NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

SECURE DISTRIBUTION OF OPEN SOURCE INFORMATION

by

Jason Rogers

December 2004

Thesis Advisor:	George Dinolt
Second Reader:	Timothy Levin

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.			
1. AGENCY USE ONLY ***(Leave blank)***	**2. REPORT DATE** December 2004	**3. REPORT TYPE AND DATES COVERED** Master's Thesis	
4. TITLE AND SUBTITLE: Secure Distribution of Open Source Information			**5. FUNDING NUMBERS**
6. AUTHOR(S) Rogers, Jason			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000			**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A			**10. SPONSORING / MONITORING AGENCY REPORT NUMBER**
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.			
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited			**12b. DISTRIBUTION CODE**

13. ABSTRACT ***(maximum 200 words)***

Cryptographic protocols provide security services through the application of cryptography. When designing a cryptographic protocol, the requirements are, often, specified informally. Informal specification can lead to incorrect protocols from misinterpreting the security requirements and environmental assumptions. Formal tools have been shown to reduce ambiguity.

In this paper, a cryptographic protocol, called the Secure Open Distribution Protocol (SODP), is developed to provide authentication services for open source information. A formal development process is proposed to aid in the design of the SODP. The Strand Space method has been selected as the formal mechanism for specifying requirements, architecting a protocol design, and assuring the correctness of the protocol. First, the informal authentication requirements are modeled as agreement properties. Next, Authentication Tests, a Strand Space concept, are introduced to aid in the design of the SODP. Finally, a formal proof is constructed to assure that the SODP has satisfied all requirements.

The result of the development process proposed in this paper is a cryptographic protocol that can be used to securely distribute open source information. Also, the Strand Space method is demonstrated as a viable option for the formal development of a cryptographic protocol.

14. SUBJECT TERMS Formal Methods, Protocol Analysis, Cryptography			**15. NUMBER OF PAGES** 76
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UL

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

SECURE DISTRIBUTION OF OPEN SOURCE INFORMATION

Jason Lee Rogers
Civilian, Federal Cyber Corps
B.S, North Central College, 2002

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
December 2004

Author: Jason Lee Rogers

Approved by: George Dinolt
Thesis Advisor

Timothy Levin
Second Reader

Peter Denning
Chairman, Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Cryptographic protocols provide security services through the application of cryptography. When designing a cryptographic protocol, the requirements are, often, specified informally. Informal specification can lead to incorrect protocols from misinterpreting the security requirements and environmental assumptions. Formal tools have been shown to reduce ambiguity.

In this paper, a cryptographic protocol, called the Secure Open Distribution Protocol (SODP), is developed to provide authentication services for open source information. A formal development process is proposed to aid in the design of the SODP. The Strand Space method has been selected as the formal mechanism for specifying requirements, architecting a protocol design, and assuring the correctness of the protocol. First, the informal authentication requirements are modeled as agreement properties. Next, Authentication Tests, a Strand Space concept, are introduced to aid in the design of the SODP. Finally, a formal proof is constructed to assure that the SODP has satisfied all requirements.

The result of the development process proposed in this paper is a cryptographic protocol that can be used to securely distribute open source information. Also, the Strand Space method is demonstrated as a viable option for the formal development of a cryptographic protocol.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

This material is based upon work supported by the National Science Foundation under Grant No.DUE-0114018.

Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of the National Science Foundation.

I would like to thank my advisor George Dinolt for all his advice and encouragement. I know I can depend on further support for many years to come.

I would also like to thank the National Science Foundation and the Naval Postgraduate School for giving me the opportunity to further my education and serve my country.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

Information is the new global currency. Each day information is bought and sold on a global marketplace called the Internet. Web sites are the brokers of the information market, processing millions of transactions each day. Users, operating behind electronic tellers, barter with web sites to gain valuable pieces of information. Users make deposits by viewing advertisements, filling out forms, or handing over credit card information. For every transaction the value of information is scrutinized.

The value of information is measured by its integrity and availability. The timely nature of information requires that it be available and correct. If air traffic information is incorrect or unavailable an aviator will have a difficult time avoiding mid-air collisions. Therefore, the FAA invests millions each year to insure that air traffic control system reports correct information in a timely manner.[Ref. 11] The majority of information seekers, however, cannot afford to spend half that amount to insure their availability demands.

The open source movement has eased availability demands by saturating the Internet with free sources of information. Under the umbrella of a barrage of open source licenses, like the GNU General Public License (GPL)[Ref. 12] and the Creative Commons licenses[Ref. 7], books, software and even media is shared freely on the open market. As open source information is shared, the number of available copies grows without bounds. As the number of copies grow, the availability improves.

The improved availability, though, is not without consequence. Improved ability to share information also allows for the easy dissemination of all kinds of false information. Patrons of open source information desire procedures for validating the integrity of that information without disrupting availability.

A. TRUSTED COMPUTING EXEMPLAR PROJECT

The Trusted Computing Exemplar project [Ref. 8] is one patron interested in the secure distribution of open source information. In 2002, it was recognized by a group of researchers that the private and government sectors had not invested a significant amount of time in the development of high assurance systems. Even though consumers were slowly recognizing the need for security and reliability in their computing systems, few in the industry understood how to develop such systems.

The Center of Information Security Research at the Naval Postgraduate School decided to provide an example. The goal of the Trusted Computing Exemplar (TCX)[Ref. 8] project is to educate others on the development of trusted computing systems through a worked example. One of the goals of the project will be the "open distribution of project deliverables." The successful sharing of the project deliverables will enable others to study the results of the project, so that future high assurance initiatives are more likely. In order to securely provide this deliverable, a secure distribution protocol for open source information is required.

The TCX has chosen the Common Criteria for Information Technology Security Evaluation[Ref. 20] as the high assurance criteria to exemplify. The goal of the Common Criteria is to establish a basis for evaluating information security products. The Common Criteria provides assurance through documentation and evaluation. The Common Criteria identifies 7 increasing levels of assurance, describing the evidence that is required for a product to meet that level.

The documentation is produced in two distinct phases. In the first phase, the developer defines product requirements and declares all procedures that will be followed during the life-cycle of the product to satisfy each assurance requirement. The second phase of documentation, provides the design documentation as well as the evidence that the procedures have been followed. The documentation and product are then given to an evaluator who determines if the evidence is sufficient for the product to reach the desired level of assurance. For a product to be construed as high

assurance under the common criteria, it must satisfy the highest level of assurance. The TCX project will develop a product that meets the requirements for EAL 7, the highest level of assurance under the Common Criteria.

B. PROTOCOLS

Protocols dictate rules for achieving goals. For example, when a pilot wants to land an aircraft, he must follow a protocol, a set of steps that he must take in a specified order. The air traffic controller, who responds to the pilot request, is following the same protocol under rules defined by his role.

The world is filled with protocols establishing the rules for withdrawing money, making laws, and paying taxes, etc. The rules explain the steps participants must complete in order for a desired goal to be met. When a participant chooses not to follow the rules, the protocol should also enforce the proper consequence. If it does not and the unruly participant is able to achieve some unauthorized behavior of the system, the protocol should be fixed. The proper design and analysis of protocols keeps money safe, the government running, and aircraft flying. In object oriented computer languages, the methods defined for an object are the "protocol" that object supports or accepts.

The growth of interconnected machines in the form of large networks of computers - the INTERNET - has brought the importance of good protocol design to the forefront. When engineers wanted the first two computers to communicate a protocol - an agreement about the order and meaning of the messages to be exchanged - was written. The communication was most likely symmetric, with the output tape of one computer fed directly into the input tape of the other computer. When three computers attempted to communicate, a more sophisticated protocol was required to stop two computers from speaking at the same time. As the size and distance of the networks grew, protocols were required to reduce latency, improve reliability, and route messages accurately.

Today, millions of machines are able to communicate through a complex mesh of protocols. The working groups of the Internet Engineering Task Force (IETF) have been charged with making sense of the Internet by standardizing the communication protocols that define it.[Ref. 4] Manned by volunteers of network designers, operators, vendors and researchers the working groups seek to provide an open environment for establishing protocol standards in the form of a "Request for Comments" (RFC) documents. For example, RFC's 793[Ref. 36] and 791[Ref. 35] provide a technical specification of the Transport Control Protocol (TCP) and the Internet Protocol (IP), the building blocks of the TCP/IP standard that carries the majority of Internet traffic. For users requiring more security than is offered by the Internet Protocol (IP), RFC 2401[Ref. 23] was created, which describes the IP Security Protocol (IPSEC). IPSEC belongs to a class of protocols, called cryptographic protocols.

C. CRYPTOGRAPHIC PROTOCOLS

Cryptographic protocols are communication protocols that use cryptography. Cryptography is the study of disguising, or *encrypting*, messages by changing the form of the message. The original message is called the *plain text*, while the encrypted version is called the *cipher text*. A strong cryptographic algorithm, or *cryptosystem*, is one in which only an authorized party can transform, or *decrypt*, the cipher text back into plain text. Modern cryptosystems use keys as an additional input into the encryption algorithm when encrypting messages. Therefore, a single algorithm can be used by multiple parties, each possessing a different key. Cryptosystems that use the same key to encrypt and decrypt are called *symmetric*, while those that use different keys are called *asymmetric*.

Information can be encrypted using a symmetric key to keep the contents secret from those who don't possess the key. Or a challenge message can be signed with a private asymmetric key to authenticate the identity of the one who signed the key. Security properties, like secrecy and authentication, are valuable when communicating

with parties across a global infrastructure, like the INTERNET. Including encryption in communication protocols one can distribute these valuable security services to users throughout the world.

The goals of a cryptographic protocol are to provide security services using the message passing mechanism provided by communication protocols. For example, the security goal for a symmetric key distribution protocol is to transmit keys to participants on a distributed network, while keeping the contents secret. Another goals for a key distribution will include assuring the integrity of the key, so the correct key is received. Participants will also want to identify the source of the key, so that the participant isn't tricked into receiving false keys. The success of the key distribution protocol relies on the ability to provide each of the subgoals.

