as complexity theory, probability theory, or logic programming, is unlikely to be truly effective.

Finally, a methodology must be *automatable* to some degree, since the proofs involved in protocol analysis are often both complex and tedious.

We present a formal method of protocol analysis that meets all these criteria: it has a truly formal semantics, it is necessarily and sufficiently expressive, its use requires no specialized knowledge other than being able to read standard informal protocol specifications, and it is fully automatable and very fast. In particular, the formal specifications utilized in the analysis can be automatically derived from the "standard" high-level specifications seen in the literature; it detects protocol flaws; it is logically sound and complete, so an absence of detected flaws corresponds to a proof of security; and automated analyses are very fast, taking less than a second.

This thesis is organized as follows: in Chapter 2 we give an informal description of the analysis tool we have developed; in Chapter 3 we provide a formal semantics for our analysis; in Chapter 4 we discuss our implementation and some results of protocol analysis; and in Chapter 5 we comment on our work and suggest directions for future inquiry. In the balance of this chapter, we review a number of approaches to the formal analysis of cryptographic protocols.

1.1 Approaches to Formal Analysis

Since Needham and Schroeder's original observation of the need for rigorous analysis of cryptographic protocols, a number of formal methods have been produced and used, with varying degrees of success. Meadows [Mea92] places these approaches into four categories:

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

(I) applications of generic specification/verification tools, (II) expert systems, (III) modeling with logics of knowledge and belief, and (IV) algebraic modeling.

Recent trends have refined this categorization. First, attention has turned more explicitly toward automated analysis; a number of groups have reported semi-automated or automated protocol analysis tools that are claimed to perform full analyses in a matter of minutes. Second, most analysis techniques are intended primarily to expose flaws in protocol design, i.e. to provide proofs of *insecurity*. But, while this is certainly a critical element of protocol development, it does not suffice if protocol designers wish to make claims about the security properties of their protocols. Thus, some researchers are investigating frameworks in which proofs of security can be produced and justified. In particular, Bellare and Rogaway [BR93b, BR93a, BR95] proposed an analytic method that rests on complexity-theoretic assumptions and techniques; they provide proofs that they claim demonstrate the security of the protocols they analyze.

Below, we review some of the techniques that have shown the most promise; these can be loosely categorized into (1) logics of knowledge and belief, (2) algebraic techniques, (3) first-order and related logics, and (4) complexity-theoretic techniques.

1.1.1 Logics of Knowledge and Belief

The seminal work in logic-based analysis of cryptographic protocols is that of Burrows, Abadi, and Needham [BAN90], who present a specialized logic based on reasoning about belief. While their method does reveal a number of flaws and appears to confirm the correctness of some protocols, it has been criticized widely for a number of reasons.

Formulae in BAN logic generally represent statements about the beliefs of the agents participating in a protocol; e.g. "P believes the message X is fresh", "P believes Q said X", "P believes Q believes X". BAN logic provides a set of such constructs and a number of inference rules used for reasoning about these statements.

Protocol analysis in BAN logic involves first "idealizing" the protocol — that is, translating the protocol from its original representation into formulae the logic can manipulate — then annotating the protocol with initial assumptions and other information about the agents' beliefs, and finally applying inference rules to reach conclusions about the agents'

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

final beliefs about the success of the protocol.

BAN logic as it was first formulated suffers from a number of flaws, as explained in [MB93, Sne91, Syv91]. In particular, criticisms center on the idealization step, the reliance on belief, and the semantics of the logic.

The idealization step often involves deleting fragments of messages, reformulating other message fragments, or inserting extra information. The BAN authors motivate this by arguing that the original presentation of a protocol is often "ambiguous and obscure" and hence not suitable for formal analysis. The idealization process, however, does not solve this problem. Rather, it is based on an intuitive understanding of the entire protocol — from the outset incorporating the very intuition-based reasoning that formal methods seek to outmode. Protocol specification should instead take place at a level close to that of the protocol description in a way that requires only a minimal understanding of how protocol messages are constructed.

The importance of belief in BAN logic is also problematic. First, as Mao and Boyd [MB93] point out, the logic permits the derivation of a formula of the form "P believes a nonce", even though the semantics of this statement is obviously nonsensical. More importantly, modeling of these protocols as collections of agents which reason about shared or individually held beliefs does not capture the real purpose of these protocols — to prevent intruders from acquiring secret information. The agents in these protocols simply exchange messages: a given principal sends certain messages at certain points and checks to ensure that messages received are of the correct form. If all messages are sent and received correctly, the principals behave as though the protocol was successful. Rather than trying to prove that agents will believe in the success of the protocol at its conclusion, analyses should seek to prove that intruders cannot "see" the secret information that cryptographic protocols are intended to protect.

Finally, the loosely-stated semantics of BAN logic — as well as its admittedly extendible character, which allows protocol analysts to add inference rules in an ad hoc manner — makes reasoning about properties of the logic itself difficult. We have already seen that it is possible to derive nonsensical statements within BAN logic, which calls its soundness into question. Abadi and Tuttle [AT91] do present a more formal semantics for the logic, but

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

their goal is only to "explain the logic of authentication", so while they do give an axiomatization for a restricted version of the logic that they state is sound, they also demonstrate its incompleteness without justifying this gap or explaining how it might be eliminated.

Thus, BAN logic, while it is able to detect some protocol flaws (which arise as additional assumptions that are required in order to show that the protocol accomplishes its goals), is not well-suited to general protocol analysis, as it lacks both a strong formal foundation and an automated analysis tool.

The BAN logic has been reformulated and revised in a number of different ways which address some of these criticisms.

Gong, Needham, and Yahalom [GNY90] use the BAN framework but include a number of new ideas and distinctions. For example, they incorporate the idea of *recognizability*, capturing the idea that agents have expectations about the contents of the messages they receive. They also distinguish between the informational content of a message and the beliefs that an agent may acquire from reasoning based on that information. While their approach has a much smaller dependence on idealization (they present a "protocol parser" algorithm that does some of that work in a deterministic way), they invoke intuition in "message extensions" — formal expressions of verbal statements of preconditions of particular messages. With the inclusion of ideas of recognizability and different levels of reasoning, their logic has over 40 inference rules, which nearly guarantees that their logic is unsound and incomplete.

Syverson and van Oorschot [SvO94] unify several ideas from [BAN90, GNY90, AT91, vO93] and develop a logic that subsumes these logics as well as a model theoretic semantics with respect to which their logic is sound.

Alves-Foss and Soule [AFS97] present a weakest precondition calculus for a BAN-like logic. In this system, the goals of the protocol are formulated as a set of postconditions. The analysis then works backwards through the protocol, at each step determining the weakest preconditions for which execution of the current step of the protocol will yield the given postcondition. The result of the analysis is a set of premises, or initial preconditions, which must be met in order for the protocol to function correctly. These can then be inspected to determine the correctness of the protocol. This approach represents a promising start

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

for the development of a tool for automatic generation of protocols that meet a set of postconditions. Many of the flaws inherent in BAN logic do remain: the original protocol needs to be reformulated ("expanded", in the authors' terminology), and, as the authors observe, it is not clear that the definition of the weakest preconditions is logically sound.

De Decker and Piessens [DP97] have developed an analyzer called CryptoLog (written in Prolog) based on belief logic that attempts to solve the problem of idealization by allowing assumptions to be stated at a high level and by explicitly specifying agents' actions. Thus, input to their analyzer consists of a set of assumptions (such as A, S **Trust** $K_{as} :=:$ **SharedKey** (A, S)) and an extended protocol specification, which includes both message exchanges and agent actions (such as B generates nonce N_b). The analyzer then does table-lookups to translate the assumptions and actions into belief logic formulae. Then the analyzer performs a series of checks to determine e.g. whether the sender is able to send the given message and if and how the message-receiver's beliefs should be extended (using the deduction rules from [GNY90]). At the end of the analysis, the analyzer lists all the beliefs held by each agent, announces whether any information has been leaked, and analyzes the possibilities for and ramifications of dictionary attacks. The automation of the analysis process is a welcome addition to belief-based analysis, but it is not clear that their replacement of idealization by the translation of high-level statements about the protocol eliminates the problems with the process as a whole. In addition, that their analysis is based on the GNY logic calls the logical properties of the analysis into question.

1.1.2 Algebraic Approaches

The idea of using algebraic specifications for security protocols was first developed by Dolev and Yao [DY83]. In their model, each agent X has associated operations E_X and D_X, corresponding to encryption and decryption. They assume that $E_X D_X = D_X E_X = 1$, and that knowledge of $E_X(M)$ and of all (X, E_X) pairs reveals nothing about the value of M. They also assume that the one-way functions used for encryption/decryption are unbreakable, that the (X, E_X) pairs cannot be tampered with, that all agents have access to all E_X, and that only X knows D_X. An intruder in their model can (1) function as a legitimate user and thereby initiate protocols with and potentially receive protocol messages

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

from any other user and (2) see any message on the network. Thus, from the perspective of the intruder, the protocol is a machine for producing words in some language. They consider two related types of protocols: "cascade" protocols, in which messages are simply plaintext to which a number of encryption or decryption operations have been applied; and "name-stamp" protocols, in which sending agents are allowed to append names to plaintext and encrypt the pair, and receiving agents may check or delete the names appended to the plaintext. Through some analysis, they develop a polynomial-time algorithm that determines whether or not a protocol of this type is secure.

Meadows [Mea96] based her Analyzer on Dolev and Yao's work. She noted two limitations of Dolev and Yao's work: first, that their analysis is applicable only to a restrictive class of protocols, and second that intruders may do more than just discover secret information: many protocols are "broken" when the intruder convinces an agent that information in the intruder's possession has a certain function (e.g. when the intruder successfully introduces a bogus session key). The intruder in Meadows' analysis, then, may view a protocol as a machine for generating words, beliefs, and events. Each participant has a set of beliefs that is modified by receiving messages; these messages are composed of words. Events are the state transitions during which participants create new words and modify their beliefs. Protocol specifications consist of (1) the intruder's initial knowledge (i.e. words the intruder knows), (2) operations that can be used to produce new words (such as encryption and decryption), (3) the initial states of the participants, (4) a set of reduction rules (which specify properties of the operations), and (5) a set of state transitions. State transitions are given by (1) an event statement that describes the transition, (2) a set of words input by the intruder, (3) a set of beliefs that must hold before the transition fires, (4) a set of words that are output when the transition fires, and (5) a set of beliefs that will hold after the transition fires. Beliefs have four parts: (1) the name of the participant who holds the belief, (2) a list of words, (3) what is believed about those words, and (4) the belief's status, i.e. whether the participant holds the belief. As an example, consider the following (partial) specification of the last message in the Needham-Schroeder protocol:

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

IF ⟨u(B),possibleseskey(R),[K],yes⟩
⟨u(B),nonce2(R),[I],yes⟩
not(⟨u(B),seskey(R),[K'],yes⟩)
{M} ⊆ W
THEN
⟨u(B),seskey(R),[K],id_check(d(K,M),s(I))⟩

This means that if user B believes that K is a possible session key for session R, and if he believes that I is the nonce generated by user B in round R, and if he does not already believe that some K' is the key for session R, and if he receives a message M, then if $d(K, M)$ reduces to $s(I)$, where s is the successor function, then he will believe that K is the session key for round R.

To analyze a protocol, we first specify a set of states which we wish to prove unreachable, i.e. those states which represent security violations. These are given as a list of (1) words learned by the intruder, (2) beliefs held by the participants, (3) beliefs that must not be held, and (4) words that must not be known by the intruder. We can also give sequences of events that must have occurred, sequences of events that must not have occurred, and sequences of events that will occur in the future. The Analyzer then takes each subset of the words, beliefs, and events in the state and compares them with each subset of the words, beliefs, and events output by each rule, and attempts to unify the two sets. In each successful case, it outputs the result after applying the unification to the input of the rule. The infinite search space can be limited by performing subgoal analysis to identify unreachable states or by providing the Analyzer with a grammar for a language of unreachable states. Protocols are generally analyzed interactively, with the user providing some guidance regarding the direction of the state space search. This method is an extremely powerful tool and has been used to find flaws in a variety of protocols. But it is not designed to effectively prove positive results about security: the state machine specification must be manually derived from the protocol specification; its interactive analysis is not very fast; and it is not clear that this approach to specification and analysis has a formal semantics with respect to which it is sound or complete.

Woo and Lam [WL93] present an algebraic system in which the specification is given in terms of the protocol each agent follows (rather than in terms of the intruder's state). Each

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

protocol is a sequence of actions (e.g. **Send**, **NewNonce**, **Accept**) that the agent takes. The semantics of the analysis specifies the effect of each action on the agent's state (essentially its knowledge); the global state is a tuple of all participating agents' states. A global transition is either an internal agent action or a communication action, which requires that the agents involved simultaneously execute a **Send** and **Receive** action, respectively. Security properties are formalized as assertions over protocol executions, and consist of correspondence assertions, which address authentication concerns, and secrecy assertions, which address secrecy concerns. The authors assert that their model is sufficiently formal to address precisely issues of soundness and completeness, although this is deferred as a question about specific analysis techniques. Additionally, it is unclear what proofs in this system actually entail, as the correctness assertions need to be proved for all possible executions.

Bolignano [Bol96] has developed a method that has been described [MCJ97] as occupying an intermediate position between the techniques of Woo and Lam and Meadows. In this framework, the intruder's state is represented by the words it can derive, while the agents' states are simply denoted by program counters. Proofs of security require first the identification of "windowing" properties which describe acceptable sequences of the agents' program counters; then these properties are proved by identifying and proving properties of the intruder's state that must be shown to be invariant over all protocol steps. While Bolignano states that his goal is simple and concise proofs (rather than automatability, for example), the proof process becomes unwieldy and must be customized for each protocol.

The process algebra CSP [Hoa85] has served as the foundation for a number of investigations of modeling protocols as communicating processes. In [Low96], the asymmetric key variant of the Needham-Schroeder protocol is modeled as a set of CSP processes; this model is tested using FDR, a model checker for CSP. Ryan and Zakiuddin [RZ97] present a summary of methods of formalizing security properties in CSP notation. Bryans and Schneider [BS97] present a CSP-based analysis of a recursive protocol presented in [BO97] (and also analyzed in [Pau97b]). While all of these analyses benefit from the wide usage CSP enjoys it is also clear that the state space limitations incurred by model-checking sharply affect the quality of the results, although research in this area is ongoing.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

In [AG98], Abadi and Gordon introduce a new calculus for communicating processes, called the *spi* calculus, that is specifically designed for analysis of cryptographic protocols. This calculus is an extension of the pi calculus that is designed for the analysis of cryptographic protocols. This method of protocol analysis allows protocols to be specified using a precise semantics; security properties are proven using equivalence of protocols.

Bella and Riccobene [BR97] use the Gurevich Abstract State Machine [Gur94] formalism to specify the Kerberos Authentication System. They use their specification to prove that the Kerberos system is correct — that is, that authorized users are allowed to gain the services they require — and demonstrate some flaws in the security of the system. They do not discuss how this formalism may be used to effect a systematic security analysis of a given arbitrary protocol.

1.1.3 Logic-Based Approaches

The logic-based approach has proven to be particularly well-suited for automation. Millen's Interrogator [KMM94] is a Prolog-based search engine that searches for instantiations of a predicate signifying the intruder's knowledge of specified data via some sequence of messages to an insecure goal state. Protocols are specified as a sequence of transmit and receive actions with additional information about the initial state and message format. While the Interrogator has successfully found protocol flaws fairly quickly, search time may vary significantly depending on the precise format of the protocol specification and the amount of information provided about the insecure goal states. Additionally, search heuristics may prevent some flaws from being noticed.

Paulson [Pau96b, Pau97a, Pau97b] has developed a model that is tailored for use with Isabelle, an ML-based theorem prover. His model is adversary-centric: the adversary can read all messages and send new messages; it can also act as an "honest" agent using its own long-term key. Protocols are modeled by defining the set of possible utterances (both by the honest agents and, possibly, by the adversary), then inductively defining the set of possible protocol traces. Protocol specifications are sets of rules for extending a protocol trace, including a rule describing what the adversary may send. Protocol analysis then consists of proving properties (regularity, uniqueness, secrecy, etc.) of all possible traces. Paulson has

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

developed a semi-automated analysis system based on Isabelle scripts designed specifically for protocol analysis. The foundation of his theory is a system of message analysis; he defines operators over sets of messages that can be used to express rules for trace extension (including how traces may be extended by the intruder's actions). Paulson makes good use of the inductive nature of protocol definitions in structuring his formal model, but his analysis requires an excessive amount of by-hand computation and pruning of the proof tree. According to Paulson, analysis of a new protocol can take several days; analysis of a revised protocol takes some hours. In the course of the analysis over 100 theorems are proved about the protocol. Thus, this technique features a semi-automated analysis tool that is capable of discovering a variety of protocol flaws, but it is not clear that the underlying semantics are appropriate for proving protocols to be secure.