IPSEC is a family of cryptographic protocols that, together, provide a collection of security services. For example, the Internet Key Exchange (IKE) protocol, a member of the IPSEC family, is used to distribute keys used for digital signature, public key encryption, and symmetric key encryption. The IP Encapsulating Security Payload (ESP) protocol uses the keys distributed by IKE to encrypt data in order for it to remain secret as it is transmitted between computers on a communication network. If a flaw existed in the IKE protocol that allowed an authorized user to read a symmetric key, then the ESP protocol would also be affected. Of course, even a well written RFC can lead to insecure protocols.

As of September 2004, 3890 RFC's have been published by the IETF, many of them security related. Aside from the informal specification provided by the RFC, proper implementation of protocols, including cryptographic protocols, is the responsibility of vendors, and other implementors. Understanding the intricate relationships among protocols, their participants, and the achieved outcome is a complicated task.

Formal tools have been shown to reduce complexity and ambiguity of informal specifications, like RFC's, and to focus the efforts of protocol analysts and designers. In 1999[Ref. 30], Meadows used the NRL Protocol Analyzer to formally analyze the

IKE protocol. She uncovered several ambiguous statements in the specification given in the RFC's that could have led to possible attacks.

D. OVERVIEW OF THESIS

This paper will propose a process for developing cryptographic protocols. This process will be used to produce a cryptographic protocol to handle the secure delivery of open source information. Chapters II will describe the internals of communication protocols. Chapter III will describe communication protocols will introduce the Strand Space method; the formal framework that has been selected to model requirements, architect the design, and prove correctness of the secure open distribution protocol. The development of the SODP will be described in chapters IV through VI. Chapter IV will outline the security requirements of the SODP using the Strand Space method. Chapter V will suggest a possible design to satisfy the requirements in Chapter IV. Chapter VI will apply formal analysis tools to prove the correctness of the SODP. Chapter VII will conclude by summarizing the process and recommending future research directives.

Using this information a working implementation of the SODP can be derived to provide the open source community with a secure distribution mechanism. More generally, this paper provides a worked example of how cryptographic protocols can be developed.

II. COMMUNICATION PROTOCOLS

A communication protocol specifies rules for transferring information. The rules govern the actions of participants engaging in the communication protocol. Participants can be humans, or machines acting on behalf of their human users. The telephone, for example, is one machine that acts as a participant in a communication protocol on behalf of its user. Once the numbers have been dialed, the phone must transmit messages across the communication network, built using wires and switches, to ring the phone on the other end. People using the telephone use a protocol, "Hello", "Goodbye", etc. that is layered on top of the telephone communications. In this example, the user, the telephone, and the switches are each participants engaged in a different protocol.

Layering protocols, as in the case of the telephone example, allows communication networks to remain flexible to meet new technologies and load demands. Layering aggregates protocols with similar functions to the same layer. Interfaces are used between layers to hide the internal functionality of the protocol. As long as the protocol meets the requirements of the interface, it can be used to provide functionality for that layer. In the telephone example, the microphone, headphone and keypad provided an interface between the user and the telephone. Even if the communication network was to change, as was the case with cellular phones, users would still able to communicate.

The Transmission Control Protocol (TCP) [Ref. 36] is a communication protocol that can be used to exchange information between processes executing on a host computer. The TCP is considered a Transport protocol, found on layer-4 of the International Standard Organization's Open System Interconnect (ISO/OSI) model.[Ref. 17] The TCP will be used to describe the structure and semantics of communication protocols.

As a layer-4 protocol, the TCP must rely on the lower layers to transfer the

TCP message to the appropriate computer. The interface for the TCP protocol defines four operations. The first two are for opening and closing connections. These operations are used to establish a communication channel between two processes, much like one does when making a telephone call. The second pair of operations are for sending and receiving data on the established channel.

Protocol rules are specified in terms of format and order. Once a participant has assumed a role, the protocol defines the sequence of transmission and reception events and the order in which they should occur. Each event defines the message format that the participant should send or expect to receive. The message format of the TCP includes a header and a data section, as described in Figure 1.

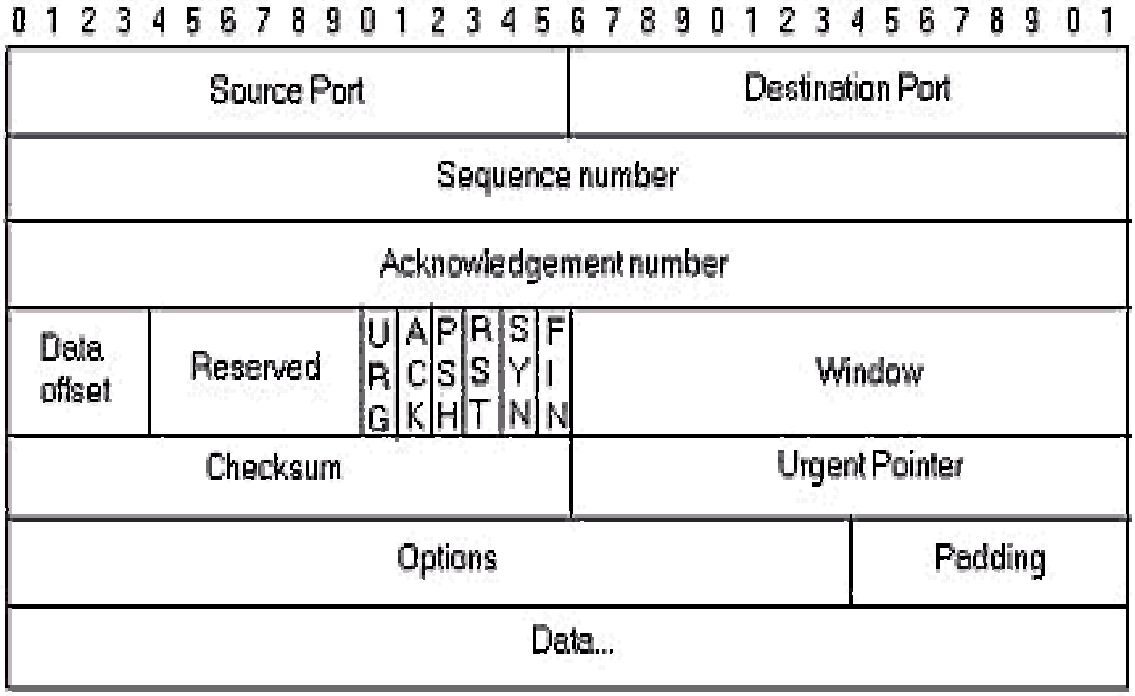


Figure 1. TCP Header Format

The header information is used by participants, in conjunction with the protocols rules, to help construct the other messages specified by the protocol. The TCP header includes a source port field and a destination port field. When a process wants to establish a connection, it is bound to a specific port. The source port identifies the process that send the TCP message, while the destination port tells which the process the message should be delivered to. If a message arrives out of order or in the wrong format, the protocol will define rules for handling either case. In the case of TCP, the header includes a field for uniquely identifying messages, the sequence number, as well as one for telling a participant the number of the last message received, the acknowledgment number. This information can be used for handling out of order packets.

Participants in communication protocols can take on many roles in the course of a protocol. The two most common roles for communication protocols are that of initiator and responder. To begin a protocol, a participant must initiate the conversation. This participant is bound to the rules of the protocol that dictate the behavior of the initiator. Most often the first step for the initiator is to send a message that notifies other participants that a protocol has begun. The TCP header includes control flags for instructing participants on how the message should be handle. The SYN flag is used to identify the message used to initiate a TCP session. The message passing to establish a TCP connection is illustrated in Figure 2. The values for the sequence numbers, acknowledgment numbers, and control flags are listed in the message contents.

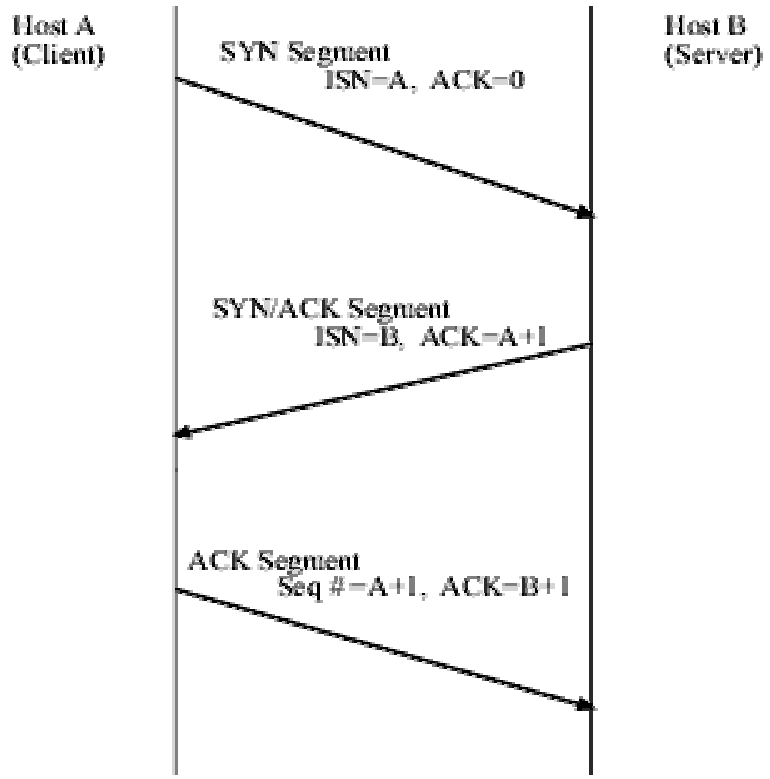


Figure 2. TCP 3-Way Handshake for Establishing a Connection

Once the initiator has sent the SYN message, the participant who receives it can choose whether or not to engage in the protocol by responding. If no participant were to respond in a timely manner, the protocol will either terminate, try again, or present the initiator with some set of options. The proper response is usually a message whose format has been predetermined by the rules of the protocol. After a process has received a SYN message in the TCP, a SYN-ACK message can be sent to acknowledge the reception of the SYN packet. The initiator will then send an ACK message, which acknowledges the receipt of the SYN, ACK packet. At this point the three-way handshake is completed, and a TCP session is open. Particpants can now use the TCP session to exchange messages between protocols.

THIS PAGE INTENTIONALLY LEFT BLANK

III. FORMAL METHODS

Formal methods have been recognized as a valuable tool for reducing the complexity of cryptographic protocols. The WIFT'98 Report[Ref. 15] describes a formal method as an approach to developing computer systems that includes,

1. [A] notation with a well-defined syntax and semantics
2. some guidelines and procedures for using the notation
3. techniques for analyzing specifications expressed in the notation

Over the past couple decades, formal methods have been applied in ways in the design and analysis of cryptographic protocols. Catherine Meadows finds that the majority of formal methods used for cryptographic protocol development are based on two fundamental aspects of Computer Science: logic and finite state automata.[Ref. 32]

A. LOGICS

Logics of knowledge and belief model the behavior of participants engaged in a cryptographic protocol. Syverson[Ref. 40], through Carnap[Ref. 5], states:

> a logical system is characterized by stating its formation rules and its transformation rules. The formation rules provide us with a list of recognized characters and decidable means for delineating the grammatically well formed sentences (or formulae). The transformation rules provide us with a list of axiomatic sentences and (not necessarily decidable) means for delineating those sentences that follow from a given set of sentences - i.e the inference rules.

When modeling cryptographic protocols the two logics, knowledge and belief, have very different objectives. Logics of knowledge are concerned with what knowledge a subversive party looking to infiltrate the protocol might gain during a protocol run. Logics of belief try to capture how beliefs of participants change, and whether or not they have trust in the system at the end of a protocol run.

The most popular of the belief logics is BAN logic, so named for its creators Burrows, Abadi, and Needham [Ref. 33]. BAN logic is a modal logic that models how an honest participant's belief changes during the run of a protocol. Given an initial set of assumptions, BAN logic provides a set of inference rules for extending the belief of a participant. As messages are sent and received, inference rules are applied to determine what new beliefs can be derived from the initial set. Security goals are described by a set of beliefs that must be satisfied during and at the conclusion of a protocol run. If the beliefs can be deduced using the inference rules, then the protocol achieves the described security goal.

B. COMMUNICATING STATE MACHINES

Communicating state machines can be used to model the communication events of participants in a communication protocol. Many concepts found in Computer Science can be modeled using finite state machines. Programming languages, intrusion detection systems, and databases can all be expressed as a sequence of state machines. It is only natural, that communication protocols could be expressed in a similar manner. Though several different definitions of state machines exist, the following definition will be used for describing communication state machines.

Sipser defines a *finite automaton*[Ref. 39] as a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where

1. Q is a finite set called the states,
2. Σ is a finite set called the *alphabet*,
3. $\delta : Q \times \Sigma \rightarrow Q$ is the *transition function*,
4. $q_0 \in Q$ is the *start state*, and
5. $F \subseteq Q$ is the set of *accept states*.

A *sentence* is made up of elements from the alphabet. As it is input into the finite automaton, the transition function transforms the automaton to another element in the set of states. Since the sentence is finite, the automaton will eventually finish

transitioning and arrive at a final state. If the final state is in the set of accept states, the sentence is accepted. If it is not, the automaton rejects the sentence. A *finite state transducer* is a type of finite automaton whose output is a string and not just *accept* or *reject*.

To create a communicating state machine, two or more finite transducers are used where the input alphabet of a transducer is the output string from another transducer. Figure 3 illustrates two communicating state machines exchanging a simple message. The state transitions of each machine are determined by output received from the other machine.

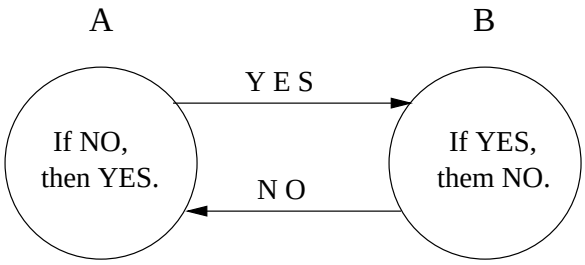


Figure 3. Communicating State Machines

To model a cryptographic protocol, each participant is represented by a communicating state machine. The transitions functions are defined by the protocol, while the messages created by the transitions functions become the input alphabets for other communicating state machines.

The advantage of this approach, cited by Meadows[Ref. 29], is that communicating state machines derive security results directly from the messages instead of a belief developed as a result of receiving the message. Removing the additional inference step reduces the complexity of specifying properties of a protocol, making it easier to understand.

Most recent work based on communicating state machines have modeled the world through a view introduced by Dolev and Yao. [Ref. 9]. The Dolev-Yao model

assumes that the communication network is under the control of an intruder who can read all traffic, alter and destroy messages, create messages, and perform any operation, such as encryption, that is available to legitimate users of the system. So, every message in the protocol passes through the attacker's communicating state machine with the attacker's own set of transition rules.

Protocols that are developed and shown to be correct under this interpretation are assumed to be more resilient to attacks from dishonest users that those developed under other formal methods. Millen's Interrogator [Ref. 22], the NRL Protocol Analyzer [Ref. 38], and the Longley-Rigby tool [Ref. 26] are all, in some respect, based on the Dolev-Yao model. In 1998, another state machine model[Ref. 42] was introduced that captured the Dolev-Yao model using a graphical structure.

C. STRAND SPACES

1. Strands

The Strand Space method is formal framework based on communicating state machines that can be used to construct and analyze cryptographic protocols. The method is described in several research papers by Fábrega, Herzog and Guttman. [Ref. 42, 14, 13] Figure 4 illustrates the syntax and semantics of the method by depicting the Needham-Schroeder-Lowe Public Key protocol, originally developed by Needham and Schroeder[Ref. 34], fixed by Lowe[Ref. 27] and expressed in Strand Spaces.[Ref. 42].

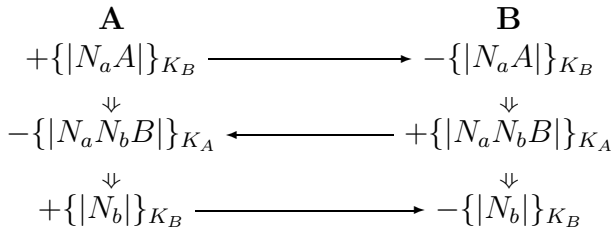


Figure 4. Needham-Schroeder-Lowe Public Key Protocol

The goal of the original Needham-Schroeder Public Key protocol is to establish mutual authentication between two participants, **A** and **B**. In order for two participants to establish mutual authentication, both must prove their identity to the other. The protocol achieves this by exchanging cryptographically transformed nonces. A *nonce* is a freshly generated value, usually created by a random number generator. By using the random number generator, the value of nonces is unpredictable. Nonces are used to identify unique instances of messages.

The participants are bound by roles of *initiator* and *responder*, respectively. Participant **A** initiates the protocol by sending a message to **B**. Acting as responder, **B** will continue to respond to **A** until the protocol is complete. The sequence of transmission and reception events for each participant are captured in a linear structure called a *strand*. Each time a message is sent or received a node is connected to the strand using a $\Downarrow$. The $\Downarrow$ defines the order in which communication events occur.

Each event is described by a tuple containing a symbol and a term. The *directional symbols*, + and -, say whether a term has been sent or received, respectively. *Terms* are the building block of messages. If **T** is the set of possible messages that can be exchanged between participants, then terms are elements of **T**. Nonces, names, keys and hashes are all types of terms. Terms can be concatenated together, encrypted or decrypted using keys to form new terms. Terms will be referred to as *positive* or *negative* depending on the directional symbol that precedes it. If no directional symbol precedes a term, then the term is *unsigned*.

For example, in Figure 4 above, the nonce, N_a, is concatenated with the name, A, and then encrypted using the key, K_B to form the term $\{|N_a A|\}_{K_B}$. In the Strand Space method, the symbol $\{|x|\}$ is used to denote encryption of term x. To form a transmission event, the + symbol is attached to the left of the term. The name, A, and the nonce, N_a are subterms of $\{|N_a A|\}_{K_B}$. A *subterm* relation is used to describe this relationship, where $t_1 \sqsubset t$ means that t_1 is a subterm of t.

Subterm Relation The subterm relation $\sqsubset$ is defined inductively, as the smallest

relation such that:

1. $a \sqsubset a$
2. $a \sqsubset \{|g|\}_K$ if $a \sqsubset g$;
3. $a \sqsubset gh$ if $a \sqsubset g$ or $a \sqsubset h$

A term t_0 is a *component* of t if $t_0 \sqsubset t, t_0$ is not a concatenated term, and every $t_1 \neq t_0$ such that $t_0 \sqsubset t_1 \sqsubset t$ is a concatenated term. So, components are either atomic values or encryptions. For example, $N_a A$ has two components while the term $\{|N_a A|\}_{K_B}$ is a single component.

Given these definitions, a Strand Space is a set Σ with a trace mapping $tr : \Sigma \rightarrow (\pm\mathbf{T})^*$, where $(\pm\mathbf{T})^*$ represents the possible sequences of tuples described above. For the example above, the traces for **A** and **B** would be described as in Figure 5. An *honest participant* is one that follows all the rules for their particular role, as

$$tr(\mathbf{A}) = \langle +\{|N_a A|\}_{K_B}, -\{|N_a N_b|\}_{K_A}, +\{|N_b|\}_{K_B} \rangle$$
and
$$tr(\mathbf{B}) = \langle -\{|N_a A|\}_{K_B}, +\{|N_a N_b|\}_{K_A}, -\{|N_b|\}_{K_B} \rangle$$

where,

1. *N_a, N_b are terms representing freshly generated nonces.*
2. *K_A, K_B are the public keys of principals A and B*
3. *$\{|M|\}_{K_B}$ expresses the encryption of term M using the public key of K_B. And similarly for K_A.*

Figure 5. Initiator and Responder Traces

defined by the protocol. Since **A** and **B** are assumed to be honest participants of the protocol, their strands are called *regular strands*. The actions of dishonest participants are represented by *penetrator strands*. A Strand Space that includes a penetrator strand is considered to be *infiltrated*. Together, both types of strands can be used to construct bundles.