Marrero et al [MCJ97] has produced a *model checker* for security protocols. Like Paulson's work, this approach involves reasoning about all possible protocol traces, but rather than reason about the logical properties of these traces, the model checker performs a state space search and tests the properties of each trace. Traces are defined (as in Woo and Lam's approach [WL93]) as alternating sequences of states and agent actions. The global state contains information about how many times agents have initiated a protocol or responded to another principal, secrets the intruder is never allowed to know, and temporary secrets generated during the protocol. Agent protocols are specified as sequences of actions (e.g. SEND, RECEIVE, NEWNONCE), which the authors assert may be derived automatically from the standard high-level protocol specification; the whole protocol is the composition of the agent protocols. Security is defined in terms of *secrecy* and *correspondence* properties — each protocol specification includes a list of terms which the intruder must not learn and a mapping between protocol runs of distinct agents. The correspondence property guarantees that each state has only a finite number of reachable next states, which gives rise to a finite bound on the size of the state space. This allows a straightforward depth-first search of the state space, checking that no reachable state violates the security specifications. These searches typically take place in under a minute. This method is currently applied to single protocol runs; because this model requires a complex representation of state, analysis of multiple concurrent runs greatly increases the size and complexity of the state space. Thus,

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

this analysis technique is both automatable and fast, and is capable of detecting some flaws (those that may arise in a single run). This technique does lack a formal semantics, so an analysis that demonstrates no flaws can only build confidence in security rather than prove it absolutely.

Kindred and Wing [KW96] have developed another ML-based tool for checking security protocols. They do not provide a new analytical framework; instead, they present a more general *theory checker* that can be used with a wide variety of specification logics; their *theory checker generator* automatically creates a theory checker for a given logic. The theory checker generator takes as input a logic — a set of function names and inference rules — and outputs a theory checker for that logic. The theory checker takes as input a protocol specification — a set of initial assumptions and messages encoded as logical formulae — and then computes a subset of the formulae derivable from the specification. To control the size of the proof tree, Kindred and Wing constrain the input logics in a manner that guarantees that the set F of derivable formulae is finite but that any valid formula can be derived from F using a subset of the provided inference rules. Kindred and Wing produce theory checkers for BAN, Autlog [KW94] (a variant of BAN), and Kailar's logic of accountability [Kai96], which is similar to BAN, and use these to check a variety of protocols (including a duplication of the original BAN analysis of the Needham-Schroeder protocol, Kerberos, and X.509). They report that analyses take 1.5 to 13 minutes. Their tool is an excellent vehicle for automated analysis using logical formalisms, but it does not (nor is it intended to) correct flaws or inadequacies in the logics, nor does it retain its desirable properties when applied to formalisms that don't meet the restrictions it imposes.

1.1.4 Complexity-Theoretic Approaches

Bellare and Rogaway [BR93b, BR93a, BR95] present a complexity-theoretic model of cryptographic protocols that does away entirely with issues of agents' knowledge, belief, and trust and frames the entire protocol in terms of the adversary. In this model, the adversary is the means by which the agents communicate: all messages are sent to the adversary, who may intercept and alter them, or not, as she pleases. In addition, she may create new messages, learn arbitrary session keys, and "corrupt" other agents. Formally, this is

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

accomplished by placing at the adversary's disposal a number of oracles, corresponding to particular agents (including the key server) engaged in a particular key exchange session. A run of the protocol consists of a set of initializations followed by oracle queries, which the adversary may generate based on a combination of earlier query responses, its internal state, and coin flips. The oracle responses (i.e. messages generated by the agents) are determined, possibly probabilistically, by the protocol: as a result of a query, the oracle outputs up to two messages (to the key server and/or to the other agent); the session key, if it has been successfully distributed; and an optional ACCEPT/REJECT bit, indicating whether the oracle has successfully participated in a complete key distribution session. In addition, the adversary can make two types of protocol-independent queries: she can ask an agent who has successfully received a session key to **Reveal** that key — this allows us to capture protocol bugs which are based on exploiting the loss of some session key; and the adversary can **Corrupt** any oracle participating in a session, at which point the oracle will reveal its long-term key, the conversation it has had so far, and its coin flips, and will replace its long-term key by a key given to it by the adversary — this reflects the situation in which an agent's security has been completely compromised. Security in this model is defined as follows: the adversary picks a session key it wishes to learn, with the restriction that it cannot have issued a **Reveal** or **Corrupt** query to any of the agents holding that key. It then issues a **Test** query of the oracle holding that key. The oracle responds randomly with either the session key or a random string. The adversary tries to guess which it receives; if the probability of it guessing correctly is significantly greater than $\frac{1}{2}$, then the protocol is insecure.

The particular emphasis placed on the adversary in this approach (greater than that of Dolev and Yao and related work) is a valuable innovation, but, while the framing of this model is very rigorous and produces provably secure protocols, it is not a strong foundation for a general protocol analysis tool. The proofs required by this model, as well as the techniques used within them, are not easily automatable or extendible to arbitrary protocols. Moreover, this model is designed solely for proving *security*; there is no mechanism for uncovering protocol flaws. Finally, while the complexity-theoretic framework of the model has the potential to unify many commonly made assumptions about cryptographic primitives,

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

it is not clear that the complexity-theoretic assumptions on which this method rests are the correct ones; Goldreich [Gol] points out that the existence of one-way functions is not necessarily sufficient for probabilistically secure encryption operations.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

CHAPTER 2

Informal Description of the Method

In this chapter we provide an informal description and discussion of the analysis tool we have developed. In particular, we describe the environment in which the protocols to be analyzed will be executed, discuss the role of the adversary, and motivate the definition of attack (or protocol flaw) used in the analysis. We then introduce our method for modeling and analyzing protocols. Finally, we discuss how to augment this technique to reason about protocols containing timestamps.

2.1 Protocols and the Environment

We consider those security protocols generally categorized as "key distribution" and/or "entity authentication" protocols.[1] In the sequel we will use the general term "security protocol" to denote protocols of this type. In this section, we will discuss the primary elements of these protocols.

2.1.1 Agents

Security protocols typically are concerned with 3 classes of entities. First, protocols are carried out in order to give two[2] communicating parties assurances about the security of subsequent communication; these parties are generally denoted by A and B. Second,

[1] Bellare and Rogaway [BR93a] clarify the distinction: "Entity authentication is the process by which an agent in a distributed system gains confidence in the identity of a communication partner"; this process may be accompanied by the distribution of a (presumably) new key.

[2] The vast majority of protocols of this type are designed to achieve distribution/authentication for two agents; [BO97], however, present a "recursive authentication protocol" that allows an arbitrary number of agents to participate. While the method discussed here can easily be generalized to accommodate any fixed number of participants, analysis in the arbitrary case is out of the scope of the present work.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

a number of protocols require the presence of one or more "trusted" parties, referred to as *key servers*; these are usually denoted, where present, by S. Finally, were all entities "honest", security protocols would be unnecessary; from the existence of such protocols we must infer the existence of some "dishonest" agent, often referred to as the *adversary* or *intruder*. While this entity is not an explicit part of a protocol specification, its presence is critical to protocol analysis; this type of agent is discussed more fully below.

2.1.2 Messages

Security protocols are most often described and specified as sequences of messages[3] sent among the 2 or 3 "legitimate" agents. These messages are constructed using a small collection of data and message construction operations. The "atomic" data items are drawn from the following sets:

Agent Identifiers. The letters A, B, and (optionally) S.

Cryptographic Keys. These are usually represented by K_A, K_B, K_i, etc. K_A and K_B represent the keys shared by agents A and B, respectively, with S. Other keys may be created during the execution of the protocol; these are denoted e.g. by K_i.[4]

Nonces. In this thesis nonces will be denoted as N_i, M_i, etc. These quantities are unpredictable numbers[5] generated by an agent during protocol execution. They are intended to be "used only once". Thus, if an agent generates a nonce N_i and incorporates this nonce into a message it sends at time t, any message it receives later that includes N_i must have been generated after time t. Nonces, then, can be used to provide a rough agent-specific sense of freshness.

Timestamps. These are data elements that represent absolute time — typically, they indicate the time at which the message containing them was sent, although sometimes they represent an expiration time. Their use requires the presence of distributed time; Gong [Gon93] observes that such an application may require *secure* distributed time proto-

[3]In particular, this is the level of specification at which we carry out our analysis. Some specifications provide additional details, e.g. state machines for the participating agents. While attacks might be mounted against a protocol by exploiting these implementation details, we regard such attacks as occurring at a lower level of abstraction and hence do not seek to capture them in our analysis.

[4]We represent information generated during a particular session by subscripting with i, the *session index*. Thus in two different sessions i and j, $K_i \neq K_j$, as we would hope.

[5]Typically intended to be implemented as pseudorandom numbers; [Gon93] notes that other kinds of "freshness identifiers", such as counters, may be sufficient in certain contexts.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

cols, although Neuman and Stubblebine [NS93] assert that this requirement may be overly strong.[6]

Arbitrary Data. Although it is often not an explicit part of protocol specification, most security protocols are executed so that the agents involved may make some assumptions about subsequent data exchange. In this thesis, the data exchanged after the execution of the protocol proper will often be made explicit and denoted as P_i or Q_i.

These data items may be combined (recursively) in the following ways:

Symmetric Key Encryption. Any message m may be encrypted using key k; the result is denoted $\{m\}_k$. In this thesis we also use the functional notation $e(m, k)$. This message can be decrypted (and the value of m recovered) by any agent knowing k.

Cryptographic Hash. This operation allows agents to produce secure "message digests". Any agent that knows key k may produce the cryptographic hash of message m, denoted $\langle m \rangle_k$ or $h(m, k)$. The value of m cannot be recovered from this quantity, but, when transmitted with m, the hash allows any receiving agent that knows k to perform an integrity check on m.

Handshake. This type of function allows agents to permute messages in an easily recognizable fashion; for example, it is often implemented as the decrement function. In general, this function is easily computed and easily inverted, although in some applications it may be instantiated as a hash function. We will denote this function as f.

Concatenation and Projection. These allow agents to construct lists of messages and to extract elements of these lists. We denote the list of messages $m_1, \ldots, m_k$ as $[m_1, \ldots, m_k]$.

2.2 The Adversary

The role of the adversary varies considerably across the many protocol analysis tools that have been developed. Most analyses based on logics of belief never make the adversary's presence explicit, as the goal of these analyses is to draw conclusions about the beliefs of the legitimate agents. A number of other approaches model the adversary as roughly the same sort of agent as the other legitimate agents; these approaches face the challenge of

[6] Indeed, [DGT95] have implemented a timestamp-based protocol in a manner that allows for drifting or intermittently synchronized clocks.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

finding a way to express the range of the adversary's possible action. In work such as that of Bellare and Rogaway, the adversary is modeled explicitly and the legitimate agents become implicit; in effect, the adversary *becomes* the network on which messages of particular forms (that is, those sent by the legitimate agents) may appear.

This is essentially the view of the adversary we take in this thesis. The adversary has complete control of the network: it may read, block, delay or alter any message sent as well as create new messages. Additionally, the adversary has all the abilities of a legitimate agent: it knows the agent identifiers that may be used, it can create nonces and timestamps, it can perform encryption and decryption, and it may share a key (or possibly several) with the key server. It does not have any particular capacity to perform cryptanalysis; that is, it cannot break an encryption immediately, but it may be able to if given sufficient time.

The goal of our adversary is, most generally, to acquire illegitimately[7] some privilege of a legitimate user.[8] In particular, the adversary may try to learn some information not intended for it (violating *secrecy*), or it may attempt to masquerade as another (legitimate) agent (violating *authenticity*).

2.3 Defining Attack

Crucial to an analysis technique which is designed to search for attacks is a clear definition of attack. In this section we discuss some relevant ideas from the literature and arrive at the definition of attack used in our analysis.

The idea (and definition) of attack is one that, like the idea of the adversary, varies widely across the literature. In the presentation of BAN logic, attacks are not discussed. Instead, BAN analysis may demonstrate the necessity of making additional assumptions about the protocol; whether or not the protocol is "flawed" depends on the reasonableness of these assumptions. In Dolev and Yao's work, an attack corresponds to the adversary's ability to learn plaintext that has been encrypted. Meadows extends this idea to accom-

[7] Note that this framing of the adversary's goal admits the possibility that the adversary is not a "true" outsider but rather a legitimate user acting in an unauthorized fashion. We make no effective distinction between the two cases.

[8] In the most general case, a malicious entity may have a broader set of goals, including e.g. denial of service. In this work, we focus on the kinds of attacks the protocols in question are specifically designed to prevent.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

modate the possibility that the adversary is able to cause a bogus key to be distributed;[9] in general, Meadows allows the specification of a class of *insecure* states, the reachability of which corresponds to an attack on the protocol. Bellare and Rogaway do not define an attack directly, but instead define *security* in terms of complexity-theoretic assumptions. A protocol which allows those assumptions to be (somehow) violated is thus *insecure*.

Several researchers have explicitly addressed the issue of protocol flaws (we use the terms *attack* and *flaw* interchangeably). Carlsen [Car94] considers flaws that may occur at all levels of protocol development, from high-level protocol design to protocol implementation. He proposes the following list of flaw categories: *elementary* flaws relating to the improper use of cryptography or the protocol operating environment; *password-guessing* flaws relating to reliance on inadequate password policies; *freshless* [*sic*] flaws based on a principal's inability to determine whether or not a message has been created recently; *oracle* flaws that allow an adversary to obtain secret information from the protocol dialog; *type* flaws that allow the adversary to exploit a principal's inability to determine the type (key, nonce, etc.) of a particular message; *internal* flaws that arise from a principal's failure to perform some necessary operation (e.g. verification) on a message; and *crypto-system related* flaws that arise from subtle flaws in the use of a particular cryptosystem.

While flaws of any type may prove fatal to a protocol implementation, we restrict our attention in this thesis to flaws that arise in the design stage (*functional specification flaws* in Carlsen's terminology). Specifically, we consider only freshless and oracle flaws, and assume that other aspects of the system (including typing) are robust. Carlsen describes freshless flaws by observing that a key element is the ability of the adversary to use intercepted messages in a masquerade, as this allows it nearly unlimited time to mount an attack. Attacks based on this flaw consist of an adversary intercepting a message from one protocol session, then either performing cryptanalysis or waiting for keys to be revealed by careless principals, then using the results in a new protocol session — and successfully masquerading as a principal without any knowledge specific to the later protocol session. Oracle flaws may be exploited either in *single-session* attacks or in *cross-session* attacks. In either case, attacks

[9] Of course, this is captured by Dolev and Yao's condition, since if the adversary can spoof a session key with a key that it knows, it will be able to recover plaintext encrypted with that key.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

based on these flaws involve the adversary exploiting similarities in message structure. The adversary echoes a message from a (usually) current session to one of the participants; this message is of acceptable form but unusual content. The unwitting participant, by simply conforming to its role in the protocol, emits a response to this message in which it inadvertently reveals some secret information.

Syverson [Syv94] also proposes a classification system for protocol attacks, which is in some sense a refinement of Carlsen's freshless and oracle flaw categories. He observes that protocol designs generally capture only gross features of the cryptographic system underpinning the protocol; hence analysis is primarily concerned with restricting an adversary's malevolent use of the protocol itself. That is, "most of the effort in devising and reasoning about cryptographic protocols is directed at precluding one form or another of replay attack." Syverson's taxonomy is a complete classification of replay attacks. He categorizes attacks with respect to two axes: message origin and message destination. On the message origin axis he places run external attacks (replay of messages from other protocol runs) and run internal attacks (replay of messages from the current protocol run). On the destination axis he places two major categories: interleavings (which requires contemporaneous protocol runs and therefore admits only run external attacks) and classic replays. Within each of these two categories are reflections (in which messages are "returned to sender"), deflections to a third party, and delays (in which the intended recipient receives the message after a delay).

While Syverson's and Carlsen's analyses are valuable both for determining what kinds of attacks are captured by a particular protocol analysis technique and for providing guidance in the design of secure protocols, they do not provide a method for determining whether any of these flaws exist in a particular protocol. As our method is devoted to detecting freshless/oracle/replay flaws, we must find a way generally to characterize flawed protocols; we will then be able simply to test protocols against this characterization.