2. Bundles

West[Ref. 43] defines a *directed graph* as triple consisting of a vertex set, edge set, and a function that assigns a pair of vertices to each edge. The first vertex is the tail of the edge, and the second vertex is the head. For example, let a graph G be $\langle V, E, R \rangle$ where V = {w, x, y, z}, E = {a, b, c, d} and the function is defined by the table in Figure 6. Using the table a directed graph can be drawn, as in Figure

Edges	*Tail*	*Head*
a	w	x
b	y	z
c	w	y
d	x	z

Figure 6. Graph Function

7, where an $\rightarrow$ is used to describe an edge going from one vertex to another. If a

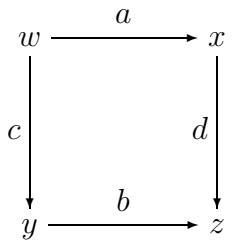


Figure 7. Graph Example

graph is formed using a subset of the vertex set and a subset of the edge set, then the graph is called a *subgraph* of the original graph. If the vertex set and edge set of the subgraph are finite, then the graph is called a *finite subgraph*.

Combining graphs and the Strand Space definitions, a protocol can be described as a bundle. A *bundle* is a finite directed graph that describes the causal relationships among participants of a protocol run.

Fix a Strand Space Σ. Let the vertex set, N, be the set of possible communication events in Σ. Vertices will be referred to as nodes under the Strand Space method. The edge set will be the union of two different edge types. The first set of edges, $\Rightarrow$ will be the edges used to connect nodes that immediately precede one another on the same strand. The other set of edges, $\rightarrow$, will be used to connect nodes on different strands. A subset of either edge set is denoted $\Rightarrow_X$ and $\rightarrow_X$, for edges belonging to the subset X. A bundle, therefore, is a finite acyclic directed graph whose vertex set is a finite subset of N, and whose edge set is formed by taking the union of two finite subsets of the two edge types. A formal definition follows.

Bundle:[Ref. 42] Suppose $N_C \subset N$, $\rightarrow_C \subset \rightarrow$ and $\Rightarrow_C \subset \Rightarrow$; and suppose $C = \langle N_C, (\rightarrow_C \bigcup \Rightarrow_C) \rangle$. C is a bundle if:

1. C is finite
2. If $n_2 \in N_C$ and $term(n_2)$ is negative, then there is a unique n_1 such that $n_1 \rightarrow_C n_2$.
3. If $n_2 \in N_C$ and $n_1 \Rightarrow n_2$ then $n_1 \Rightarrow_C n_2$.
4. C is acyclic.

Figure 4 illustrates a particular bundle for the Needham-Schroeder-Lowe protocol. A *path, p* through C is any finite sequence of nodes and edges. The notation $\Rightarrow^+$ can be used to denote the existence of a path between nodes on the same strand. The definition of a bundle describes a collection of communicating state machines with the following three properties:

1. A strand may send or receive a message, but not both at the same time
2. When a strand receives a message m, there is a unique node transmitting m from which the message was immediately received
3. When a strand transmits a message m, many strands may immediately receive m.

Since no attack has been discovered for the Needham-Schroeder-Lowe protocol, no penetrator strand is displayed. If an attack did exist, a penetrator strand could be described using the Dolev-Yao model of an intruder.

3. The Penetrator

To model actions of the Dolev-Yao intruder, the Strand Space method provides a framework for generating possible penetrator strands.[Ref. 42] Each capability of the intruder is characterized by a trace. The possible penetrator traces appear in Figure 8. Penetrator strands are formed by combining penetrator traces in such a

M. *Text message* : $\langle +t \rangle$ *where* $t \in \mathbf{T}$
F. *Flushing:* $\langle -g \rangle$
T. *Tee:* $\langle -g, +g, +g \rangle$
C. *Concatenation:* $\langle -g, -h, +gh \rangle$
S. *Separation into components:* $\langle -gh, +g, +h \rangle$
K. *Key* : $\langle +K \rangle$ *where* $K \in \mathbf{K_p}$
E. *Encryption:* $\langle -K, -h, +\{h\}_K \rangle$
D. *Decryption:* $\langle -K^{-1}, -\{h\}_K, +h \rangle$

where $\mathbf{T}$ *represents all possible text messages, and* $\mathbf{K_p}$ *are the keys known by the penetrator.*

Figure 8. Penetrator Traces

way that an attack can be generated.

Permutations, such as applying the penetrator traces multiple times, can lead to an unbounded penetrator Strand Space. An unbounded attack space makes verifying a cryptographic protocol impossible. Therefore, one needs to find a way of placing restrictions on the possible penetrator strands, so that all possible attacks can be represented and in addition the process is decidable.If this process can be accomplished, one can tractably prove the correctness of a protocol.

Bounding the possible penetrator strands is accomplished by restricting the order a penetrator can apply certain operations.[Ref. 14] The Strand Space method introduces a *Normal Form lemma* and an *efficiency condition* to limit the Penetrator's

Strand Space. The normal form lemma states that for any bundle there exists an equivalent bundle in normal form. A bundle is normal if, for any penetrator path in the bundle, every destructive edge precedes all constructive edges. Constructive edges are part of an **E** or **C** strand, while destructive edges are part of a **D** or **S** strand, see above Figure 8.

The authors of the Strand Space method prove that every penetrator strand can be represented in this normal form. Therefore, any properties that are proved under the normal bundle, also hold under the original bundle. The efficiency condition eliminates negative penetrator nodes where all or part of the terms have already been received by the penetrator. The Normal Form lemma and the efficiency condition lead to a bounded attack space. Guttman's conclusion is that the penetrator restrictions help identify certain components that the penetrator is unable to affect beyond denial of service.[Ref. 14] This result allows regular participants to reach certain conclusions about the origination of an encrypted component.

4. Cryptographic Assumptions

In order for Strand Spaces to accommodate a large set of cryptographic protocols, certain cryptographic assumptions are needed. [Ref. 13] The cryptographic assumptions state the cryptographic properties required of a cryptosystem under the Strand Space framework. For a public key cryptosystem the following assumptions have been made.

1. Public keys for any participant can be determined reliably, e.g. via a public key infrastructure or are distributed manually.
2. All private keys are safe, where safety insures that only the participants whose name is bound to the private key can read and use it.
3. Only the possessor of the private key, K, can tractably recover terms encrypted with K^{-1}.
4. Only the possessor of the private key, K, can tractably construct signed messages using uniquely originating terms.

A cryptographic assumption for hash functions is also required. Given a hash function *h(t)*, it assumed that no principal can tractably find a pair of values t_1, t_2 such that $h(t_1) = h(t_2)$, or given v, can tractably find t such that $h(t) = v$. Any cryptographic algorithm or hash function proven to abide by these assumptions is a possible candidate for use in a cryptographic protocol designed using the Strand Space method.

D. SUMMARY

Formal methods continue to provide valuable support for specifying and verifying cryptographic protocols. Logics of knowledge and belief and communicating state machines are the two formal approaches most often applied to the analysis of cryptographic protocols. The Strand Space method has a emerged as an easy-to-apply method based on the Dolev-Yao model. Strands, which describe communication events, form bundles, which describe protocols. The Strand Space method provides an abstraction for describing intruders and cryptographic primitives. By specifying a protocol using a formal method, like Strand Spaces, the specification of the protocol is clearer.

Next, the Strand Space method will be used to model the requirements of the secure open distribution protocol.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. REQUIREMENTS MODELING

Securing information is a human problem. Personal relationships are central to the policies that define how information will be handled. These relationships lead to the development of security policies. Before a cryptographic protocol can be designed, a requirements specification must be created that describes the assumptions the protocol makes about its environment and the security policies it should support.

Modeling requirements involves transforming the informal requirements to a formal language and providing evidence that the formal model captures the intent of the information requirements. In [Ref. 31] Catherine Meadows states that "Many problems with security protocols arise, not because the protocol as designed did not satisfy its requirements, but because the requirements were not well understood in the first place." In her survey on requirements modeling, she identified three properties a formal framework for expressing protocol requirements should satisfy.[Ref. 31]

1. It should be expressive enough to specify properties of interest.
2. It should be unambiguous, and preferably compatible with some system for formal analysis.
3. Most importantly, it should be easy to read and write.

A. FORMAL REQUIREMENTS MODELING

Security is a moving target. Even when modeling requirements for cryptographic protocols, security properties, such as authentication, are defined differently in different environments. The "What you know", "What you are", and "What you have" factors of authentication[Ref. 6] are applied in a variety of ways to authenticate "Who you are" in order to determine "What access you are allowed." Being able to express requirements, in whichever way the environment defines the security is an important capability for developing a requirements modeling language.

Traditionally, requirements modeling languages have been bound to the language of a particular formal analysis tool. Once the requirements had been defined in the language of the tool, verifiers could analyze the protocol using a general purpose tool, like a model checker or theorem prover. Specifying requirements this way relieved analysts from having to construct protocol-specific tools. For an analyst familiar with the particular tool, understanding the formal specification was trivial. But for the engineer, tasked with verifying that the formal model mapped to the original requirements, the formal specification language was cryptic and difficult to understand. The advantages of having an unambiguous formal language that was compatible with a formal analysis system, led to requirements models that couldn't be understood by those outside the analyst's domain. This led researchers to develop more user friendly modeling techniques that could still be analyzed using a formal framework.

B. STRAND SPACE REQUIREMENTS MODEL

A goal of the Strand Space method was to develop a formal analysis tool that could be used to specify and analyze protocols in a easily understandable manner. The graphical illustrations model an environment that more closely resembles the traditional computing environment. Those that aren't versed in the formal logic can still understand the message passing of even the most complex protocols, once they have been described in a Strand Space picture.

The difficult part of modeling requirements is describing the security properties of a particular requirement using only strands and the messages that connect them. Guttman has noted[Ref. 10] that agreement properties could be formalized by the relative placement of strands. Some success has been achieved using agreement properties to formally describe more sophisticated security properties.

1. Agreement Properties

Lowe believes that an authentication protocol is designed to ensure that one participant can assure the identity of the other participant engaged in the protocol.[Ref. 28] Other cases exist where authentication is achieved as long as a participant can determine that another is running the protocol, not necessarily with A. Lowe, therefore, describes four layers of agreement that can be used to identify authentication properties of a protocol.

Given a participant, **A**, the level of agreement is derived by what **A** can infer about another participant, **B**, upon completion of a protocol run. Listed in order of strength, the levels of agreement are,

1. Aliveness - **B** had been running the protocol.
2. Weak agreement - **B** had been running the protocol with **A**.
3. Non-injective agreement - Given a set of data values **ds**, **B** had been running the protocol with **A**, **B** was acting as responder, and the two participants agreed on the data values, **ds**.
4. Agreement - All previous definitions hold, and each such run of **A** corresponds to a unique run of **B**

A formal method capable of modeling Lowe's agreement properties will be able to capture authentication requirements of varying strengths.

2. Modeling Agreement Properties

The Strand Space method uses the relative placement of strands to model Lowe's agreement properties. For example, aliveness would be modeled by saying that for all strands of participant A, there exists a B strand. Since A only knows that B is running the protocol, not necessarily with A, the strands are not connected, but they are in the same bundle. Also, the B strand can take the form of any possible regular strand. The inability to provide definite structure to the bundle is, in effect, the weakness of the aliveness property.

The bundle, modeling agreement, would illustrate a much more definite structure. The two strands for A and B would be completely connected, and all data values would be instantiated with the same values for matching communication events. Figure 9 provide Strand Space illustrations for all of Lowe's agreement properties for a simple protocol. For protocols whose authentication requirements can be modeled as agreement properties, these structures can be applied.

C. SODP REQUIREMENTS

1. Common Criteria Requirements

The requirements for the secure open distribution protocol are describe by "Part 3: Security assurance requirements"[Ref. 21] of the Common Criteria for Information Technology Security Evaluation[Ref. 20] The assurance requirements define what precautions must be taken during the life-cycle of the product to achieve a certain assurance level. The Trusted Computing Exemplar project will adhere to the assurance requirements for a product achieving Evaluated Assurance Level (EAL) 7, the highest classification under the Common Criteria.

As a distribution mechanism for the TCX, the secure open distribution protocol must meet all delivery requirements stated in the Common Criteria. Within the Common Criteria, requirements are separated, first, into classes, and then into families. Different levels of requirements are described within each family. For example, the class "ADO: Delivery and operation"[Ref. 19] includes a specific family for delivery "DEL: Delivery". The Delivery family has three different assurance levels for delivery: Delivery procedures, Detection of modification, and Prevention of modification. For an EAL7 product, all requirements listed under "Prevention of modification"[Ref. 18] must be satisfied. Those requirements are,

1. The delivery mechanism shall provide technical measures for the prevention of modifications, or any discrepancy between the developer's master copy and the version received at the user site.

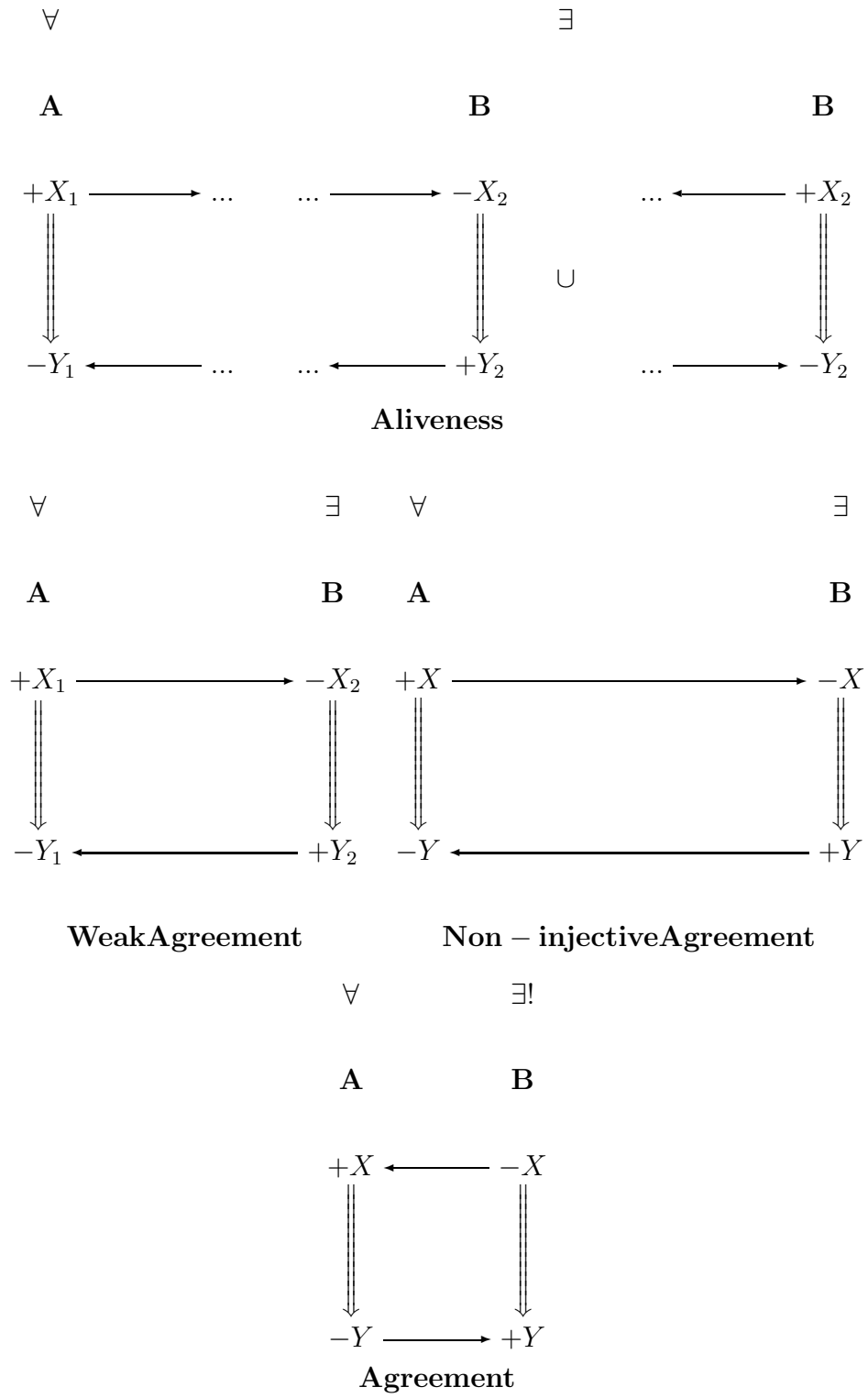


Figure 9. Lowe's Agreement Properties modeled using Strand Spaces

2. The delivery mechanism shall provide procedures that allow detection of attempts to masquerade as the developer, even in cases in which the developer has sent nothing to the user's site.

In order to prove that the secure open distribution protocol is correct, the requirements must be specified in a language that can be analyzed using the Strand Space method.

2. Modeling Common Criteria Requirements with Strand Spaces

The delivery requirements defined by the Common Criteria use a language similar to the language of Lowe's non-injective agreement properties. The first requirement states that no discrepancy may exist "between the developer's master copy and the version received at the user site." In other words, the two participants, the developer and user, must agree on the master copy.

The second requirement is a composition of two agreement properties. First, the user must detect "attempts to masquerade as the developer." Second, that detection of the event must be possible "even in cases in which the developer has sent nothing." Stated as an agreement property: Upon completion of a protocol run the user must agree that the developer had been running the protocol with the user. Combining the two requirements yields all three conditions for non-injective agreement. Since the user is only concerned that the contents of the message match the developer's master copy, replay is not an issue. So, full agreement isn't required. Therefore, the requirements for the secure open distribution protocol can be modeled using the Strand Space method representation for non-injective agreement.

Modeling the requirements of the secure open distribution protocol using Strand Spaces requires instantiating the variables of the particular agreement definition. In the definition for non-injective agreement, the participants were the initiator, A, and the responder, B. For the SODP protocol, these participants will be instantiated with the user, U, and the developer, D. The data value the user and

developer must agree on is the master copy of the document, M. Therefore, modeling the requirements of the SODP as a non-injective agreement property leads to the Strand Space diagram in figure 10. The diagram illustrates that for all user strands,

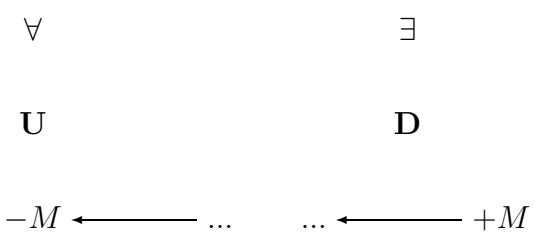


Figure 10. The SODP Requirements Model

where a document has been received, there must exist a matching developer strand where the master copy has been transmitted. This description follows directly from the definition of non-injective agreement.

D. SUMMARY

Requirements are created to enforce security policies. Formal requirements attempt to describe the elements of those policies and capture environmental assumptions. Specifying requirements in a user friendly format that describes the requirements accurately has been a struggle for formal methods. Strand Spaces appears to be an excellent candidate due to its graphical illustrations and flexibility. Lowe's agreement properties help define the characteristics of protocols that lead to authentication results. The relative placement of strands can be used to model Lowe's agreement properties. The Common Criteria defines the authentication requirements for the SODP. Non-injective agreement can be used to describe the authentication properties of the SODP. This result leads to a requirements model for the SODP using Strand Spaces. In the next chapter, this requirements model will be used to guide the design of the SODP.

THIS PAGE INTENTIONALLY LEFT BLANK

V. PROTOCOL DESIGN

Designing a correct protocol is an experimental exercise. A requirements model describes the properties a correct protocol will have, but not how to design a protocol that has those properties. Therefore, the protocol architect often must experiment with design choices to produce a correct protocol.