First, we revisit the adversary's goals, as discussed at the end of the previous section. The purpose of a security protocol is to assure each participant of the identity of the other agent (that is, to achieve *authenticity*), and, possibly, to distribute a new cryptographic key intended to secure some subsequent communication (thereby achieving *secrecy*). Thus,

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

an entity who wishes to subvert a protocol may do this in two ways: it may convey false assurance of its identity — that is, it may masquerade as another (legitimate) agent — or it may acquire the material transmitted (presumably securely) after the protocol is executed. In our model, we test these two possibilities by determining (1) whether the adversary is able to produce all the protocol messages generated by one participant, thereby creating the false impression that a legitimate agent is causing these messages to be sent, and (2) whether the adversary is able to learn any of the information transmitted at the end of the protocol (that is, P_i or Q_i).

Second, we must characterize how the adversary differs from a legitimate agent. Clearly, a crucial distinction is that the legitimate agents may share keys K_A and K_B with the key server. In our analysis, we assume that the adversary will never learn these keys[10] (although the adversary may share other keys with S). In addition, we observe that the idea of legitimacy is actually session-dependent — as we have noted, the adversary may be a legitimate participant in some protocol session even as it is trying to attack another session. Thus, the adversary is also distinguished from the other participants in terms of its access to session-dependent information.[11] Specifically, we assume that the adversary has no a *priori* knowledge of session-dependent information generated by other agents, although it may be able to gain access to this information during the course of the protocol's execution.

Combining these ideas, we conclude that an adversary may attack a protocol if it can, without any knowledge of the shared keys of the participating agents or a *priori* knowledge of session-dependent information, successfully produce the messages received by either participant or learn the information protected by the distributed key.

2.4 Modeling Protocols

Our method for modeling and analyzing protocols is based on ideas from logic programming (see e.g. [Hog84]). The core of this method is a relation *Knows* which represents what the adversary knows. That is, *Knows*(x) is true iff the adversary knows x. *Knows* initially

[10] While it may be possible for an adversary to learn these keys from some protocol in which they are sent as messages, we exclude such poorly-designed protocols from our analysis.

[11] That is, information that is generated during the course of the protocol execution, such as nonces, session keys, timestamps, etc.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

contains some atomic data items, such as agent identifiers; we model the protocol as a set of inference rules which mirror the process by which the adversary adds to its knowledge set.

In the remainder of this section, we present our modeling technique using a case-study of the well-known Needham-Schroeder [NS78] protocol.[12]

2.4.1 Example: Modeling the Needham-Schroeder Protocol

We first reproduce the Needham-Schroeder protocol in Figure 2.1 as it was originally presented in [NS78], with some minor notational changes for the sake of consistency.

$$
\begin{array}{rll}
1. & A \rightarrow S: & A, B, N_i \\
2. & S \rightarrow A: & \{N_i, B, K_i, \{K_i, A\}_{K_B}\}_{K_A} \\
3. & A \rightarrow B: & \{K_i, A\}_{K_B} \\
4. & B \rightarrow A: & \{M_i\}_{K_i} \\
5. & A \rightarrow B: & \{M_i - 1\}_{K_i} \\
6a. & A \rightarrow B: & \{P_i\}_{K_i} \\
6b. & B \rightarrow A: & \{Q_i\}_{K_i}
\end{array}
$$

Figure 2.1: The Needham-Schroeder protocol

In the first message, one of the parties, A, sends a message to the key server S, indicating that A wishes to initiate communication with B. A includes a nonce N_i, which A uses to identify the response to its request. S then responds with a message encrypted under the key A and S share. This message contains A's nonce, the new key K_i, and a message encrypted under the key B and S share that A will forward to B. A then forwards this message to B, which contains the new key and an indication that it is to be used for communication with A. The next two messages are a "confirmation" that the parties have received the new key. B sends a new nonce M_i to A, encrypted under the new key; A then decrypts the message and replies with a handshake. Then the agents proceed with their session. We assume that the session key will be used at some point by either or both agents to encrypt information that is to be kept secret from the adversary; we model this secret information explicitly as P_i and Q_i (messages 6*a* and 6*b* are added for our analysis and are not part of the original

[12] The symmetric-key version.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

version of the protocol).

The inference rules we use to model protocols are grouped into two categories. First, we include inference rules which allows us to reason about message structure independent of message content. The inference rules of this sort necessary for the analysis of the Needham-Schroeder protocol are given in Figure 2.2.

$$[a]\ \frac{x\ k}{e(x,k)} \qquad [c]\ \frac{x_1\ x_2\ \ldots\ x_k}{[x_1,x_2,\ldots,x_k]} \qquad [e]\ \frac{x}{f(x)}$$

$$[b]\ \frac{k\ e(x,k)}{x} \qquad [d]\ \frac{[x_1,x_2,\ldots,x_k]}{x_i} \qquad [f]\ \frac{f(x)}{x}$$

Figure 2.2: Message-structure rules for the Needham-Schroeder protocol

These rules capture the essential properties of the message composition operations used in the protocol; for example, rule $[b]$ says that in order to learn a message x from its encryption under the key k, it suffices to know the encrypted text and the encryption key.

The other category of inference rule consists of rules derived from the protocol itself. To construct the inference rules associated with this protocol, we consider the messages sent by each agent: from the perspective we use in our analysis, each protocol message represents an opportunity for the adversary to acquire new information. In order for the adversary to learn the contents of a message sent by an agent, it must be able to construct all the messages received by the agent before the message is sent. The agents participating in the protocol have only limited expectations about the messages they receive: they must arrive in the correct order, and they must be of the correct form with certain fixed submessages. The adversary may be able to exploit the freedom in message construction these limited expectations afford. In the inference rules given in Figure 2.3, w, x, and y vary over all possible messages; a and b vary over agent IDs, and m varies over session keys.

2.5 Analyzing Protocols

Once we have provided the protocol-specific inference rules, the analysis of the protocol consists of the generation of four proof trees: one attempting to prove that the adversary learns P_i, one that it learns Q_i, and one for each of the cases in which it is able to counterfeit

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

[A.1]	$\dfrac{-}{[A, B, N_i]}$	A initiates the protocol by sending $[A, B, N_i]$, which is therefore immediately available to the adversary.
[A.2]	$\dfrac{e([N_i, B, m, y], k(A))}{y}$	A receives a message of the form $\{N_i, B, K_i, \{K_i, A\}_{K_B}\}_{K_A}$. In response, it sends $\{K_i, A\}_{K_B}$. Thus, if the adversary can produce a message of the form $e([N_i, B, m, y], k(A))$, it will learn y.
[A.3]	$\dfrac{e([N_i, B, m, y], k(A))\ \ e(w, m)}{e(f(w), m)}$	A then receives a message of the form $\{M_i\}_{K_i}$. In response, it sends $\{M_i - 1\}_{K_i}$. Thus, if the adversary can produce both the previous message and a message $e(w, m)$, it can learn $e(f(w), m)$.
[A.4]	$\dfrac{e([N_i, B, m, y], k(A))\ \ e(w, m)}{e(P_i, m)}$	Finally, A will emit a message $\{P_i\}_{K_i}$. It requires no other message receipt.
[B.1]	$\dfrac{e([m, a], k(B))}{e(M_i, m)}$	B first receives a message of the form $\{K_i, A\}_{K_B}$; it then creates a nonce M_i and sends this encrypted under the key it just received. Since B has no expectation of the contents of this message, the adversary may attempt to substitute some other message for K_i.
[B.2]	$\dfrac{e(f(M_i), m)\ \ e([m, a], k(B))}{e(Q_i, m)}$	B will then receive a message of the form $\{M_i - 1\}_{K_i}$, at which point B may send out Q_i, encrypted under whatever it has received as the session key.
[S.1]	$\dfrac{[a, b, x]}{e([x, b, K_i, e([K_i, a], k(b))], k(a))}$	S only sends and receives one message in a given protocol session. It receives a nonce, then incorporates that nonce into the message it emits in response. The adversary may substitute other information for the nonce.

Figure 2.3: Needham-Schroeder protocol-specific inference rules

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

P_i or Q_i by producing all the messages generated by one of the agents. In logic programming terminology, the goal of each proof is the quantity the adversary wishes to know. The analysis proceeds backwards, searching for a sequence of rules the adversary can apply so that the only initial knowledge necessary is session-independent; additionally, any proof candidate that requires the adversary's knowledge of K_A or K_B is disqualified. To illustrate, we continue with our example of the Needham-Schroeder protocol.

2.5.1 Example: Analyzing the Needham-Schroeder Protocol

This protocol has a well-known flaw [DS81]: if an adversary has seen and remembered an old message of the form $\{K_x, A\}_{K_B}$, she may masquerade as A and send this message to B, who will interpret this as the third message in the protocol. If the adversary also knows the value of K_x, either through cryptanalysis or from the context of some earlier protocol session, she and B may complete the "handshake", and B will believe he has mutually authenticated with A.

We consider a branch of the proof tree from the analysis of Q_i. We begin with the statement that the adversary sees Q_i, then apply inference rules until all possibilities have been exhausted or until an attack has been found. See Figure 2.4 for a diagram of the proof tree corresponding to the known attack on this protocol.

In order to know Q_i, it suffices to learn some key m and the encryption of Q_i under that key. From rule [B.2], we know that the adversary can learn $e(Q_i, m)$ by learning $e([m, a], k(B))$ and $e(f(M_i), m)$. From [a], in order to learn $e(f(M_i), m)$, it suffices to learn $f(M_i)$ and m. From [f], it is sufficient to learn M_i in order to learn $f(M_i)$. In order to learn N_i', the adversary may learn $e(M_i, m)$ and m. Finally, from [B.1], the adversary may learn $e(M_i, m)$ by learning $e([m, a], k(B))$.

2.6 Timestamps

Timestamps are a type of atomic message that is qualitatively different from the other messages we have discussed so far. In particular, they are session-dependent in the sense that they are created during the execution of the protocol, but they are globally understood (whereas the session-dependent messages discussed above are associated with a particular

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

$$Q_i$$

$$e(Q_i, m), m \qquad [b]$$

$$e([m, a], k(B)), e(f(M_i), m), m \qquad [B.2]$$

$$e([m, a], k(B)), f(M_i), m \qquad [a]$$

$$e([m, a], k(B)), M_i, m \qquad [f]$$

$$e([m, a], k(B)), e(M_i, m), m \qquad [b]$$

$$e([m, a], k(B)), m \qquad [B.1]$$

Figure 2.4: Partial analysis of the Needham-Schroeder protocol

agent).

To discover how to reason about timestamps we must determine how they are used to foil the adversary, or, conversely, how the adversary may mount an attack on a timestamp-based protocol. The role of the timestamp is to impose a specific temporal window of message validity; that is, the timestamp gives an indication of the interval of time during which its enclosing message must be received in order to be acceptable. The implicit assumption is that only legitimate agents will be able to send acceptably timestamped messages. Thus, when timestamps are used, an agent is not satisfied simply by receiving a message; it may also need to check the value of the timestamps in that message. We can capture the role of the timestamp quite simply, then: we will augment the protocol-specific rules by allowing an additional form of clause in the antecedents. We introduce a new relation, *Curr*; if a clause $Curr(t)$ appears in the antecedent of a rule, this means that the timestamp t must be current in order for the agent to produce the message represented by the consequent. *Curr* is defined in terms of *Knows*; in essence $Curr(t)$ is true if the adversary has sufficient knowledge (in particular of encryption keys) to produce the message containing t.

This is best illustrated by a simple example. Figure 2.5 gives the description of a simple timestamp-based protocol called the Wide-Mouthed Frog protocol. In this protocol, A

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

generates the new sesion key K_i and sends it to S along with a timestamp TS_1. If TS_1 is valid, S then sends K_i on to B with another timestamp TS_2. If TS_2 is valid, B then takes K_i as a new session key and engages in communication with A.

$$
\begin{aligned}
1.\ & A \rightarrow S: \quad A, \{TS_1, B, K_i\}_{K_A} \\
2.\ & S \rightarrow B: \quad \{TS_2, A, K_i\}_{K_B} \\
(3a.)\ & B \rightarrow A: \quad \{Q_i\}_{K_i} \\
(3b.)\ & A \rightarrow B: \quad \{P_i\}_{K_i}
\end{aligned}
$$

Figure 2.5: The Wide-Mouthed Frog protocol

[A.1]	$\dfrac{-}{e([TS_1, B, K_i], k(A))}$	A initiates the protocol by sending the new session key K_i and a timestamp TS_1 to S, encrypted the shared key of A and S.
[S.1]	$\dfrac{e([t, b, m], k(a)), Curr(t)}{e([TS_2, a, m], k(b))}$	S receives an encrypted message containing a session key and a timestamp. If the timestamp is valid, it sends the session key on to agent b tagged with a new timestamp TS_2.
[A.2]	$\dfrac{-}{e(P_i, K_i))}$	A may send P_i at any time, since it originates the session key.
[B.1]	$\dfrac{e([t, a, m], k(B)), Curr(t)}{e(Q_i, m)}$	B receives a messages containing a session key and a timestamp. If the timestamp is current, it accepts the session key and starts communication with agent a.

Figure 2.6: Wide-Mouthed Frog protocol-specific inference rules

The inference rules for this protocol are given in Figure 2.6. The translation from the protocol description is much as before, but now we add an additional condition to the antecedents corresponding to messages containing timestamps which must be checked. As we said above, *Curr* is defined in terms of *Knows*, so in this example we have $Knows(K_A) \rightarrow Curr(TS_1)$ and $Knows(K_B) \rightarrow Curr(TS_2)$. In the search for an attack, then, if the adversary can learn the conditions for $Curr(t)$ then $Curr(t)$ is no longer a requirement for an attack — the adversary has successfully forged the timestamp. Otherwise, the $Curr(t)$ condition will be carried along so that any successful attack will be conditional upon the

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

validity of timestamps.[13] We discuss this more fully in Chapter 4.

This serves to illustrate the methodology of our analysis. In Chapter 4 we give a more extensive catalog of analytical results. In the next chapter we continue discussion of our method with the purpose of providing a formal semantics for it.

[13]Since the issues of the size of the interval of validity and the difficulty of altering accepted messages are implementation dependent, we do not address them directly in our analysis.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

CHAPTER 3

A Formal Semantics

We divide the process of developing a formal semantics into two parts. We first give more precise model theory for the messages that comprise the protocol, then we address the semantics of reasoning about the adversary's knowledge of these messages. In this chapter, we will present a form of resolution which we demonstrate is sound and complete; this resolution is the basis for the implementation of our analysis as discussed in the next chapter.

3.1 A Model of Messages

We reason about a universe of **Messages**, which contains 3 disjoint subsets: **Agents**, **Keys**, and **Data**. **Agents** is the set of agent IDs in the system; **Keys** is the set of long-term keys (those shared between agents and the key server), and **Data** is the set of nonces, timestamps, session keys, and other session-based information.

In order to map **Agents** to their **Keys** we need a one-to-one function k : **Agents** $\rightarrow$ **Keys**. While it is possible that a given agent may have a number of keys, we make the conservative assumption that only one will be used.

We define the handshake function f as a one-to-one function f : **Data** $\rightarrow$ **Messages**.

Encryption and hashing are modeled as functions e and h from **Messages** $\times$ **Keys** to **Messages** such that $e(\cdot, k)$ and $h(\cdot, k)$ are one-to-one (this is an assumption made for the sake of the model; in general hash functions are not one-to-one).

Finally, concatenation is modeled as the usual function from **Messages**$^+$ to **Messages**.

We then define a relation, *Knows*, on **Messages** which represents the adversary's knowl-

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

edge. We derive this relation by first specifying the adversary's knowledge prior to a protocol session; we assume that the adversary knows everything in **Agents**, as well as all elements of **Data** *except* those which will be used in the protocol session (in particular, the adversary does not initially know the nonces and session keys used in the session), and all elements of **Keys** except $k(A)$ and $k(B)$.

The adversary's ability to expand its knowledge is captured by a set of inference rules. As described in Chapter 2, our inference rules fall into two categories: those inference rules which describe message structure and those rules which specify a particular protocol. In Figure 3.1 we list the message structure inference rules. *Knows* is implicit in these rules; that is, a rule of the form

$$\frac{\gamma_1 \ \cdots \ \gamma_m}{\delta_1 \ \ldots \ \delta_n}$$

is understood as

$$\frac{Knows(\gamma_1) \ \ldots \ Knows(\gamma_m)}{Knows(\delta_1) \ \ldots \ Knows(\delta_n)}$$

$$[a] \ \frac{x \ k}{e(x,k)} \qquad [c] \ \frac{x_1 \ x_2 \ \ldots \ x_k}{[x_1, x_2, \ldots, x_k]} \qquad [e] \ \frac{x}{f(x)}$$

$$[b] \ \frac{k \ e(x,k)}{x} \qquad [d] \ \frac{[x_1, x_2, \ldots, x_k]}{x_i} \qquad [f] \ \frac{f(x)}{x}$$

$$[g] \ \frac{x \ k}{h(x,k)}$$

Figure 3.1: Message-structure rules

The inference rules that specify a particular protocol are derived from the informal specification of the protocol as described in Chapter 2.