When performed in an ad hoc manner, design choices can lead to an incorrect protocol. But, if a designer can apply a design strategy demonstrated to produce correct protocols, errors are eliminated and and correctness can be reached sooner. Several different design approaches have been suggested over the past decade.

A. DESIGN PRINCIPLES

In 1996, Martin Abadi and Roger Needham proposed a design strategy to respond to basic mistakes that had been appearing in cryptographic protocols protocols.[Ref. 2] Many of the design flaws the authors had uncovered were the result of assumptions that had not been explicitly stated. Their solution was to provide a collection of design principles that described rules-of-thumb for engineering cryptographic protocols. Abadi and Needham believed applying the principles would result in secure protocols that could be analyzed without formal methods. The authors believed that informal design principles, not formal methods, would be a large part of the solution. Anderson and Needham would later extend the approach by introducing robustness principles for public key protocols.[Ref. 3].

Soon after both papers had been published, Paul Syverson, of the Naval Research Laboratory, explored the limitations of the design principle approach.[Ref. 41] In his report, Syverson found several examples where assumptions were made that had not been made explicit. For example, Syverson found the first principle of the Anderson-Needham approach to be based on an invalid assumption about the purpose behind using digital signatures. The authors of the design principles had claimed

that digital signatures are used to hold principles accountable to a trusted third-party. Several examples exist in literature where encrypting a signed message could provide security services other than non-repudiation.

Though limited in application, Syverson still thought the approach could be useful as an after-the-fact idiot check. He recommended that the user apply careful reasoning when using the principles to make design decisions. Though Needham and Abadi's guiding principles may not have provided the best solution, Swiss researches were able to combine his work in BAN logic and the Spi Calculus[Ref. 1] to develop a more formal design strategy.

B. LOGICAL DESIGN

In 1998, Levente Buttyán, Sebastian Staaman and Uwe Willhelm of the Swiss Federal Institute of Technology introduced a simple design strategy for authentication protocols.[Ref. 25] Using pieces of GNY logic, their flavor of BAN logic proposes synthetic rules that can be used to generate a formal protocol description, given security goals described in the language proposed by the others. After investigating the security goals specified in the requirements, an developer can choose the synthetic rules that best satisfies them.

Though the protocol designer is still left with a decision, the number of possibilities are greatly reduced. Once all goals have been satisfied, the construction of the protocol is straight forward. Though development of their strategy ended soon after the paper's publishing, their use of channels to represent encryption and decryption abstractly that was most intriguing.

The notion of channels can be found throughout protocol literature in many different forms. Lampson defined channels[Ref. 24], practically, as principals who can speak directly with a computer. Rueppel identifies the elementary security channels by the type of cryptographic protection that has been applied: physical, symmetric, and asymmetric. The authors of the previous design strategy characterize channels by

who can send and receive messages via the channel.[Ref. 25] Viewed in this manner, channels can describe cryptographic protocols using an information flow model. The difference from traditional information flow models, is that the participants enforce access control through encryption, rather than with an overarching security policy.

Public key cryptography can be used to determine members of the read and write sets that define the control of access to channels. When no encryption is used a *public channel* is formed where every principal is a member of the read and write set. On the other hand, if encryption is applied to data values both sets can be reduced.

For instance, if a public key is used to encrypt, and the private key is assume to be safe, then only a single participant, the owner of the private key, is a member of the read set. If a private key is used to encrypt, as is the case with a digital signature,the owner of the private key is the only member of the write set. A singular write set implies source authentication for all messages received on the channel. A singular read set implies secrecy for all messages sent on the channel. By authenticating the possible senders and receivers, channels can be constructed to provide security services.

C. DESIGN USING STRAND SPACES

1. Authentication Tests

The Strand Space method also provides a method for authenticating the source of messages called the authentication tests. In the previous discussion on channels, the encryption placed restrictions on who was capable of reading and writing certain messages. In the Strand Space method a channel is called a *cryptographic context* In the authentication tests, the same cryptographic properties used to construct channels can help determine which participant can move message between different cryptographic contexts. "Suppose a principal in a cryptographic protocol creates and transmits a message containing a new value v, later receiving v back in a different cryptographic context. It can be concluded that some principal possessing the rele-

vant key K has received and transformed the message in which v was emitted. If $K \in$ S is safe, this principal cannot be the penetrator, but instead must be a regular principal."[Ref. 13] The regular edges where the cryptographic transform occurs is called a *transforming edge*. The component that changes contexts is called an authentication test component.

Authentication Test Component $t = \{|h|\}_K$ is a *test component* for a term, a, in a node, n, if:

1. $a \sqsubset t$ and t is a component of n;
2. The term t is not a proper subterm of a component of any regular node $n' \in \Sigma$.

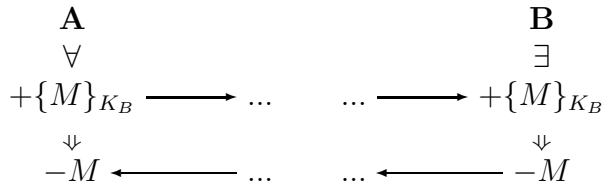
Authentication Test components produce two different tests: incoming and outgoing.

Outgoing test Suppose a uniquely originating value is transmitted in encrypted form where the decryption key is safe. If it is later received, outside the original cryptographic context, then a regular participant, not the penetrator, must have been responsible for transforming the value out of the original context. A Strand Space illustration of the outgoing test for a public key cryptosystem appears in Figure 11.

Incoming Test If instead, a value is received in encrypted form, where originally it was not sent in that cryptographic context, and the encryption key is safe, then a regular participant, must have been responsible the for placing the value in this context. A Strand Space illustration of the incoming test for a public key cryptosystem appears in Figure 12.

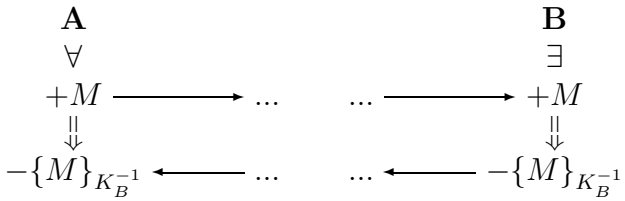
2. Using Authentication Tests to Model Agreement

The Authentication Tests provide a sufficient structure for achieving an authentication security goal under Lowe's non-injective agreement. In order for participant A to achieve non-injective agreement, given the structures above, the following three conclusions must be reached.



where M is a uniquely originating message, and K_B is the public key of a regular participant whose private key is safe and $\{M\}_{K_B}$ is a test component for term M.

Figure 11. Outgoing Authentication Test



where M is not necessarily uniquely originating, but is known to exist publicly. And K_B^{-1} is the safe private key of a regular participant with $\{M\}_{K_B^{-1}}$ a test component for M.

Figure 12. Incoming Authentication Test

1. B has been running the protocol with A
2. B was acting as responder
3. A and B agree on the message M

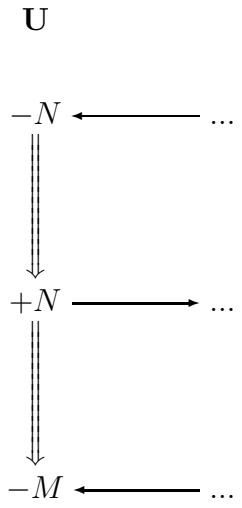
Therefore. when operating under the cryptographic assumptions, i.e. the safety of the keys, one can deduce information about the identity of a principal from the existence of transforming edges. No one, except B, is capable of the transformations illustrated in Figures 11 and 12. Therefore, when A establishes the existence of the the matching strand, the identity of the participant is trivially determined. And, A can conclude that B has been running the protocol.

In the Authentication Tests, participant A is implied as the initiator, which forces B to act as responder. Finally, once is has been established that a transforming edge exists on B's strand, A and B agree on the contents of the message.

D. DESIGNING THE SODP USING STRAND SPACES

The goal for designing the SODP is to construct a cryptographic protocol that achieves the structure proposed in the requirements model. Every protocol must begin with an initiator. The role of the initiator is to pass the first message, thereby initiating an instance of the protocol. For the TCX, since the user requires assurance, the user assumes the role of initiator. As the initiator, the user will send the name of the document, and expect to receive the requested document from the responder. Since the user must have, at some previous time, received the list of document names from the developer, a reception of the document list will also be included in the initiators strand. The corresponding initiator appears in Figure 13.

In order to achieve the security goals, the user needs a way to prove the existence of the responder's strand that sends the message M.



where, N ∈ Document Names and M ∈ Messages

Figure 13. Initiator strand without Encryption

The Authentication Tests can be used to produce the necessary responder strand. In the SODP, the initiator is attempting to authenticate the original source of the message. In either Authentication Test, the initiator achieves authentication

by inspecting a subsequently received message. Since the user needs to ultimately receive a message from the developer, an incoming Authentication Test could be used to achieve the authentication goal required for the SODP.

In the current version of the initiator strand, the document name, N, and the document message, M, are received unencrypted. If the private key of the developer were used to sign the name of the document and the message, N would be considered an incoming Authentication Test component. This would allow the name of the document, N, to bind the request to the document, M. The resulting initiator strand would look like Figure 14. Assuming the test was successful, a responder strand could be constructed, resulting in the final design of Figure 15. To prove that the incoming Authentication Test achieves the security requirements for the secure distribution protocol, i.e. the design is correct, formal verification is required.