3.2 Linear Logic

The key to providing a formal semantics for our technique is recognizing the precise nature of the behavior of the inference rules. Specifically, the two types of inference rules we have discussed have two distinct behaviors. The message-structure rules may be used arbitrarily, but the protocol-specific rules must be used more carefully, because they must reflect exactly how the agents behave. When an agent participates in a protocol session,

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

it expects to send each of the messages in its "script" at most once. For example, for some particular session i of the Needham-Schroeder protocol, with corresponding session key K_i, we would *not* expect S to send two messages such as $\{N_i, B, K_i, \{K_i, A\}_{K_B}\}_{K_A}$ and $\{N_i, C, K_i, \{K_i, D\}_{K_C}\}_{K_D}$. Thus, while the agent variables a and b and the message variable x (corresponding to A's nonce) may be instantiated arbitrarily for a given session, that instantiation is fixed. Therefore, while the message-structure rules may be instantiated and used arbitrarily, the rules corresponding to each agent's script must be instantiated only once per session (after which they may be used arbitrarily in the adversary's reasoning). To capture this subtle distinction, we must turn to linear logic.

3.2.1 Introduction to Linear Logic

Linear logic was introduced in 1987 by Girard ([Gir87]); since then a number of applications have demonstrated its relevance to computer science, including natural encodings of computation models such as Petri nets and Turing machines; natural language processing; compiler optimization; and concurrency (see [Lin92, WH96]). By conceptualizing linear logic as a *logic of resources*, we begin to understand the nature of the connection between linear logic and computer science.

As illustration, consider this simple example, taken from [Lin92]. We conceive of the following propositions as resources:

$$\begin{array}{rcl} D & = & \text{"One Dollar"} \\ M & = & \text{"One Pack of Marlboros"} \\ C & = & \text{"One Pack of Camels"} \end{array}$$

We may then axiomatize a simple vending machine:

$$D \Rightarrow M$$

$$D \Rightarrow C$$

meaning that if we have One Dollar we may acquire One Pack of Marlboros; similarly, if we have One Dollar we may acquire One Pack of Camels. In classical logic, we may deduce from these axioms

$$D \text{ implies } (M \text{ and } C)$$

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

or that with One Dollar we may acquire both One Pack of Marlboros and One Pack of Camels.

The absurdity of this conclusion reveals that there are in fact two types of conjunction, between which classical logic fails to distinguish. Linear logic, however, does make this distinction: one type of conjunction, written $\otimes$, intuitively means "I have both"; the other type, written &, means "I have a choice." Similarly, there are two types of disjunction: $\oplus$ is the dual of &, which means "one or the other, but I don't know which;" and $\wp$ is the dual of $\otimes$, intuitively representing a consolidation of debts. With these four connectives come four unit constants; we have

$$\begin{aligned} \bot \wp G &= G \\ \top \& G &= G \\ 1 \otimes G &= G \\ 0 \oplus G &= G \end{aligned}$$

Thus, our vending machine example may be expressed in linear logic formulae as

$$\begin{aligned} D &\multimap M \\ D &\multimap C \\ D &\multimap (M \& C) \end{aligned}$$

Linear negation is defined inductively over formulae; for all ground formulae ϕ, ϕ and $\phi^\bot$ are atomic. $(\phi^\bot)^\bot$ is defined as ϕ; negations of other formulae are defined using DeMorgan's laws. As in classical logic, implication is defined in terms of disjunction and negation; the linear implication $(\phi \multimap \psi)$ is defined as $(\phi^\bot \wp \psi)$. Quantifiers $\forall$ and $\exists$ work as usual, and can be thought of as infinite generalizations of & and $\oplus$. Finally, to model situations in which it is not necessary to count resources precisely, the logic includes the "exponentials" ! and ?. $!\phi$ intuitively means an unbounded number (including 0) of ϕ; ? is the dual of !. Thus, with the introduction of the exponentials we see that linear logic subsumes classical logic. To summarize, inference rules for linear logic (taken from [HP94]) are given in Figure 3.2; we denote this proof system by *LL*. In these sequents, each side is viewed as a multiset

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

of formulae; commas on the left are interpreted by $\otimes$; commas on the right are interpreted by $\wp$.

In fact, our protocol analysis is based on a variant of linear logic called *affine* logic, in which the more general rule of *weakening* replaces the W!-L rule:

$$\frac{\Gamma \vdash \Delta}{\Gamma, \phi \vdash \Delta}$$

That is, in essence, the adversary need not use all of its knowledge to attack a protocol.

3.2.2 Logic Programming for Linear and Affine Logic

From this discussion it becomes clear that our system of protocol analysis depends on some form of logic programming based on some fragment of affine logic. As Hodas and Miller ([HM94]) point out, not all logics appear to be appropriate as the foundation for a logic programming language. Full first-order logic, for example, is too rich to allow the "goal-directed" search that logic programming requires; but the Horn clause fragment of first-order logic is very well suited to logic programming. Similarly, full linear logic is far too rich to lend itself to logic programming, but many suitable fragments have been identified. Our goal, then, is to identify such a fragment that admits some form of logic programming, is sufficiently rich to express the "logic programs" corresponding to our analysis, and has a semantics with respect to which it is sound and complete.

Tammet [Tam94] has explored a variety of automated proof techniques for propositional linear logic; he sketches a "bottom-up" search procedure that is complete for formulae where only literals appear in the scope of ?; he also gives a "top-down" resolution method that is complete for the propositional fragment.

Andeoli ([And92]) develops a logic programming language, *LinLog*, which applies to a syntactically restricted fragment of linear logic. He shows that any linear logic formula can be "normalized" to meet the syntactic restrictions, thus showing that *LinLog* is complete.

Hodas and Miller ([HM94]) also identify a logic programming approach for a syntactic restriction of linear logic. Although this syntactic restriction does not have the full expressive powers of linear logic, Hodas and Miller are able to give a model-theoretic semantics for this fragment with respect to which their proof system is sound and complete. Their frag-

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

axiom	$\dfrac{}{\phi \vdash \phi}$	$\dfrac{\Gamma \vdash \phi, \Delta \quad \Delta', \phi \vdash \Delta'}{\Gamma, \Gamma' \vdash \Delta, \Delta'}$	cut
1-L	$\dfrac{\Gamma \vdash \Delta}{\Gamma, 1 \vdash \Delta}$	$\dfrac{}{\vdash 1}$	1-R
$L^{\perp}$	$\dfrac{}{\perp \vdash}$	$\dfrac{\Gamma \vdash \Delta}{\Gamma \vdash \perp, \Delta}$	$R^{\perp}$
0-L	$\dfrac{}{\Gamma, 0 \vdash \Delta}$	$\dfrac{}{\Gamma \vdash \top, \Delta}$	$\top$-R
$\perp$-L	$\dfrac{\Gamma \vdash \phi, \Delta}{\Gamma, \phi^{\perp} \vdash \Delta}$	$\dfrac{\Gamma, \phi \vdash \Delta}{\Gamma \vdash \phi^{\perp}, \Delta}$	$\perp$-R
$\otimes$-L	$\dfrac{\Gamma, \phi, \psi \vdash \Delta}{\Gamma, \phi \otimes \psi \vdash \Delta}$	$\dfrac{\Gamma \vdash \phi, \Delta \quad \Gamma' \vdash \psi, \Delta'}{\Gamma, \Gamma' \vdash \phi \otimes \psi, \Delta, \Delta'}$	$\otimes$-R
&-L	$\dfrac{\Gamma, \phi_i \vdash \Delta}{\Gamma, \phi_1 \,\&\, \phi_2 \vdash \Delta}$	$\dfrac{\Gamma \vdash \phi, \Delta \quad \Gamma \vdash \psi, \Delta}{\Gamma \vdash \phi \,\&\, \psi, \Delta}$	&-R
$\oplus$-L	$\dfrac{\Gamma, \phi \vdash \Delta \quad \Gamma, \psi \vdash \Delta}{\Gamma, \phi \oplus \psi \vdash \Delta}$	$\dfrac{\Gamma \vdash \phi_i, \Delta}{\Gamma \vdash \phi_1 \oplus \phi_2, \Delta}$	$\oplus$-R
$\wp$-L	$\dfrac{\Gamma, \phi \vdash \Delta \quad \Gamma', \psi \vdash \Delta'}{\Gamma, \Gamma', \phi \wp \psi \vdash \Delta, \Delta'}$	$\dfrac{\Gamma \vdash \phi, \psi, \Delta}{\Gamma \vdash \phi \wp \psi, \Delta}$	$\wp$-R
$\multimap$-L	$\dfrac{\Gamma \vdash \phi, \Delta \quad \Gamma', \psi \vdash \Delta'}{\Gamma, \Gamma' \phi \multimap \psi \vdash \Delta, \Delta'}$	$\dfrac{\Gamma, \phi \vdash \psi, \Delta}{\Gamma \vdash \phi \multimap \psi, \Delta}$	$\multimap$-R
!-L	$\dfrac{\Gamma, \phi \vdash \Delta}{\Gamma, !\phi \vdash \Delta}$	$\dfrac{!\Gamma \vdash \phi, ?\Delta}{!\Gamma \vdash !\phi, ?\Delta}$	!-R
?-L	$\dfrac{!\Gamma, \phi \vdash ?\Delta}{!\Delta, ?\phi \vdash ?\Delta}$	$\dfrac{\Gamma \vdash \phi, \Delta}{\Gamma \vdash ?\phi, \Delta}$	?-R
W!-L	$\dfrac{\Gamma \vdash \Delta}{\Gamma, !\phi \vdash \Delta}$	$\dfrac{\Gamma \vdash \Delta}{\Gamma \vdash ?\phi, \Delta}$	W?-R
C!-L	$\dfrac{\Gamma, !\phi, !\phi \vdash \Delta}{\Gamma, !\phi \vdash \Delta}$	$\dfrac{\Gamma \vdash ?\phi, ?\phi, \Delta}{\Gamma \vdash ?\phi, \Delta}$	C?-R
$\forall$-L	$\dfrac{\Gamma, \phi[x \leftarrow t] \vdash \Delta}{\Gamma, \forall x\, \phi \vdash \Delta}$	$\dfrac{\Gamma \vdash \phi[x \leftarrow y], \Delta}{\Gamma \vdash \forall x\, \phi, \Delta}$	$\forall$-R (y not free in Δ, Γ)
$\exists$-L (y not free in Γ, Δ)	$\dfrac{\Gamma, \phi[x \leftarrow y] \vdash \Delta}{\Gamma, \exists x\, \phi \vdash \Delta}$	$\dfrac{\Gamma \vdash \phi[x \leftarrow t], \Delta}{\Gamma \vdash \exists x\, \phi, \Delta}$	$\exists$-R

Figure 3.2: Linear logic proof system *LL*

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

ment, however, is too restrictive for our problem; in particular, they do not allow formulae containing ! on the left side of any sequent.

Harland and Pym ([HP94]) give a resolution-based proof strategy for sequents of the form $D \vdash G$ where D and G are elements of the sets $\mathcal{D}$ and $\mathcal{G}$, defined recursively as

$$\mathcal{D} := A \mid 1 \mid D \,\&\, D \mid D \otimes D \mid G \multimap A \mid \forall x.D \mid !\,D \mid D \parr D$$

$$\mathcal{G} := A \mid 1 \mid \bot \mid \top \mid G \otimes G \mid G \oplus G \mid G \parr G \mid G \,\&\, G \mid D \multimap G \mid \forall x.G \mid \exists x.G \mid !G \mid ?,G$$

where A ranges over atomic formulae. This fragment is clearly sufficiently expressive for our analysis, but Harland and Pym's completeness analysis is purely proof theoretic, while we desire a natural model-theoretic completeness analysis.

We derive our analysis from a fragment of linear logic given in [HM94]; we motivate the use of this fragment below. This fragment includes only the connectives $\otimes$, $\multimap$, !, and $\forall$, and allows one formula on the right-hand side of each sequent.

3.2.3 Affine Logic and Protocol Analysis

Now we may return to our system of inference rules and develop a formal statement of the power of our analysis. We wish ultimately to demonstrate two things: first, that the existence of an attack can be expressed as a formula in linear logic; and second, that our analysis is sound and complete for these formulae.

We will derive the formula corresponding to an attack through a process of successive refinement. First, we recall that an attack exists if the adversary is able to use the protocol to learn some particular message m_0. Knowledge of this message we can represent by the formula $\mathit{Knows}(m_0)$. The adversary's reasoning we will describe using a set of formulae Γ (the structure of which we develop below). Thus the presence of an attack can be expressed as $\Gamma \models \mathit{Knows}(m_0)$.

Now we elucidate the structure of Γ, which we think of as encoding the adversary's knowledge and the ways in which it may use this knowledge. We first break Γ into three parts: $\Gamma = \Gamma_1 \otimes \Gamma_2 \otimes \Gamma_3$. Γ_1 represents the adversary's initial knowledge, Γ_2 represents the message-structure rules, and Γ_3 represents the protocol specific rules. Note that the use of the connective $\otimes$ means that the adversary has all of these at its disposal.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Γ_1 consists of all primitive messages containing only session-independent information, with the exception of the long-term keys K_A and K_B. This set includes formulae such as $M(A)$, $Knows(k(C))$, etc. (recall that the adversary may know some long-term keys by functioning as a legitimate agent in some sessions). Since the adversary may use this knowledge arbitrarily often in its reasoning, Γ_1 will have the form $!\phi_1 \otimes \cdots \otimes !\phi_k$.

Since Γ_2 reflects a *set* of inference rules, it is itself another $\otimes$-conjunction, where each conjunct represents an inference rule. The inference rule

$$\frac{k \quad e(x,k)}{x},$$

for example, suggests an implication of the form $((Knows(k) \otimes Knows(e(x,k))) \multimap Knows(x))$. Implicit in the rule, of course, is a universal quantification of variables, and, as we said above, these rules may be used arbitrarily often with arbitrary instantiation, so this inference rule will be expressed as $!\forall x, k((Knows(k) \otimes Knows(e(x,k))) \multimap Knows(x))$.

Finally, Γ_3 represents the protocol-specific rules. Again, subformulae will have essentially the form $(\phi_1 \otimes \cdots \otimes \phi_k) \multimap \psi$. As we observed above, each agent will only emit messages corresponding to one variable instantiation per session; the adversary may then use these messages arbitrarily often. Thus, Γ_3 will contain subformulae of the form $\forall \vec{x} !(\sigma_1 \otimes \cdots \otimes \sigma_n)$ for each session index i (that is, i is instantiated as a particular index in each copy of this implication) and for each agent, where each σ_k is the implication corresponding to one inference rule in the agent's "script".

Thus, Γ captures the adversary's capacity to reason about the protocol. In order to determine whether the adversary can learn some message m from the protocol, it remains to determine whether $\Gamma \models Knows(m)$. In the next section, we present a proof-theoretic argument that the resolution-based deductive system in our application is sound and complete for formulae of this form; in section 3.4 we give a model theoretic semantics for this system which we demonstrate is sound and conjecture to be complete.

Observe that the restricted form of the sequents we wish to prove allows us to consider a restricted set of proof rules given in Figure 3.3. These proof rules are based on a fragment from [HM94], modified for affine logic to include general weakening); ϕ_i and ψ are arbitrary formulae over the connectives $\otimes$, $\multimap$, $!$, and $\forall$.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

identity	$\dfrac{}{\phi \vdash \phi}$	$\dfrac{\Delta \vdash \theta}{\Delta, \phi \vdash \theta}$	weakening
$\multimap$-L	$\dfrac{\Delta_1 \vdash \phi \quad \Delta_2, \psi \vdash \theta}{\Delta_1, \Delta_2, \phi \multimap \psi \vdash \theta}$	$\dfrac{\Delta_1 \vdash \phi \quad \Delta_2 \vdash \psi}{\Delta_1, \Delta_2 \vdash \phi \otimes \psi}$	$\otimes$-R
!-contraction	$\dfrac{\Delta, !\phi, !\phi \vdash \theta}{\Delta, !\phi \vdash \theta}$	$\dfrac{\Delta, \phi \vdash \theta}{\Delta, !\phi \vdash \theta}$	!-dereliction
$\forall$-L	$\dfrac{\Delta, \phi[x \leftarrow t] \vdash \theta}{\Delta, \forall x.\phi \vdash \theta}$		

Figure 3.3: A relevant fragment of affine logic

3.3 Toward Resolution in Affine Logic

Cerrito [Cer92] presents a proof system *LLH* for linear logic that is equivalent to the system given in Figure 3.2. She observes that the effectiveness (and efficiency) of logic programming is due in large part to the utilization of *unification*, rather than arbitrary existential instantiation, to reduce nondeterminism and complexity in proof search. Motivated by this observation, she modifies the "standard" proof rules for linear logic to incorporate unification in the place of general instantiation. This modification is based on the introduction of *witness variables*, denoted $W_1, W_2, \ldots$, which instantiate quantified variables. Thus, the proof rules involving quantifiers are modified to utilize witness variables; at the leaves of the proof tree, the identity axioms are modified to include instantiations of the witness variables. Specifically, the $\forall$-L rule is modified as follows:[1]

$$\frac{\Delta, \phi[x \leftarrow W_i] \vdash \theta}{\Delta, \forall x\, \phi \vdash \theta}$$

where $\phi[x \leftarrow W_i]$ denotes the formula ϕ with all occurrences of x replaced by the witness variable W_i. In this rule, W_i is "fresh"; that is, it does not appear in the lower sequent. Then the identity rule is modified as follows:

$$\frac{W_1 = t_1 \ldots W_n = t_n}{\phi \vdash \phi'}$$

where $W_1 \ldots W_n$ are among the witness variables in ϕ, ϕ', and $W_1 = t_1 \ldots W_n = t_n$ represents a substitution which is the most general unifier of ϕ and ϕ'. An *LLH* proof tree

[1] Cerrito gives proof rules for the right-handed sequent calculus; we give the equivalent two-sided formulation.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

represents an *LLH*-proof if the "partial answers" given by the witness variable equations at the leaves of the tree have a solution (i.e. they unify).

Since *LLH* is equivalent to *LL* (in a strong sense: the application of a proof rule in one proof system corresponds to exactly one application of the same rule in the other system), the restricted version of *LL* given in Figure 3.3 is equivalent to the similarly restricted fragment of *LLH*. It now remains to show that the resolution-based system used in our application is sound and complete with respect to this proof system.

3.3.1 Resolution

We give a resolution system in Figure 3.4. We include the usual axiom and weakening rules. The notation in the resolution rules is as follows: $!\Gamma$ denotes a multiset of formulae of the form $!\phi$ or $!\forall\bar{x}((\otimes_i\phi_i) \multimap \psi)$; $\forall\Gamma_i$ denotes a multiset of formulae of the form $\forall\bar{x}\,!((\bigotimes_k(\bigotimes_{i_k}\phi_{i_k}) \multimap \psi_k))$; $\forall\Gamma_j[\mu_j]$ denotes a formula of the form $!(\bigotimes_k(\bigotimes_{i_k}\phi_{i_k}))[\mu_j]$, where μ_j is a substitution for quantified variables. All ϕ, ψ, θ, etc. are atomic.

$$\frac{}{\theta \vdash \theta}$$

$$\frac{\Gamma \vdash \theta}{\Gamma, \phi \vdash \theta}$$

$$\frac{!\Gamma, \forall\Gamma_j[\mu_j], \forall\Gamma_n, !\forall\bar{x}((\otimes_i\phi_i) \multimap \psi) \vdash \phi_n[\mu] \quad \text{for } 1 \le n \le k}{!\Gamma, \forall\Gamma_j, \forall\Gamma_1, \ldots\forall\Gamma_k, !\forall\bar{x}((\otimes_i\phi_i) \multimap \psi) \vdash \theta} \quad \mu = \mathrm{mgu}(\theta, \psi)$$

$$\frac{!\Gamma, \forall\Gamma_j[\mu_j], \forall\Gamma_n, !((\bigotimes_k(\bigotimes_{i_k}\phi_{i_k}) \vdash \phi_n[\mu] \quad \text{for } 1 \le n \le i_k}{!\Gamma, \forall\Gamma_j, \forall\Gamma_1, \ldots\forall\Gamma_k, \forall\bar{x}\,!((\bigotimes_k(\bigotimes_{i_k}\phi_{i_k}) \multimap \psi_k)) \vdash \theta} \quad \mu = \mathrm{mgu}(\theta, \psi_k)$$

Figure 3.4: Resolution rules for restricted linear logic

The resolution rules themselves behave similarly to classical resolution: if the goal, θ, unifies with the head of some implication (ψ or ψ_k, in Figure 3.4), it suffices to prove each formula in the antecedent subject to the same unification. The distinction between the rules arises from the interaction between the ! and $\forall$ connectives. If θ unifies with some ψ enclosed in $!\forall\bar{x}$, then this $!\forall\bar{x}$ formula remains available to prove the antecedent (due to !-contraction). If θ unifies with some ψ_k enclosed in $\forall\bar{x}\,!$, only that formula with the fixed instantiation corresponding to μ remains available to prove the antecedent. Additionally,

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

if any other $\forall\bar{x}$!-formula (that is, some $\forall\Gamma_j$) is instantiated to prove some antecedent, that instantiation must agree across the proofs of all the formulae in the antecedent.

We now show that this proof system is sound and complete with respect to the *LLH*-based proof system discussed above.

3.3.2 *LLH*-Soundness

We show that any resolution proof can be converted to an *LLH*-proof. We proceed by induction on the height of the proof. The axiom and weakening rules are straightforward. For convenience, we observe that the !-contraction rule can be applied repeatedly to a formula $!\phi$ to produce k copies of this formula; we denote these copies in the aggregate by $!\phi^k$. Thus, an application of the $!\forall$-resolution rule can be converted into an *LLH*-subproof as follows:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{!\Gamma, \forall\Gamma_j[\overline{x_j}\leftarrow\overline{W_j}], \forall\Gamma_n, !\forall\bar{x}((\otimes_i\phi_i)\multimap\psi)\vdash\phi_n[\overline{x}\leftarrow\overline{W}] \quad \text{for } 1\le n\le k \qquad \overline{W}=\bar{t}}{!\Gamma^k, \forall\Gamma_j[\overline{x_j}\leftarrow\overline{W_j}]^k, \forall\Gamma_1,\ldots\forall\Gamma_k, !\forall\bar{x}((\otimes_i\phi_i)\multimap\psi)^k\vdash(\otimes_i\phi_i)[\overline{x}\leftarrow\overline{W}] \qquad \psi[\overline{x}\leftarrow\overline{W}]\vdash\theta}}{!\Gamma, \forall\Gamma_j, \forall\Gamma_1,\ldots\forall\Gamma_k, !\forall\bar{x}((\otimes_i\phi_i)\multimap\psi)^k, ((\otimes_i\phi_i)\multimap\psi)[\overline{x}\leftarrow\overline{W}]\vdash\theta}}{!\Gamma, \forall\Gamma_j, \forall\Gamma_1,\ldots\forall\Gamma_k, !\forall\bar{x}((\otimes_i\phi_i)\multimap\psi)^k, \forall\bar{x}((\otimes_i\phi_i)\multimap\psi)\vdash\theta}}{!\Gamma, \forall\Gamma_j, \forall\Gamma_1,\ldots\forall\Gamma_k, !\forall\bar{x}((\otimes_i\phi_i)\multimap\psi)^k, !\forall\bar{x}((\otimes_i\phi_i)\multimap\psi)\vdash\theta}}{!\Gamma, \forall\Gamma_j, \forall\Gamma_1,\ldots\forall\Gamma_k, !\forall\bar{x}((\otimes_i\phi_i)\multimap\psi)^{k+1}\vdash\theta}}{!\Gamma, \forall\Gamma_j, \forall\Gamma_1,\ldots\forall\Gamma_k, !\forall\bar{x}((\otimes_i\phi_i)\multimap\psi)\vdash\theta}$$

where $\overline{W}=\bar{t}$ represents a unifier for ψ and θ.

Applications of the $\forall\bar{x}$-resolution rule can be transformed similarly. Finally, since the substitutions $[\overline{x}\leftarrow\overline{W}]$ agree across all branches of the tree, this is indeed a valid *LLH*-proof.

3.3.3 *LLH*-Completeness

We will show that any *LLH*-proof can be transformed into a resolution proof. We will proceed by induction on the number of $\multimap$-L rules used in the shortest proof of $\Gamma\vdash\theta$. We begin by observing some useful permutability properties of the proof rules. First, the $\multimap$-L rule can be moved above the weakening, !-dereliction, $\forall$-L, and !-contraction rules in the sense illustrated by the following example: a subproof of the form

$$\dfrac{\dfrac{\Delta_1\vdash\phi}{\Delta_1,\delta\vdash\phi} \qquad \dfrac{\Delta_2,\psi\vdash\theta}{\Delta_2,\epsilon,\psi\vdash\theta}}{\Delta_1,\Delta_2,\delta,\epsilon,\phi\multimap\psi\vdash\theta}$$

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

can be transformed into a subproof of the form

$$\dfrac{\dfrac{\Delta_1 \vdash \phi \qquad \Delta_2 \psi \vdash \theta}{\Delta_1, \Delta_2, \phi \multimap \psi \vdash \theta}}{\Delta_1, \Delta_2, \delta, \epsilon, \phi \multimap \psi \vdash \theta}$$

That is, the weakening rules applied above the $\multimap$-L rule can be combined and applied below the $\multimap$-L rule. The remaining rules (!-dereliction, $\forall$-L, and !-contraction) can be permuted similarly.

Second, the $\otimes$-R rule can be permuted below the weakening, !-dereliction, $\forall$-L, and !-contraction rules. For example, a subproof of the form

$$\dfrac{\dfrac{\Delta_1, !\delta, !\delta \vdash \phi \qquad \Delta_2, !\epsilon, !\epsilon \vdash \psi}{\Delta_1, !\delta, !\delta, \Delta_2, !\epsilon, !\epsilon \vdash \phi \otimes \psi}}{\Delta_1, !\delta, \Delta_2, !\epsilon \vdash \phi \otimes \psi}$$

can be transformed into a subproof of the form

$$\dfrac{\dfrac{\Delta_1, !\delta, !\delta \vdash \phi}{\Delta_1, !\delta \vdash \phi} \qquad \dfrac{\Delta_2, !\epsilon, !\epsilon \vdash \psi}{\Delta_2, !\epsilon \vdash \psi}}{\Delta_1, !\delta, \Delta_2, !\epsilon \vdash \phi \otimes \psi}$$

and the other rules can be permuted similarly.

Now we proceed with our analysis of completeness. For the base case, if an *LLH*-proof of $\Gamma \vdash \theta$ contains no occurrences of the $\multimap$-L rule, it must be the case that Γ contains a formula of the form $!\theta$. In this case, the resolution proof of $\Gamma \vdash \theta$ simply uses the weakening rule and the axiom rule.

For the induction, we consider the last occurrence of a $\multimap$-L rule in a proof of $\Gamma \vdash \theta$. In this case, the lower sequent of the rule must be of the form

$$\Gamma''', (\otimes_i \phi_i \multimap \psi)[\overline{x} \leftarrow \overline{W}] \vdash \theta$$

There are two similar cases corresponding to the originating formula of the implication. If it was prefixed by $!\forall \overline{x}$, then the subproof below this sequent has the form

$$\dfrac{\dfrac{\dfrac{\Gamma''', (\otimes_i \phi_i \multimap \psi)[\overline{x} \leftarrow \overline{W}] \vdash \theta}{\vdots}}{\Gamma'', \forall \overline{x}(\otimes_i \phi_i \multimap \psi) \vdash \theta}}{\vdots}$$
$$\Gamma', !\forall \overline{x}(\otimes_i \phi_i \multimap \psi) \vdash \theta$$

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

where $\Gamma' = \Gamma - \{!\forall\bar{x}(\otimes_i\phi_i \multimap \psi)\}$; Γ' is derived from Γ'' by applications of the weakening, !-dereliction, $\forall$-L, and !-contraction rules; and Γ'' is similarly derived from Γ''' via further applications of those rules.

The sequent $\Gamma''', (\otimes_i\phi_i \multimap \psi)[\bar{x}\leftarrow\overline{W}] \vdash \theta$ is in turn the root of a subproof of the form

$$\frac{\dfrac{\Gamma_1 \vdash \phi_1[\bar{x}\leftarrow\overline{W}] \ldots \Gamma_k \vdash \phi_k[\bar{x}\leftarrow\overline{W}]}{\Gamma_1 \ldots, \Gamma_k \vdash \otimes_i\phi_i[\bar{x}\leftarrow\overline{W}]} \quad \dfrac{\Xi}{\Gamma_{k+1}, \psi[\bar{x}\leftarrow\overline{W}] \vdash \theta}}{\Gamma''', (\otimes_i\phi_i \multimap \psi)[\bar{x}\leftarrow\overline{W}] \vdash \theta}$$

where $\Gamma_1 \cup \cdots \cup \Gamma_{k+1} = \Gamma'''$.

To see that that this subproof can be transformed into a resolution subproof, we first observe that if $\psi[\bar{x}\leftarrow\bar{W}] \not\vdash \theta$ then it must be the case that $\Gamma_{k+1} \vdash \theta$. But then there would be a shorter proof of $\Gamma \vdash \theta$. Thus it must be the case that $\psi[\bar{x}\leftarrow\overline{W}] \vdash \theta$; i.e. that the substitution $[\bar{x}\leftarrow\overline{W}]$ represents a unification of ψ and θ, and Γ_{k+1} is unnecessary. Hence the subproof

$$\frac{\Xi}{\Gamma_{k+1}, \psi[\bar{x}\leftarrow\overline{W}] \vdash \theta}$$

has the form

$$\frac{\overline{W} = \bar{t}}{\psi[\bar{x}\leftarrow\overline{W}] \vdash \theta}$$

Additionally, we observe that we may further normalize the structure of this subproof. Specifically, we require that if some formula $!\phi$ appears on the left of a sequent below $\Gamma''', (\otimes_i\phi_i \multimap \psi)[\bar{x}\leftarrow\bar{W}] \vdash \theta$, then $!\phi^k$ appears in Γ''' and $!\phi$ appears (exactly once) in each Γ_i. That is, we apply !-contraction as necessary below $\Gamma''', (\otimes_i\phi_i \multimap \psi)[\bar{x}\leftarrow\bar{W}] \vdash \theta$. If for any Γ_i, it happens that $\Gamma_i - \{!\phi\} \vdash \phi_i[\bar{x}\leftarrow\overline{W}]$ (i.e. $!\phi_i$ is not necessary to the proof), then weakening may be applied above this sequent (in effect, this forces all "relevant" applications of weakening to the leaves of the tree, directly below the axiom rules).

The resulting subproof can then be replaced by one application of the $!\forall$-resolution rule. Since the constraints on *LLH*-proofs require that all $W_i = T_i$ equations at the leaves unify, we are certain to have an instance of the $!\forall$-resolution rule in which the unifiers μ_j and μ agree across all branches of the proof.

The other case, in which $(\otimes_i\phi_i \multimap \psi)$ originated from a formula enclosed in $\forall\bar{x}!$, proceeds similarly.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Thus, resolution is sound and complete for this fragment of affine logic. In the next section, we present a model-theoretic semantics for this fragment. We demonstrate that it is sound for the proof system in Figure 3.3 and conjecture that is complete.

3.4 An Intended Model

Below, we present a Kripke-like model-theoretic semantics for the fragment of affine logic given in Figure 3.3. We motivate our semantics by recalling that affine logic can be naturally regarded as a logic of resources; in our case, the adversary's resources are ground atomic formulae such as $Knows(N_i)$ in conjunction with a set of inference rules. Reasoning using these resources is constrained by the number of available instances of each resource. Thus, rather than assigning simple Boolean values to each atomic formula, a model must indicate the availability of the resource represented by the formula. For this purpose, we introduce the idea of an *intensity structure*. An intensity structure (or just "a structure") A is a mapping I^A from ground atomic formulae to "intensities" in $\mathbf{N} \cup \{\infty\}$. We define the *sum* of two structures $A + B$ to be the structure C where $I^C(\phi) = I^A(\phi) + I^B(\phi)$ for all ground atomic ϕ. (We use the usual idea that $x + \infty = \infty + x = \infty$.) A structure B is a *reduction* of structure A if $I^B(\phi) \leq I^A(\phi)$ for all ground atomic ϕ.

To fully capture reasoning in affine logic, we must also express the ways in which the reasoning process consumes resources. That is, we can regard applications of the proof rules as performing transformations on intensity structures. This perspective provides further insight into the utility of linear logic for modeling computation: Gurevich's Abstract State Machine formalism ([Gur94]), for example, models computation as a sequence of transformation of algebraic structures. Thus, to understand how affine logic formulae may be modeled by an intensity structure, we must express how proofs of these formulae depend on (and alter) the structure.