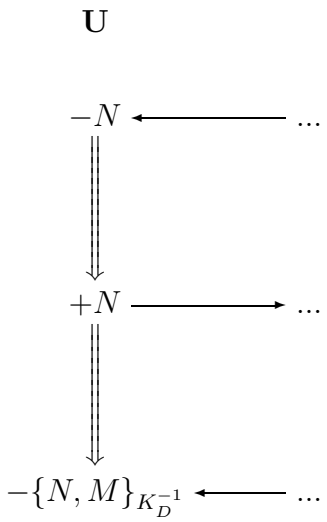


Figure 14. Initiator strand with Encryption

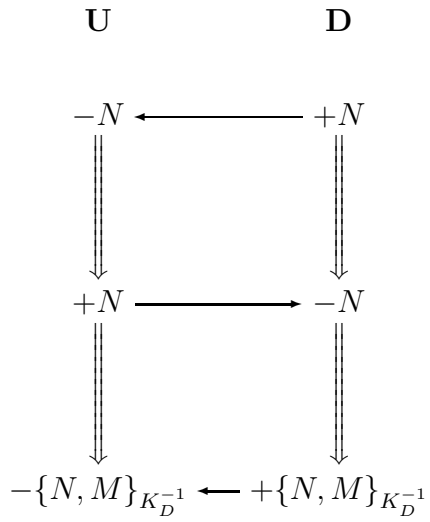


Figure 15. Secure Distribution protocol

E. SUMMARY

Designing a correct protocol involves experimenting with different design decisions. A design strategy can help with choosing a particular design. Design principles can be used as an informal design strategy, but more formal strategies produce more explicit results. Channels describe cryptographic protocols as an information flow model, restricted by who can read and write on a particular channel. The Authentication Tests, found in Strand Spaces, also place restrictions on information. The Authentication Tests allow a participant to make certain assumptions about who created certain components of a message. The Authentication Tests were used to produce a possible design for the SODP. In the next chapter, that design will formally verified using the Strand Space method.

THIS PAGE INTENTIONALLY LEFT BLANK

VI. FORMAL VERIFICATION

The goal of protocol verification is to provide the sufficient evidence that a protocol has satisfied a set of requirements. Sufficient evidence is difficult to determine. Each formal verification tool attempts to not only provide the necessary evidence, but also argue as to why their evidence sufficiently proves correctness.

For a protocol to be correct it must satisfy all conditions set forth by the requirements model. The most common form of evidence is a mathematical proof; with the requirement stated as a theorem and an explanation as to why a particular design proves the theorem. The other approach is to describe the system in its entirety. Once a person can see the entire system, careful inspection may determine that all requirements have been satisfied.

A. MODEL CHECKING

Model checkers verify protocols by exhaustively searching the state space of the communicating state machines for insecure states. A state is defined by a set of variables used describe the system. Transition rules govern how the variables change, which allows the system to move between states. Security goals are defined as a state, or set of states, the system should never reach.

For example, a secrecy goal could be described by as variable, that can be read by a attacker, which is instantiated with a value that is suppose to remain secret. Once the insecure state has been described, model checking can be used in either of two directions. In the forward direction, all the possible states are instantiated until either an insecure state is found or the space is exhausted. The other method is to work backward from insecure states to initial insecure conditions. In the first case one proves security by showing that no set of initial conditions can lead to an insecure state. In the second case one proves that any set of initial conditions that lead to an insecure state are impossible. Since the number of states can increase very rapidly

given only a few variables, model checkers, like the NRL Protocol Analyzer have been developed to automate the process and handle larger state spaces.

The NRL Protocol Analyzer is a prototype special-purpose verification tool.[Ref. 29] More generally, the Analyzer is a model checker. Based on the Dolev-Yao model, the Analyzer model checks in a backward direction using term-rewriting. As before, insecure states are described using variables. A combination of transition rules and reduction rules are used to give a complete description of all states that may precede the specified state. This process continues until either an attack is found, or until all selected states have been examined.

Since the search space is infinite the NRL Protocol Analyzer employs several mechanisms to bound the search. For example, the user can make partial queries that only search a portion of the state pace. The user may also specify a database of requirements on reachable states that can determine which states are reachable given certain conditions. This approach has not only led to new attacks, but proven correctness for many cryptographic protocols.

B. THEOREM PROVING

Theorem proving verifies cryptographic protocols by proving that a security result is true given certain conditions. Given a particular reasoning strategy, satisfying the conditions, or antecedents, allow the prover to reach a conclusion, or consequent. In BAN logic, the antecedents are a set of beliefs, and the theorems are the inference rules that allow one to generate new beliefs. In protocol verification, the requirements model is the consequent the theorem prover wishes to achieve. The antecedents, on the other hand, are more difficult to determine.

Each formal method has its own strategy for specifying antecedents and providing the evidence that proves the consequent. In the early days of protocol verification, proofs were hand written and protocol specific. Not only did the process take a considerable amount of time, but user dependence introduced error or uncertainties

that required further verification. The current approach is to offload much of the theorem proving to computers. This has led to the development of automated theorem provers, like PVS[Ref. 37] and Isabelle.[Ref. 16]

Automated theorem provers have provided mechanical aid to the verification of protocols. When a proof fails, the fault is usually because the details about bookkeeping are often assumed or ignored. Automated Theorem Provers address this failure by requiring the user to state the details of the protocol explicitly. This has been accomplished through the use of logic-based specification languages. The specification language forces the writer to state all necessary information in a precise formal statement.

Automated theorem provers also reduce the time to construct proofs. One method is to keep a library of previously proved theorems. Using query tools, the user can locate and apply these theorems without having to restate them. Once a theorem has been applied, all antecedents required for the theorem are automatically stated for the user. This prevents the user from reaching invalid conclusions, thereby reducing human error. If the user finds themselves performing a similar series of proof steps on occasion, the specification language can be used to develop a proof strategy for handling such cases. Much work has also been to improve the user friendliness of the proof output. For example, proof trees can be used to illustrate the logical structure of the proof.

C. PROTOCOL VERIFICATION USING STRAND SPACES

Theorem proving is the current approach for verifying protocols using Strand Spaces. Security goals are stated in the form of theorems from the perspective of a regular participant, such as "Agreement: The Responder's Guarantee" or "Secrecy: The Initiator's Nonce."[Ref. 42] The security requirements become the consequent of the theorem. Proving the consequent involves providing the required evidence that certain nodes do not exist on the penetrator's strand, and, therefore, are the

work of a regular participant. For example, proving a correspondence goal for the initiator involves constructing the matching responder's strand from the initiator's, and proving that the penetrator is unable to construct a similar strand.

D. VERIFYING THE SODP PROTOCOL

The Strand Space method will be used to verify the design of the secure open distribution protocol. While the secure distribution protocol shown in Figure 15 defines strands with a height of three, the first level only provides information to the user, but is not required to prove that the protocol is secure. As a matter of convenience, only the second and third levels of the protocol are considered below. Specifying the design given in chapter V leads to the following SODP Strand Space.

SODP Strand Spaces *Let Σ be an infiltrated Strand Space. Let P be a penetrator in Σ. Σ is an SODP space if it is the union of three kinds of strands:*

1. Penetrator strands $s \in P$;
2. "Initiator strands" with trace Init[N,M],*defined to be:*
   $< +N, -\{|N, M|\}_{K_D^{-1}} >$
   where $N \in$ possible document names and $M \in$ possible documents.
3. *Complementary "responder strands" with trace* Resp[N,M] *, defined to be:*
   $< -N, +\{|N, M|\}_{K_D^{-1}} >$
   where $N \in$ possible document names and $M \in$ possible documents.

As was described above, one must show that this Strand Space satisfies the non-injective agreement property. Per the requirements for the SODP, verifying non-injective agreement involves proving that the responder strand specified in the protocol design exists and is regular.

Non-injective agreement: The Initiator's Guarantee

Suppose:

1. Σ is an SODP space and C is a bundle containing an initiator strand with trace Init[N,M];
2. K_D^{-1} is safe

Then one must show that C contains an responder's strand with the trace Resp[N,M].

The proof will be performed by applying the results of the incoming Authentication Tests.

Proof: By definition of a test component, the edge $< s_i, 1 > \Rightarrow < s_i, 2 >$ is an incoming test for a term N where $< s_i, k >$ refers to the k^{th} node on the strand, s_i.

By the definition of an incoming Authentication Test, there exists regular nodes $m_1, m_2 \in \mathbf{C}$ such that N is a component of m_1. So, $m_1 \Rightarrow^+ m_2$ is a transforming edge for N. Since $\{N, M\}_{K_D^{-1}}$ is an incoming test component for the term N there must exist a negative regular node containing a component of the form -N. The only component of this form is $< s_r, 1 >$ for $s_r \in$ Resp[N,M]. Thus, the transforming edge for $m_1 \Rightarrow^+ m_2$ must be $< s_r, 1 > \Rightarrow^+ < s_r, 2 >$. Under the cryptographic assumption of safe keys, the developer is the only participant capable of producing s_r. Therefore, the developer was acting as responder in this run of the protocol.

E. SUMMARY

Protocol verification attempts to assure that a protocol design has satisfies a requirements model. The most common approaches for formal cryptographic protocol verification are model checking and theorem proving. Model checking describes the system as a state machine and checks that the model is correct. Theorem proving produces a proof that the design is correct. Automation has been applied to both methods, in order to increase efficiency and reduce error. The Strand Space method uses theorem proving to assure correctness of a cryptographic protocol. Using the Strand Space method, the SODP has been verified to achieve the requirements identified by the requirements model in chapter IV.

THIS PAGE INTENTIONALLY LEFT BLANK

VII. CONCLUSION

In this paper, a process has been proposed for developing cryptographic protocols. The process is composed of the following steps:

1. State informal requirements in terms of agreement properties (Chapter IV)
2. Model formal agreement properties using Strand Space structures (Chapter IV)
3. Apply the Authentication Tests to design a protocol that satisfies the Strand Space requirements model (Chapter V)
4. Prove the correctness of the design by demonstrating how the protocol satisfies the requirements model (Chapter VI)

The process begins with an informal statement about the requirements of the secure open distribution protocol. As the delivery mechanism for the Trusted Computing Exemplar project, the informal requirements are taken from the Common Criteria for Information Technology Security Evaluation. The requirements are to prevent the modification of the developer's master copy of a document that is distributed to the user.

The informal requirements are formally defined using Lowe's authentication definition. Since the user needs to authenticate the source of the document, the developer, and the integrity of the message, non-injective agreement is chosen to model authentication. A protocol that can meet the conditions of non-injective agreement will assure the user that he is running the protocol with the developer, and that the document received matches the developer's master copy.