For a given structure A, we define K_A to be the smallest collection of structures that contains A and is "closed downward"; that is, K_A contains all reductions of A.

We may now proceed to give interpretations for formulae: we first define the formulae modeled by the pair K, B (where K is a collection of structures and B is a structure), then define the formulae modeled by an intensity structure A.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

For ground atomic ϕ, $K, B \models \phi$ iff $I^B(\phi) > 0$.

$K, B \models \phi \otimes \psi$ iff there are structures $C, D \in K$ such that $B = C + D$, $K, C \models \phi$, and $K, D \models \psi$.

$K, B \models \phi \multimap \psi$ iff for every $C \in K$ such that $K, C \models \phi$ and $B + C \in K$, it is the case that $K, B + C \models \psi$.

$K, B \models !\phi$ iff $K, B \models \bigotimes_k \phi$ for all k.

$K, B \models \forall x \phi(x)$ iff for every term t, $K, B \models \phi[x \leftarrow t]$.

We define $A \models \Gamma$ iff $K_A, A \models \Gamma$.

We check soundness of rules w.r. to this model.

identity. Obvious.

$\multimap$-L. Suppose we have $K, A \models \Delta_1$, $K, B \models \Delta_2$, and $K, C \models \phi \multimap \psi$. By the first upper sequent, $K, A \models \phi$, so $K, A + C \models \phi, \phi \multimap \psi$. We show that $K, A + C \models \psi$ (and hence that $K, A + B + C \models \Delta_2, \phi$, from which we conclude that $K, A + B + C \models \theta$, as desired). Toward contradiction, we assume $K, A + C \not\models \psi$. Then in particular $K, C \not\models \psi$. But since $K, C \models \phi \multimap \psi$ it must be that for every $C' \in K$ such that $K, C' \models \phi$ and $C' + C \in K$, it is the case that $K, C' + C \models \psi$. Choose $C' = A$; we conclude that $K, A + C \models \psi$. Contradiction. Thus $K, A + B + C \models \theta$.

$\otimes$-R. Suppose $K, A \models \Delta_1, \Delta_2$. Then A splits into B, C such that $K, B \models \Delta_1$ and $K, C \models \Delta_2$. By the upper sequents, $K, B \models \phi$ and $K, C \models \psi$. Thus, $K, A \models \phi \otimes \psi$.

weakening. Suppose $K, A \models \Delta, \phi$. Then A splits into B, C such that $K, B \models \Delta$ and $K, C \models \phi$. By the upper sequent, $K, B \models \theta$, so certainly $K, B + C \models \theta$; thus $K, A \models \theta$.

!-contraction. Suppose $K, A \models \Delta, !\phi$. Then A can be split into B,C so $K, B \models \Delta$ and $K, C \models !\phi$. Then $C \models \otimes_k \phi$ for any k. Thus for any i, j, C can be split into D and E so that $K, D \models \otimes_i \phi$ and $K, E \models \otimes_j \phi$. Thus $K, B + D + E \models \Delta, !\phi, !\phi$, (and $A = B + D + E$) so by the upper sequent $A \models \theta$.

!-dereliction. Suppose $K, A \models \Delta, !\phi$. Then A can be split into B,C so $K, B \models \Delta$ and $K, C \models !\phi$. Then $K, C \models \phi$ (choose $k = 1$). So $k, A = B + C \models \Delta, \phi$, so by the upper sequent $A \models \theta$.

$\forall$-L. Suppose $K, A \models \Delta, \forall x \phi$. Then A can be split into B,C so $K, B \models \Delta$ and $K, C \models \forall x \phi$. Then $K, C \models \&_t \phi[x \leftarrow t]$. Thus for any t, $K, C \models \phi[x \leftarrow t]$. So we have $K, A = B + C \models$

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

$\Delta, \phi[x \leftarrow t]$, so by the upper sequent $K, A \models \theta$.

We conjecture that this model-theoretic semantics is, in addition, complete for this fragment. From this conjecture we may conclude that resolution is complete as well.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

CHAPTER 4

Implementation and Results

In this chapter we give an overview of our implementation of the analysis method described earlier. We also present results from our analysis of some well-known protocols.

4.1 Implementation

We have implemented this technique as a special-purpose theorem prover written in *Standard ML of New Jersey*. The theorem prover operates on input consisting of a protocol specification and a goal message. As output, it produces a list of attacks — sequences of messages that demonstrate how the adversary may use the protocol to learn the goal message.

The theorem prover consists of two components: the *preprocessor*, which converts the protocol specification into a more efficient internal representation, and the *proof generator*, which produces a proof tree from which the list of attacks can be straightforwardly extracted. Each of these components is described more fully below.

4.1.1 The Preprocessor

The preprocessor allows input specifications to be written in a natural and intuitive way, even though the proof generator operates on specifications such as those given in Figure 2.3. The input specification has three sections: 1) a list of the atomic messages recognized by each agent, 2) a list of those timestamps (if any) that each agent will check for currency, and 3) the protocol specification given in a form similar to that seen in Figure 2.1. See Figure 4.1 for the specification of the Needham-Schroeder protocol given in this form.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
A & A,k(A),B,Ni,Pi &
B & B,k(B),Mi,Qi &
S & S,Ki,k(A),k(B) &

&

&

A S [A,B,Ni] &
S A e([Ni,B,Ki,e([Ki,A],k(B))],k(A)) &
A B e([Ki,A],k(B)) &
B A e(Mi,Ki) &
A B e(f(Mi),Ki) &
```

Figure 4.1: Specification of the Needham-Schroeder protocol

The preprocessor then converts this specification into a set of inference rules. First, it partitions the protocol into the sub-protocols consisting of all messages sent by each agent; each message m is associated with the message m' that must be received in order for m to be sent (the preprocessor automatically adds to the protocol messages corresponding to P_i and Q_i). The m' are then augmented to reflect the fact that a message will not be sent by an agent unless the agent has received all earlier expected messages. Finally, each of these message pairs is converted into an inference rule by replacing unrecognized sub-messages with variables. Figure 2.3 shows the result of this process for the Needham-Schroeder protocol.

4.1.2 The Proof Generator

The proof generator takes as input the list of inference rules produced by the preprocessor and a set of messages to be learned by the adversary. It produces as output an **AND-OR** tree representing the possible attacks on the goal message set.

Intuitively, the tree is structured as follows: at the root is an **AND** node labeled by the initial goal set S representing the message(s) the adversary wishes to learn. We then select a message to "attack", creating subtrees corresponding to each possible selected message; thus there is an attack on S if there is an attack based on selecting any message. Since our selection may affect variable substitutions we must consider all cases. Once we select a particular message m, we then determine whether we can attack it through the protocol. We

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

search the protocol-specific rules trying to unify m with sub-messages of the consequents. Such a unifying sub-message represent a possible attack on m — if we can extract this sub-message from the consequent (for example, we may need to perform decryption) and learn the antecedent, we can learn m. Thus, for each such sub-message m_r we first compute the most general unifier, μ, of m and m_r. We then apply μ to the antecedent and consequent, as well as to the other messages in the subset from which m was selected. The message-structure rules tell us what messages we need to learn to extract m from the consequent; we union this set with the unified antecedent and other messages from the subset. Each of these results (one per unifying sub-message) is a child node; its subtrees are determined by repeating this process on the our newly derived goal set.

More precisely, the proof generator executes in the following resolution-based cycle:

1. Partition the given set S into a set $\mathcal{S}$ of subsets of S such that no subsets have any variables in common. (This is done purely for the sake of efficiency.) Create an **AND** node labeled by S; corresponding to each element of $\mathcal{S}$ is a subtree as generated by the next step.

2. For each message m in the given set S, denote the set $S - \{m\}$ by $\overline{m}$. Create an **OR**-node labeled by S; corresponding to each $(m, \overline{m})$ pair is a subtree as generated by the next step.

3. Given an $(m, \overline{m})$ pair, create an **OR**-node labeled by $\{m\} \cup \overline{m}$. Construct subtrees as follows: Search the list of inference rules for those having a consequent with some sub-message which unifies with m. For each such sub-message m_r in each such rule, apply the most general unifier of m and m_r to the antecedent, consequent, and to $\overline{m}$. Denote these results, respectively, by A, C, and $\overline{m}'$. Denote by C_m the set of messages necessary, as determined by the message-structure inference rules, to learn m from C. Form the set $S = A \cup C_m \cup \overline{m}'$; create a subtree of $(m, \overline{m})$ labeled by S as generated by applying step 1 to S. Additionally, create the set M by decomposing m into submessages by one application of the appropriate message-structure rules. Create a subtree labeled by $M \cup \overline{m}$ as generated by applying step 1 to this set.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

The cycle terminates under three conditions on steps 1 and 2. First, if S contains only session-independent messages, create a node labeled by S with no subtrees; this corresponds to a successful attack. Second, if S has already appeared in the current branch or S contains information the adversary will never acquire (such as K_A),[1] create an empty tree; this corresponds to a failed attack. Third, if S has already appeared to the left of the current node, then if it has a nonempty set of subtrees, copy that set, otherwise proceed as above.

We illustrate this process for the Needham-Schroeder protocol by considering the attack on Q_i. Figure 4.2 shows part of the AND-OR tree generated by this algorithm. Since Q_i is a singleton set, steps 1 and 2 are vacuous; at step 3 the protocol rule corresponding to Q_i matches, generating the next node in the tree. We then generate two child nodes, corresponding to $e(f(Mi), m)$ and $e([m, a], k(B))$. One protocol rule applies to each of them, and we perform step 3 in each case. We then repeat this process, selecting a message (indicated to the left of the vertical dashed line in each node) and searching for protocol rules until the branch terminates at either an attack, a previously reached node, or a node that is impossible for the adversary to learn.

4.2 Results

In this section we catalog the results of our analysis of a collection of protocols; some of these protocols have been well-studied in the analysis literature; others are protocols that have been used in real applications. In each subsection, we give the protocol in standard notation and note the results of our analysis.

4.2.1 Needham-Schroeder

As discussed in Chapter 2, this protocol has a well-known flaw that allows the adversary to learn Q_i, the secret information sent out by A at the conclusion of the protocol. Our methodology uncovers this flaw, as was also discussed above. Similarly, the adversary is able to masquerade as agent A, since in this example the adversary needs to be able to

[1] In general, it may be possible for the adversary to learn K_A if, for example, it is sent as a submessage of the protocol. We exclude such poorly-designed protocols from our analysis, but note that it is straightforward to extend the analysis by "pre-processing" the protocol to test whether the adversary can learn some long-lived key from it.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

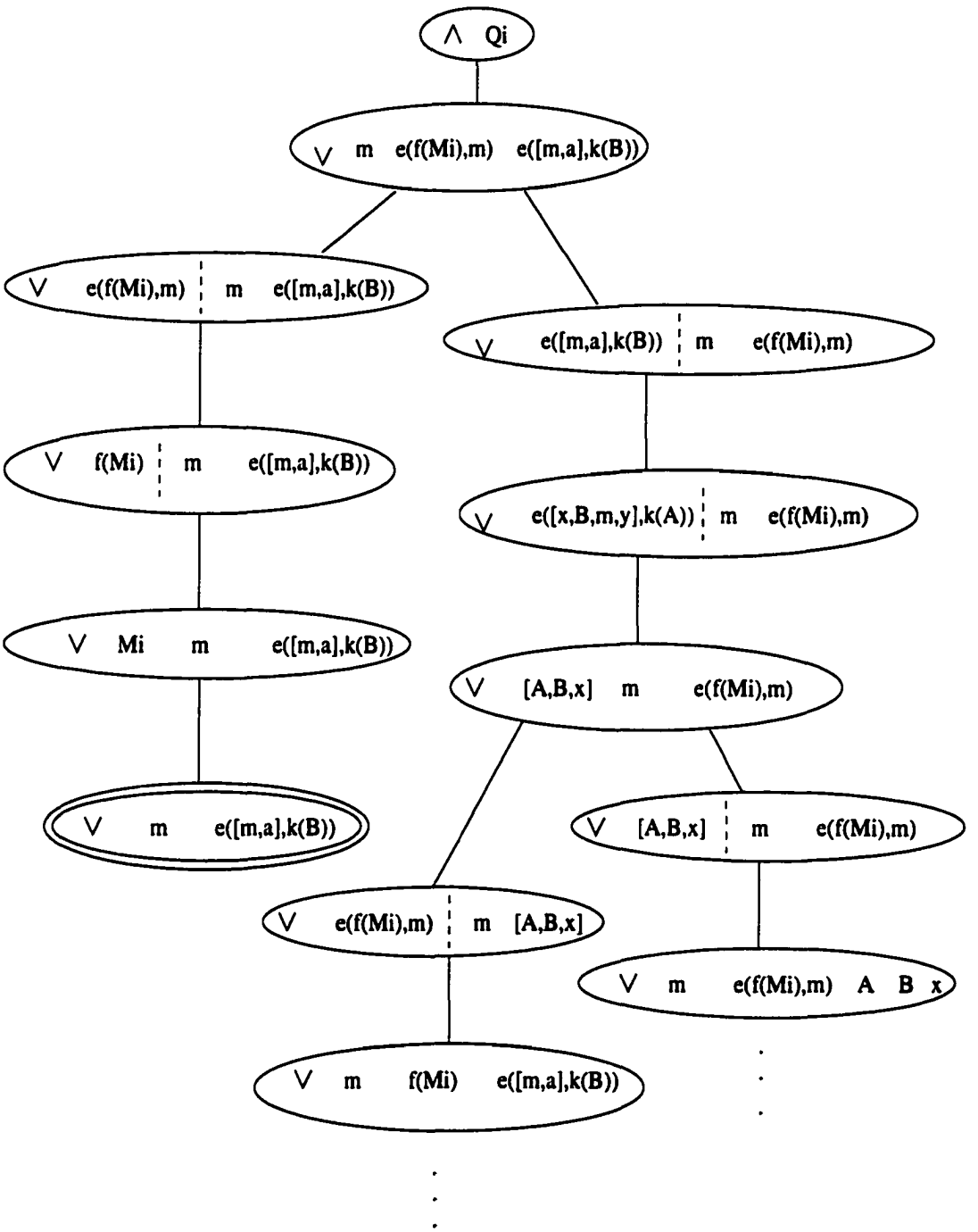


Figure 4.2: The Needham-Schroeder proof tree for Q_i

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

1. $A \rightarrow S$: A, B, N_i
2. $S \rightarrow A$: $\{N_i, B, K_i, \{K_i, A\}_{K_B}\}_{K_A}$
3. $A \rightarrow B$: $\{K_i, A\}_{K_B}$
4. $B \rightarrow A$: $\{M_i\}_{K_i}$
5. $A \rightarrow B$: $\{M_i - 1\}_{K_i}$

Figure 4.3: The Needham-Schroeder protocol

produce exactly the same messages to accomplish these two attacks.

4.2.2 Otway-Rees

1. $A \rightarrow B$: $I, A, B, \{N_i, I, A, B\}_{K_A}$
2. $B \rightarrow S$: $I, A, B, \{N_i, I, A, B\}_{K_A}, \{M_i, I, A, B\}_{K_B}$
3. $S \rightarrow B$: $I, \{N_i, K_i\}_{K_A}, \{M_i, K_i\}_{K_B}$
4. $B \rightarrow A$: $I, \{N_i, K_i\}_{K_A}$

Figure 4.4: The Otway-Rees protocol

Our system verifies that there is no attack on this protocol; other systems have produced this result.

4.2.3 "Optimized" Otway-Rees

1. $A \rightarrow B$: $N_i, A, B, \{N_i, A, B\}_{K_A}$
2. $B \rightarrow S$: $N_i, A, B, \{N_i, A, B\}_{K_A}, M_i, \{N_i, A, B\}_{K_B}$
3. $S \rightarrow B$: $N_i, \{N_i, K_i\}_{K_A}, \{M_i, K_i\}_{K_B}$
4. $B \rightarrow A$: $N_i, \{N_i, K_i\}_{K_A}$

Figure 4.5: The "optimized" Otway-Rees protocol

As a result of their analysis of the Otway-Rees protocol, Burrows et al. [BAN90] conclude that (1) the use of an "index number" distinct from A's nonce N_i is unnecessary and (2) the encryption of B's nonce M_i in message 2 is unnecessary. This changes give us the "optimized" protocol given in Figure 4.5. As [MB93] point out, the small changes made to this protocol make it vulnerable to an attack in which the adversary masquerades as agent A. See Figure 4.6 for the result of our analysis of the modified protocol.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

$$x, A, B, y, z, e([M_i, K_i], k(B)), e(R_i, K_i)$$

$$e([M_i, K_i], k(B)), e(R_i, K_i)$$

$$e([M_i, K_i], k(B)), K_i, R_i \qquad [a]$$

$$e([M_i, K_i], k(B)), e([u, K_i], k(a)), k(a) \qquad [B.2]$$

$$x, a, B, e([x, a, B], k(a)), M_i, e([x, a, B], K(B)), k(a) \qquad [S.1]$$

$$x, a, B, e([x, a, B], k(a)), k(a) \qquad [B.1]$$

$$x, e([x, a, B], k(a)), k(a)$$

Figure 4.6: Analysis of Otway-Rees protocol

To achieve the masquerade, the adversary must be able to produce all the messages received by B: $[x, a, B, y]$ (the first message received by B), $[x, z, \{M_i, K_i\}_{K_B}]$, and $\{R_i\}_{K_i}$, where R_i is the adversary's bogus version of P_i. The variables can be replaced by any arbitrary message, and the agent identifiers are known to the adversary, so it suffices for the adversary to produce $\{M_i, K_i\}_{K_B}$ and $\{R_i\}_{K_i}$. $\{R_i\}_{K_i}$ can be learned if the adversary knows K_i and R_i; since the adversary generates R_i it suffices for it to know K_i. It can learn K_i from the protocol (using message 3) by learning some message $\{u, K_i\}_{K_a}$ and the key K_a . Now, the adversary can learn $\{M_i, K_i\}_{K_B}$ and $\{u, K_i\}_{K_a}$ together if it can produce message 2: $[x, a, B, \{x, a, B\}_{K_a}, M_i, \{x, a, B\}_{K_B}]$. Then, it can learn M_i and $\{x, a, B\}_{K_B}$ by producing message 1: $[x, a, B, y]$.

Thus, the adversary needs the following messages to mount this attack: (1) a message of the form $\{N_x, C, B\}_{K_C}$, with knowledge of K_C and N_x — the adversary can produce this if it has a "legitimate" role as agent C, and (2) a message of the form $\{N_x, C, B\}_{K_B}$ — the adversary can produce this simply by initiating a session with B in its role as C (this message corresponds to B's response to that initiation). The adversary can then initiate a new session with B and intercept B's response to S. It then sends a message containing $\{N_x, C, B\}_{K_C}$ and B's response to S; from S's response the adversary will learn K_i. It

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

can then forward $\{M_i, K_i\}_{K_B}$ to B and follow it with some bogus information encrypted under the new K_i. Note that this attack depends only on session-independent information. Further, our analysis demonstrates that this is the only attack on this protocol.

4.2.4 Bellare-Rogaway 3PKD

1. $A \rightarrow B$: A, N_i
2. $B \rightarrow S$: A, B, N_i, M_i
3. $S \rightarrow A$: $\{K_i\}_{K_A}, \langle A, B, N_i, \{K_i\}_{K_A}\rangle_{K_A}$
4. $S \rightarrow B$: $\{K_i\}_{K_B}, \langle A, B, M_i, \{K_i\}_{K_B}\rangle_{K_B}$

Figure 4.7: The Bellare-Rogaway 3PKD protocol

This protocol is presented in [BR95]; it uses cryptographic hash to reduce the size of the protocol. A starts the protocol by generating a nonce N_i and sending it to B; B then generates its own nonce M_i and sends A, B, N_i, and M_i to S. S then generates the session key K_i and produces messages for A and B. These messages consist of the new session encrypted under the agent's key, together with a cryptohash "authenticator" containing the agent's nonce and the encrypted session key. When an agent receives this message, it computes the same cryptohash (note that the hash is noninvertible) and compares it with the received version. If they match, the agent has verified that the message is in response to its request at the beginning of the session, and that this message had to come from S (since S is the only other agent knowing the agent's key).

Using affine resolution analysis as discussed above, we discover no attack on this protocol. If we use classical logic, however (in effect, allowing agents to send multiple versions of a message in each session), we uncover a subtle probabilistic flaw. To illustrate this flaw, we consider part of the proof tree corresponding to P_i using this modified analysis. By the message structure rules, the adversary need learn $\{P_i\}_{K_i}$ and K_i. $\{P_i\}_{K_i}$ it can learn from the protocol by producing a message of the form $[\{K_i\}_{K_A}, \langle A, B, N_i, \{K_i\}_{K_A}\rangle_{K_A}]$; this in turn it can learn by supplying $[A, B, N_i, x]$. To learn K_i, it must learn $[\{K_i\}_{K_a}, \langle a, b, y, \{K_i\}_{K_a}\rangle_{K_a}]$ and K_a for some a, b, and y. This it can learn by providing $[a, b, y, z]$, with the constraint that it knows K_a. Since in our modified analysis the messages sent by S in particular session

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

may differ (in particular, the nonces may vary while K_i remains the same), this amounts to a possible attack. On its face, this "attack" seems nonsensical, since S will not in fact send two messages that differ in this sense during one session. But this attack actually reveals a flaw based on the so-called "birthday surprise".

The adversary may send a set of messages to S of the form A, B, N_x, M_x for varying N_x and M_x, take enough time to learn the K_x, and store the results. It may then wait for a "legitimate" message of the form A, B, N_i, M_i to be sent to S. If this message matches one of the adversary's earlier messages, it can intercept this message and substitute a message containing an "old" K_x, and thereby learn any subsequent transmissions encrypted using this key. This allows the adversary to exploit the "birthday surprise" ([Sta95]): even though there are 2^{2k} messages of the form A, B, N_x, M_x (for nonces of length k), the adversary need only store 2^k entries in its table to have a 0.5 probability of matching one of 2^k future A, B, N_i, M_i.

This result raises the possibility that our analysis may be extendible to discover such attacks based on the birthday surprise; this is something that no other analysis technique is capable of capturing.

4.2.5 Yahalom

1. $A \rightarrow B$: A, N_i
2. $B \rightarrow S$: $B, \{A, N_i, M_i\}_{K_B}$
3. $S \rightarrow A$: $\{B, K_i, N_i, M_i\}_{K_A}, \{A, K_i\}_{K_B}$
4. $A \rightarrow B$: $\{A, K_i\}_{K_B}, \{M_i\}_{K_i}$

Figure 4.8: The Yahalom protocol

This protocol was first published in [BAN90]. Our analysis finds no attacks on this protocol.

4.2.6 Wide-Mouthed Frog

This protocol first appeared in [BAN90]; it incorporates a novel combination of techniques to achieve its small size. In this protocol, A generates the session key K_i that it wishes to share with agent B. It then sends K_i encrypted with a timestamp to S. S de-

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

1. $A \rightarrow S$: $A, \{TS_1, B, K_i\}_{K_A}$
2. $S \rightarrow B$: $\{TS_2, A, K_i\}_{K_B}$

Figure 4.9: The Wide-Mouthed Frog protocol

$$Q_i$$

$$m, e([t, a, m], k(B)), Curr(t)$$

Figure 4.10: Analysis of the Wide-Mouthed Frog protocol

crypts the message and, if the timestamp is current, re-encrypts K_i with a new timestamp and sends it to B. When B receives the message, it decrypts and checks the timestamp. If it is current, B begins communicating with A using K_i as the session key. Thus, in this protocol S serves more as a "trusted third party" than as a key server in the literal sense. This protocol is relatively secure — there is no attack on P_i, but there is a small possibility that the adversary could learn Q_i. See Figure 4.10 for our analysis.

The adversary can learn Q_i if it can learn the encryption of Q_i under the session key m and learn the key m. The adversary can learn the encryption of Q_i if it can produce a message of the form $\{t, a, m\}_{K_B}$ in which the timestamp t is fresh; thus the adversary needs to know m and $\{t, a, m\}_{K_B}$ and $Curr(t)$ must be true. $Curr(t)$ is certain to be true only when the adversary knows K_B, which is not possible. Thus, the only possible attack involves the adversary intercepting some message from S to B and learning the K_i included therein before the timestamp has expired. It can forward this message on to B and learn the Q_i that B subsequently sends. This attack has not been demonstrated by other methods.

4.2.7 Modified Wide-Mouthed Frog

1. $B \rightarrow A$: A, B, M_i
2. $A \rightarrow S$: $\{A, B, M_i, K_i\}_{K_A}$
3. $S \rightarrow B$: $\{A, B, M_i, K_i\}_{K_B}$

Figure 4.11: The Modified Wide-Mouthed Frog protocol

This modification of the Wide-Mouthed Frog protocol appeared in [DP97]; it does away

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

with timestamps at the cost of one additional message. This version of the protocol has no attacks.

4.2.8 Kerberos

$$
\begin{array}{lll}
I.1. & A \rightarrow S: & A, T, TS_1 \\
I.2. & S \rightarrow A: & \{J_i, T, TS_2, \{J_i, A, T, TS_2\}_{K_T}\}_{K_A}
\end{array}
$$

$$
\begin{array}{lll}
II.1. & A \rightarrow T: & B, \{J_i, A, T, TS_2\}_{K_T}, \{A, TS_3\}_{J_i} \\
II.2. & T \rightarrow A: & \{K_i, B, TS_4, \{K_i, A, B, TS_4\}_{K_B}\}_{J_i} \\
II.3. & A \rightarrow B: & \{K_i, A, B, TS_4\}_{K_B}, \{A, TS_5\}_{K_i} \\
II.4. & B \rightarrow A: & \{TS_5 - 1\}_{K_i}
\end{array}
$$

Figure 4.12: The Kerberos protocol

The Kerberos service is intended to provide security and authenticity in a particular context.[2] It is designed for use in an open distributed environment in which users at workstations may access services on servers throughout the network. We wish to ensure that only authorized users are granted access to resources, and that servers are able to authenticate requests for service. In general, we require that the user proves its identity for each desired service; we also require that the servers prove their identity to users (thereby preventing adversarial agents from providing some bogus "service"). A particular challenge implicit in this framework arises from the fact that users identities are ultimately proven via the use of passwords. Requiring the user to prove its identity in this manner for *every* desired service is both unwieldy and potentially insecure (since this could result in many copies of the user's password being transmitted across the network). This problem is solved in the Kerberos system by breaking the process of acquiring a service into two related protocols, each using its own central server. Each user's password is tied to its long-lived key: user A's long-lived key K_A is computed from A's password. Thus, when user A must decrypt using K_A in order to proceed with the protocol (specifically, in message $I.2$), the user enters its password, K_A is computed, and (if correct), the decryption is performed.

The first protocol uses an *authentication server* to authenticate the user to the system.

[2] Our explanation of Kerberos is based on that found in [Sta95].

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Successful authentication then allows the user to communicate with a *ticket-granting server* in the second protocol. From this second protocol, the user will acquire a "ticket" which it will then be able to present in order to gain a particular service.

The sole purpose of the first protocol is to authenticate a user to the Kerberos system (which includes all services which the user may request) so that it may engage in further communication with the ticket-granting server. In this first protocol, a user A sends a message to the authentication server S. This messages consists of A's identity, the identity T of the ticket-granting server with which it wishes to communicate, and a timestamp TS_1. If the timestamp is current, S responds with a message encrypted under the key it shares with A. This messages includes a session key J_i that A will use to communicate with T, a timestamp TS_2, and a "ticket" that A will present to T. This ticket will allow T to verify that J_i is a legitimate session key for use in communicating with A. The ticket contains the key J_i, A's identity, T's identity, and the timestamp TS_2 (thus we see that TS_2 is used to limit the use of J_i — typically it imposes a "lifetime" of around 8 hours).

When A receives this message from S, it enters its user password, from which K_A is computed. Thus, if the password is correctly entered, A may then proceed to acquire some service in the next protocol. This protocol begins with a message from A to T consisting of the identity of a server, B, from which A desires a service; the ticket A received from the first protocol; and an "authenticator", which consists of A's identity and a timestamp TS_3, all encrypted under the session key J_i. In essence, the ticket allows the session key J_i to be distributed securely, while the authenticator verifies A's identity. The ticket says "Anyone who uses J_i is required to be A," and the authenticator says, "At time TS_3, A uses J_i." When T receives this message, it decrypts the ticket to extract A's identity and J_i. It then uses J_i to decrypt the authenticator and check A's identity. If these match,[3] then T generates a new ticket for A to present to the server B. This ticket (similar to the ticket from the first protocol) contains a new session key K_i, a timestamp TS_4, and the identities of A and B, all encrypted under the key K_B shared by T and B. T sends this ticket, along with plaintext versions of its contents, to A, encrypted under J_i. A may

[3]Technically, agent identities are composed of user identities and network addresses, so T not only checks that the ticket and authenticator agree but also checks that the network address from which the message originates agrees with the contents of the message.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

then decrypt this message and extract the ticket. It then sends the ticket, along with a corresponding authenticator, to B, requesting service. If all timestamps are current and A's identity is correct, B handshakes with A; A and B may then communicate using K_i.

In our analysis, we may either analyze these protocols independently, or we may combine them into one. This level of analysis allows us to explore the ramifications of the compromise of one part of the protocol while assuming the other part remains secure. Our analysis shows that the protocol (in parts and in total) is secure. Bella and Riccobene ([BR97]), however, demonstrate a security weakness of the protocol. They point out that in a situation where the adversary may steal some session key (that is, either some K_i or some J_i), it may be able to mount a successful attack. In the second protocol, if the adversary can steal some K_i, then it can mount a successful attack if it is able to modify or forge the authenticator $(\{A, TS_5\}_{K_i})$ within the validity window given by TS_4. Similarly, if the adversary can steal some J_i in the first protocol, it can attack if it can modify or forge the authenticator $\{A, TS_3\}_{J_i}$ within the window given by TS_2.

Clearly, these attacks require more of the adversary than our framework is able to capture naturally. But we can modify our analysis by explicitly giving the adversary knowledge of one or both kinds of keys, then searching for an attack to determine what practical effect this additional knowledge has on the adversary's capabilities. Thus, our analysis tool can be used both to prove that a protocol is secure and to determine the security implications of unexpected or hypothetical security breaches.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

CHAPTER 5

Discussion and Future Work

The work described in this thesis represents improvements in the analysis of security protocols along several axes. At the same time, our analysis has only been developed in the context of a limited class of protocols, and so we expect many interesting results from the project of extending our methodology to a wider array of protocols. Below, we review some of the new developments accompanying our work, and then we discuss areas for future exploration.

5.1 Attack Defined

A critical element of our work is that we have posited an elegant and broad definition of a protocol's vulnerability to attack. By placing the adversary in the center of our model, we are able to give a simple characterization of how protocols must be designed in order to foil an adversary's attack. Moreover, this definition is generalizable in several directions — within the analysis of a particular protocol we may test the effects of various extensions of the adversary's knowledge; and the idea of session-independence, as well as our characterization of secrecy and authenticity, should be extendible to other sorts of protocols.

5.2 Formal Semantics

By focusing our analysis exclusively on the adversary's knowledge, and by characterizing protocol flaws simply in terms of this knowledge, we have been able to expose some of the logical structure of protocol analysis. This in turn has allowed us to give a formal semantics for our analysis with respect to which our system is complete. That is, we are able to say

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

precisely what it means when our analysis fails to find an attack on a protocol. Our method is therefore valuable not only as a way to discover protocol flaws and *insecurity* but also as a way to make strong statements about protocol *security*.

5.3 Automation

Our understanding of the logical underpinnings of our analysis has provided for an easy translation of our technique into logic programming terms, which is a natural paradigm for automated analysis. In addition, the simplicity of our model contributes greatly to the efficiency of our automated analysis, allowing for fully automated analyses that take place in a matter of seconds. We do not need a complex reformulation of a protocol, nor do we need to prove an array of intermediate results en route to our goal. The infinite size of the search space does not affect our analysis, since our analysis focuses on only the kinds of messages that might be produced during a protocol execution.

5.4 Intuition

Finally, we believe that our technique is particularly intuitive, both to use and to understand. Where a number of other techniques require careful *a priori* analysis of a protocol in order to express it using some special-purpose syntax, our analysis works from the standard specifications given in the literature. Moreover, utilization of our method requires no special knowledge of, for example, complexity theory or belief logic; both inputs and outputs are expressed in terms of simple sequences of messages.

5.5 Future Work

While secure key distribution protocols are arguably the foundation of a secure distributed system, they are not sufficient to provide guarantees of security. We are beginning to explore the possibility of applying our technique to other kinds of secure distributed protocols, such as voting protocols or resource-allocation protocols. In addition, we are investigating ways in which our method can be extended to different environments, such as those in which asymmetric key cryptography is used, or in which agents may revoke the key(s) they

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

share with the key-server. We wish to be able to model explicitly other kinds of operations such as XOR. Since this operation has some identities, such as $((a \text{ XOR } b) \text{ XOR } a) = a$, that are not easily modeled using inference rules, we wish to extend our model to include congruences or "rewrite rules" that will allow us to capture these more complex operations. We also plan to explore the utility of our analysis in discovering attacks based on the birthday surprise.

We also have more work to do with the formal semantics we have provided. In particular, we currently operate under the assumption that the adversary can only perform a few operations — encryption, decryption, etc. While this is a common assumption across the literature, it does not accurately reflect our understanding of the adversary's capabilities. We hope to be able to ground this sort of assumption in a more careful analysis of the capabilities of an arbitrary probabilistic polynomial-time machine.

Related to this is the question of proving the security of protocol implementations — once we have proved a protocol is secure at the level of abstraction used in this thesis, we want to be able to prove that a particular implementation of the protocol meets its specification.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

BIBLIOGRAPHY

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

BIBLIOGRAPHY

[AFS97] J. Alves-Foss and T. Soule. A weakest precondition calculus for analysis of cryptographic protocols. In *Proceedings of the DIMACS Workshop on Formal Verification of Security Protocols*, September 1997.

[AG98] M. Abadi and A. Gordon. A calculus for cryptographic protocols the spi calculus. Technical Report SRC Research Report 149, Digital Systems Research Center, 1998.

[And92] J.-M. Andreoli. Logic programming with focusing proofs in linear logic. *Journal of Logic and Computation*, 2(3), 1992.

[AT91] M. Abadi and M. R. Tuttle. A semantics for a logic of authentication. In *Proceedings of the Tenth Annual ACM Symposium on Principles of Distributed Computing*, pages 201–216. ACM, August 1991.

[BAN90] M. Burrows, M. Abadi, and R. Needham. A logic of authentication. *ACM Transactions on Computer Systems*, 8, February 1990.

[BBF83] R. K. Bauer, T. A. Berson, and R. J. Feiertag. A key distribution protocol using event markers. *ACM Transactions on Computer Systems*, 1(3):249–255, August 1983.

[Bie90] Pierre Bieber. A logic of communication in a hostile environment. In *Proceedings of the Computer Security Foundations Workshop III*, pages 14–22. IEEE, June 1990.

[BM90] John Burns and Chris J. Mitchell. A security scheme for resource sharing over a network. *Computers & Security*, 9:67–75, 1990.

[BO97] John A. Bull and David J. Otway. The authentication protocol. Technical Report DRA/CIS3/PROJ/CORBA/SC/1/CSM/436-04/03, Defence Research Agency, Malvern, UK, 1997.

[Bol96] Dominique Bolignano. An approach for the formal verification of cryptographic protocols. In *Proceedings of the Third ACM Conference on Computer and Communications Security*, pages 106–118. ACM Press, 1996.

[BP97] Giampaolo Bella and Lawrence C. Paulson. Using Isabelle to prove properties of the Kerberos authentication system. In *Proceedings of the DIMACS Workshop on Formal Verification of Cryptographic Protocols*, September 1997.

[BR93a] Mihir Bellare and Phillip Rogaway. Entity authentication and key distribution. In *Advances in Cryptology — Crypto '93 Proceedings*, 1993.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

[BR93b] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *Proceedings of the First ACM Conference on Computer and Communications Security*, 1993.

[BR95] Mihir Bellare and Phillip Rogaway. Provably secure session key distribution — the three party case. In *Proceedings of the 27th ACM Symposium on the Theory of Computing*, 1995.

[BR97] Giampaolo Bella and Elvinia Riccobene. Formal analysis of the Kerberos authentication system. *Journal of Universal Computer Science*, December 1997.

[BS97] Jeremy Bryans and Steve Schneider. CSP, PVS, and a recursive authentication protocol. In *Proceedings of the DIMACS Workshop on Formal Verification of Cryptographic Protocols*, September 1997.

[Car94] Ulf Carlsen. Cryptographic protocol flaws: Know your enemy. In *Proceedings of the Computer Security Foundations Workshop VII*. IEEE Computer Society, 1994.

[Cer92] Serenella Cerrito. Herbrand methods in sequent calculi: Unification in LL. In Krzysztof Apt, editor, *Logic Programming: Proceedings of the Joint International Conference and Symposium on Logic Programming*, pages 607–621. The MIT Press, 1992.

[Com93] Kevin Compton. A deductive system for existential least fixpoint logic. *Journal of Logic and Computation*, 3(2):197–213, 1993.

[DGT95] Donald T. Davis, Daniel E. Green, and Theodore Ts'o. Kerberos with clocks adrift: History, protocols, and implementation. In *Proceedings of the 5th USENIX UNIX Security Symposium*, pages 35–40, June 1995.

[DP97] Bart De Decker and Frank Piessens. Cryptolog: A theorem prover for cryptographic protocols. In *Proceedings of the DIMACS Workshop on Formal Verification of Security Protocols*, September 1997.

[DS81] D. E. Denning and G. M. Sacco. Timestamps in key distribution protocols. *Communications of the ACM*, 24(8):533–536, 1981.

[DY83] Danny Dolev and Andrew C. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, IT-29(2):198–208, March 1983.

[Gir87] J.-Y. Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.

[GMR89] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on Computing*, pages 186–208, 1989.

[GNY90] Li Gong, Roger Needham, and Raphael Yahalom. Reasoning about belief in cryptographic protocols. In *Proceedings of the IEEE Computer Society Symposium on Security and Privacy*, pages 234–248. IEEE, May 1990.

[Gol] Oded Goldreich. Foundations of cryptography. Unpublished book fragment.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

[Gon93] Li Gong. Variations on the themes of message freshness and replay. In *Proceedings of the Computer Security Foundations Workshop VI*, volume 6, pages 131–126. IEEE, 1993.

[Gur94] Y. Gurevich. Evolving algebras 1993: Lipari guide. In E. Börger, editor, *Specification and Validation Methods*. Oxford University Press, 1994.

[HM94] Joshua S. Hodas and Dale Miller. Logic programming in a fragment of intuitionistic linear logic. *Journal of Information and Computation*, 110(2):327–365, May 1994.

[Hoa85] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, 1985.

[Hog84] Christopher John Hogger. *Introduction to Logic Programming*. Number 21 in APIC Studies in Data Processing. Harcourt Brace Jovanovich, 1984.

[HP94] J. Harland and D. Pym. A uniform proof-theoretic investigation of linear logic programming. *Journal of Logic and Computation*, 4(2), April 1994. 175–207.

[HR93] Peter Honeyman and Aviel David Rubin. Formal methods for the analysis of authentication protocols. Technical Report CITI 93–7, Center for Information Technology Integration, Information Technology Division, The University of Michigan, 1993.

[Kai96] Rajashekar Kailar. Accountability in electronic commerce protocols. *IEEE transactions on software engineering*, 22(5):313–328, May 1996.

[KMM94] R. Kemmerer, C. Meadows, and J. Millen. Three systems for cryptographic protocol analysis. *Journal of Cryptology*, 7:79–130, 1994.

[KW94] Volker Kessler and Gabriele Wedel. AUTLOG - an advanced logic of authentication. In *Proceedings of the Computer Security Foundations Workshop VII*, pages 90–99. IEEE Computer Society, June 1994.

[KW96] David Kindred and Jeannette M. Wing. Fast, automatic checking of security protocols. In *Proceedings of the 2nd Usenix Workshop on Electronic Commerce*, pages 41–52, 1996.

[Lin92] P. Lincoln. Linear logic. *ACM SIGACT Notices*, 23(2):29–37, Spring 1992.

[LM93] T. Leighton and S. Micali. Secret-key agreement without public-key cryptography. In *Proceedings of Crypto '93*, pages 456–479, 1993.

[Low96] Gavin Lowe. Breaking and fixing the needham-schroeder public key protocol using CSP and FDR. In T. Margaria and B. Steffen, editors, *Tools and Algorithms for the Construction and Analysis of Systems Second International Workshop, TACAS '96*, volume 1055 of *Lecture Notes in Computer Science*, pages 162–169. Springer-Verlag, 1996.

[LS87] Jacques Loeckx and Kurt Sieber. *The Foundations of Program Verification*. Wiley and Sons, 1987.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

[MB93] Wenbo Mao and Colin Boyd. Towards formal analysis of security protocols. In *Proceedings of the Computer Security Foundations Workshop VI*, volume 6, pages 147–158. IEEE, 1993.

[MB94] Wenbo Mao and Colin Boyd. Development of authentication protocols: Some misconceptions and a new approach. In *Proceedings of the Computer Security Foundations Workshop VII*, pages 178–186. IEEE, 1994.

[MCJ97] Will Marrero, Edmund Clarke, and Somesh Jha. Model checking for security protocols. Technical Report CMU-CS-97-139, School of Computer Science, Carnegie Mellon University, May 1997.

[Mea91] Catherine Meadows. A system for the specification and analysis of key management protocols. In *Proceedings of the 1991 IEEE Computer Society Symposium on Research in Security and Privacy*, pages 182–195, 1991.

[Mea92] Catherine Meadows. Applying formal methods to the analysis of a key management protocol. *Journal of Computer Security*, 1(1):5–35, 1992.

[Mea96] Catherine Meadows. The NRL protocol analyzer: an overview. *The Journal of Logic Programming*, pages 113–131, 1996.

[NS78] R. M. Needham and M. D. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12):993–999, 1978.

[NS93] B. Clifford Neuman and Stuart G. Stubblebine. A note on the use of timestamps as nonces. *ACM Operating Systems Review*, 27(2):10–14, April 1993.

[OR87] D. Otway and O. Rees. Efficient and timely mutual authentication. *Operating Systems Review*, 21(1):8–10, 1987.

[Pau96a] Lawrence C. Paulson. *ML for the working programmer.* Cambridge University Press, 1996.

[Pau96b] Lawrence C. Paulson. Proving properties of security protocols by induction. Technical Report TR-409, Computer Laboratory, University of Cambridge, 1996.

[Pau97a] Lawrence C. Paulson. Mechanized proofs for a recursive authentication protocol. In *Proceedings of the 10th IEEE Computer Security Foundations Workshop*, pages 84–95, June 1997.

[Pau97b] Lawrence C. Paulson. Mechanized proofs of security protcols: Needham-Schroeder with public keys. Unpublished manuscript, January 1997.

[Rub94] Aviel David Rubin. *Nonmonotonic Cryptographic Protocols.* PhD thesis, The University of Michigan, 1994.

[RZ97] Peter Ryan and Irfan Zakiuddin. Modelling and analysis of security protocols. In *Proceedings of the DIMACS Workshop on Formal Verification of Security Protocols*, September 1997.

[Sne91] Einar Snekkenes. Exploring the BAN approach to protocol analysis. In *Proceedings of the IEEE Computer Society Symposium on Security and Privacy*, pages 171–181, May 1991.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

[Sne92] Einar Snekkenes. Roles in cryptographic protocols. In *Proceedings of the 1992 IEEE Computer Society Symposium on Research in Security and Privacy.* IEEE Computer society press, 1992.

[SR96] Victor Shoup and Aviel David Rubin. Session key distribution using smartcards. In *Proceedings of Eurocrypt '96*, May 1996.

[Sta95] William Stallings. *Network and internetwork security: principles and practice.* Prentice-Hall, 1995.

[SvO94] Paul Syverson and Paul van Oorschot. On unifying some cryptographic protocol logics. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 14–28, 1994.

[Syv91] Paul Syverson. The use of logic in the analysis of cryptographic protocols. In *Proceedings of the 1991 IEEE Computer Society Symposium on Research in Security and Privacy*, pages 156–170. IEEE, May 1991.

[Syv94] Paul Syverson. A taxonomy of replay attacks. In *Proceedings of the Computer Security Foundations Workshop VII.* IEEE Computer Society, 1994.

[Tam94] Tanel Tammet. Proof strategies in linear logic. *Journal of Automated Reasoning*, 12:273–304, 1994.

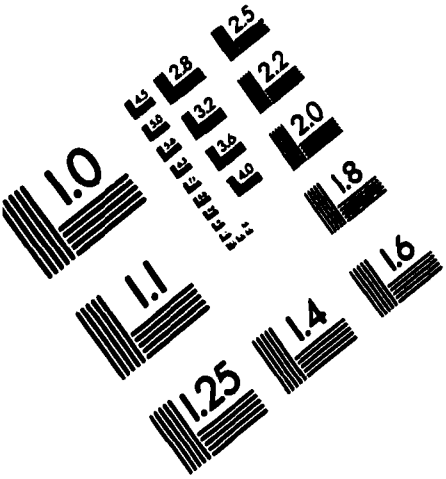
[vO93] Paul C. van Oorschot. Extending cryptographic logics of belief to key agreement protocols (extended abstract). In *Proceedings of the First ACM Conference on Computer and Communications Security*, pages 232–243, November 1993.

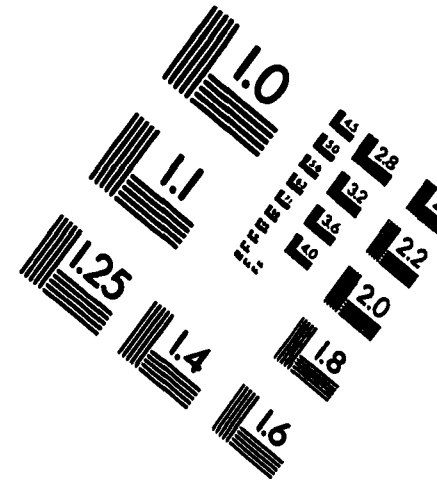
[WH96] Michael Winikoff and James Harland. Some applications of the linear logic programming language lygon. In *Proceedings of the Australasian Computer Science Conference*, pages 262–271, February 1996. Available at URL http://www.cs.rmit.edu.au/lygon/papers/gz/acsc96.ps.gz.

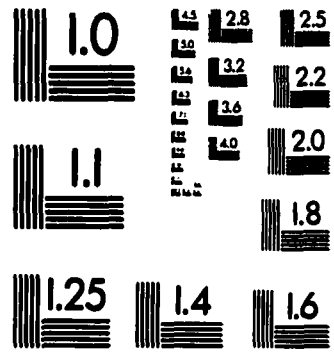
[WL93] Thomas Y. C. Woo and Simon S. Lam. A semantic model for authentication protocols. In *Proceedings of the Symposium on Research in Security and Privacy*, pages 178–194. IEEE, May 1993.

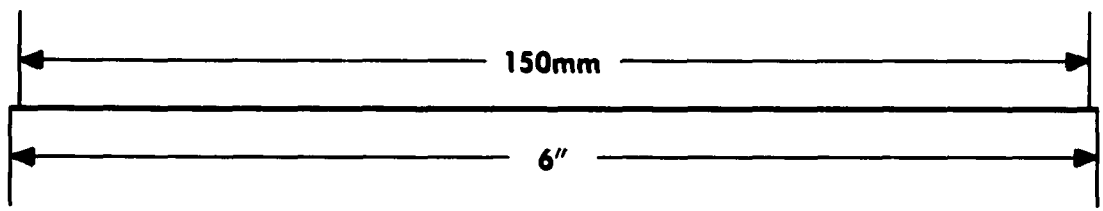
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

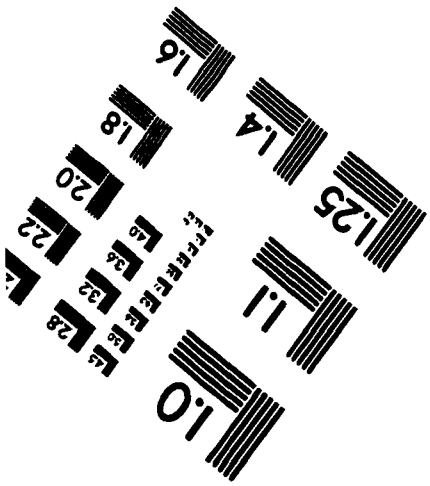
IMAGE EVALUATION TEST TARGET (QA-3)

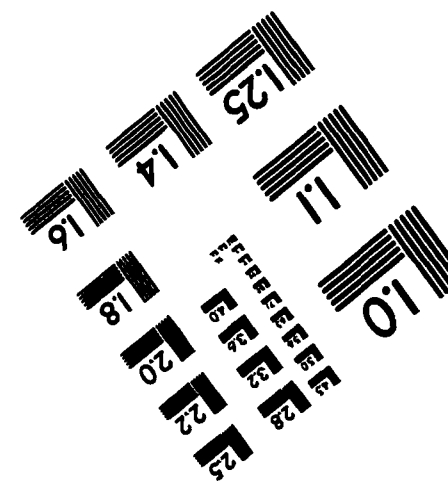












APPLIED IMAGE, Inc
1653 East Main Street
Rochester, NY 14609 USA
Phone: 716/482-0300
Fax: 716/288-5989

© 1993, Applied Image, Inc., All Rights Reserved

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.