Non-injective agreement is modeled using the Strand Space method. The Strand Space method is one form of the communicating state machine approach that models protocols graphically. In the Strand Space method, protocols are modeled as a bundle of strands. Strands connect to form bundles, which describe the communication events for every possible participant in a run of a protocol: honest and dishonest.

The causal relationships among strands appearing in the same bundle are illustrated as a finite directed graph. The graphical structure leads to some interesting proof methods and protocol specifications.

To model the non-injective agreement requirement for the SODP, a Strand Space illustration is used. Two strands are used to represent the participants. The relative placement of the strands states that for all user strands that receive a document, there is a developer strand that sent the message. Once the requirements for the SODP have been transformed from the informal definitions of the Common Criteria into a formal Strand Space definition, a design can be constructed.

The SODP is designed using the Authentication Tests; a Strand Space method for reasoning about cryptographic transformations. The Authentication Tests introduce three types of cryptographic transforms that, when appearing in cryptographic protocol, allow a participant to reach certain authentication conclusions. The test component for the incoming Authentication Test component is selected to provide information to the user about the integrity of the document and the identity of its source.

Proving that the specification of the SODP meets the requirements model, requires a formal proof. The SODP is verified using the axioms and definitions of the Strand Space Authentication Tests to develop a proof. The proof states that an initiator, the user in the case, can prove the existence of the responder's, or developer's, strand as defined by protocol specification for the SODP. Proving this result required the results derived from properties of the incoming Authentication Test and the structure of bundles. Proving the theorem proves that the design satisfies non-injective agreement as defined by the requirements model; resulting in a correct protocol.

The result of the development process proposed in this paper is a cryptographic protocol that can be used to securely distribute open source information. The Strand Space method has been illustrated as a viable option for the formal development of

a cryptographic protocol. The requirements of the SODP were formally modeled as agreement properties. The specification was aided by the Authentication Tests. And, the correctness of the SODP was was assured through the construction of a formal proof.

A. FUTURE WORK

Several further research opportunities have appeared during the construction of this work. The first is whether Lowe's agreement properties could be extended to model a secrecy property. In non-injective agreement, the participants are attempting to agree on a certain set of data values. If the participants are unable to agree on a data value, then there is no integrity for that value. If for instance the data value was the location of an enemy battle group, and non-injective agreement had not been established, then the receiving participant might choose not to believe, borrowing a term from BAN logic, that the location is accurate. This assumes that there exists a predefined trust relationship between the two participants in which the receiving party would believe the results if non-injective agreement was established. In the other direction, even if a penetrator was able to intercept and read the data value, believing its contents may also rely on the penetrators ability to reach non-injective agreement. Therefore, the lack of non-injective agreement might be able to model secrecy.

In this paper, only authentication properties are modeled using strand spaces. If Lowe's agreement properties were used to model secrecy, then that would include another property that could be modeled using Strand Spaces. Often Strand Spaces assures secrecy for a data value by proving that the data value could never appear on a penetrator strand. So, a requirements model for secrecy could look like the illustration in Figure 16. Strand Spaces might be able to model other security properties by using relative replacement for certain types of strands.

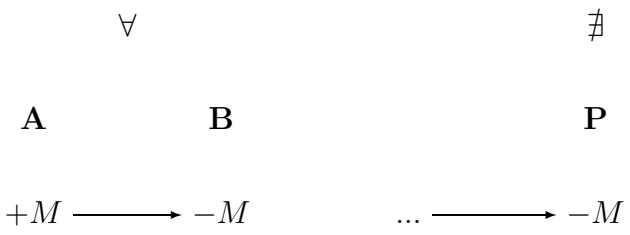


Figure 16. Secrecy Requirement Model

APPENDIX A. GLOSSARY

accept state a state that determines if a sentence is accepted by a finite automaton, 14

agreement an extension of non-injective agreement where a participant A and B agree that the instance of the protocol run was unique, 27

aliveness a type of authentication formalism where a participant, A, of a protocol agrees that a particular, B, was engaged in a run of the protocol not necessarily with A, 27

alphabet a set of valid of elements that can be processed by a particular finite automaton, 14

asymmetric cryptosystem a keyed cryptosystem where a different key is used to decrypt and encrypt, 4

Authentication Test component a Strand Space component that can be used to form an Authentication Test, 36

bundle a collection of strands that represents a run of a protocol, 19

cipher text a message whose contents have been altered by a cryptosystem, 4

component a Strand Space term that is either an atomic value or the result of an encryption, 18

cryptosystem a process for disguising the content of a message, 4

decrypt the process of transforming cipher text into plain text, 4

directed graph a triple consisting of a vertex set, an edge set, and a function that assigns a pair of vertices to each edge, 19

directional symbol a symbol that is used to label a Strand Space term to show that it is sent, +, or received, -, 17

efficiency condition a condition for reducing a Strand Space penetrator strand to a more efficient form, 21

encrypting the process of transforming plain text into cipher text, 4

finite automaton a model used for describing certain types of Computer Science concepts, 14

finite state transducer a finite automaton whose output is a sentence, 15

finite subgraph a graph whose edge set and vertex set are finite, 19

honest participant a participant who follows the rules of the protocol as they pertain to the participant's role, 18

incoming test an Authentication Test where an encrypted component that was sent is received in an unencrypted form, 36

infiltrated Strand Space a Strand Space that includes a penetrator strand, 18

initiator the participant that begins a protocol instance, 17

negative term a Strand Space term that is received, 17

non-injective agreement a type of authentication formalism where a participant, A, of a protocol agrees that a participant, B, was acting as a responder in a run of the protocol with A, where both participants agree on a set of data values, 27 , 27

nonce a freshly generated value used to identify unique instances of messages in a protocol, 17

Normal Form lemma a condition on the Strand Space penetrator strand that restricts the order that certain operations can occur, 21

outgoing test an Authentication Test where a component that was sent in an encrypted form is received in unencrypted form, 36

path through C any finite sequence of nodes and edges, 20

penetrator strand a strand that represents the actions of dishonest participants in a protocol, 18

plain text a message that is in its original form, 4

positive term a Strand Space term that sent, 17

regular strand a strand that represents the actions of honest participants in a protocol, 18

responder the participant that responds to the initiators request to begin an instance of the protocol, 17

sentence a sequence of elements from an alphabet, 14

start state the initial state of a finite automaton, 14

strand a sequence of communication events that represents the local view of a protocol run, 17

Strand Space the set of possible strands for a given protocol, 18

subgraph a graph whose vertex set and/or edge set are a subset of another graph, 19

subterm a term that is part of another term, 17

symmetric cryptosystem a keyed cryptosystem where the same key is used to encrypt and decrypt, 4

term an element of the set of possible messages for a given protocol, 17

transition function the function used to determine how a finite automaton changes state given a element of the alphabet, 14

unsigned term a term that has no directional symbol assigned to it, 17

weak agreement a type of authentication formalism where a participant, A, of a protocol agrees that a participant, B, was engaged in a run of the protocol with A, 27

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

[1] M. Abadi and A. D. Gordon. A calculus for cryptographic protocols: The spi calculus. In *Fourth ACM Conference on Computer and Communications Security*, pages 36–47. ACM Press, 1997.

[2] M. Abadi and R. Needham. Prudent engineering practice for cryptographic protocols. *IEEE Transactions on Software Engineering*, 22(1):6–15, 1996.

[3] R. Anderson and R. Needham. Robustness principles for public key protocols. *Lecture Notes in Computer Science*, 963:236–247, 1995.

[4] S. Bradner. RFC 2026: The Internet standards process — revision 3, Oct. 1996.

[5] R. Carnap. *The Logical Syntax of Language*. Routledge and Kegan Paul, London, 1937.

[6] S. Convery. *Network Security Archiectures*, pages 328–329. Cisco Press, Indianapolis, IN, 2004.

[7] Some rights reserved. http://creativecommons.org/learn/aboutus/, 2004. Last visited: September 2004.

[8] T. E. L. Cynthia E. Irvine and G. W. Dinolt. Diamond high assurance security program: Trusted computing exemplar. Technical Report NPS-CS-02-2004, Naval Postgraudate School, 2002.

[9] D. Dolev and A. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(3):198–208, 1983.

[10] J. C. H. F. Javier Thayer Fábrega and J. D. Guttman. Strand space pictures, 1998.

[11] Federal Aviation Adminstration budget. http://www.dot.gov/bib2004/faa.html, 2004. Last visited: September 2004.

[12] GNU General Public License. http://www.fsf.org/copyleft/gpl.html, 1991. Last visited: September 2004.

[13] J. Guttman. Security protocol design via authentication tests. In *15th IEEE Computer Security Foundations Workshop*, pages 92–103, 2002.

[14] J. D. Guttman and F. J. Thayer. Authentication tests. In *IEEE Symposium on Security and Privacy*, pages 96–109, 2000.

[15] M. Heimdahl and C. Heitmeyer. Formal methods for developing high assurance computer systems: Working group report. In *Second IEEE Workshop on Industrial Strength Formal Techniques*, 1998.

[16] Isabelle. http://www.cl.cam.ac.uk/Research/HVG/Isabelle/, 2004. Last visited: September 2004.

[17] ISO. Information processing systems - Open Systems Interconnection - basic reference model: The basic model. Technical report, International Standards Organisation, 1994.

[18] ISO. ADO_DEL.3: Prevention of modification. In *Part 3: Security assurance requirements*, page 87. National Institute of Standards and Technology, 1999.

[19] ISO. Class ADO: Delivery and operation. In *Part 3: Security assurance requirements*, chapter 9. National Institute of Standards and Technology, 1999.

[20] ISO. Common criteria for information technology security evaluation, 1999. version 2.1.

[21] ISO. Part 3: Security assurance requirements. In *Common criteria for information technology security evaluation*. National Institute of Standards and Technology, 1999.

[22] S. C. J.K. Millen and S. Freedman. The interrogator: Protocol security analysis. *IEEE Transactions on Software Engineering*, SE-13(2), 1987.

[23] S. Kent and R. Atkinson. RFC 2401: Security architecture for the Internet Protocol, Nov. 1998.

[24] B. Lampson, M. Abadi, M. Burrows, and E. Wobber. Authentication in distributed systems: Theory and practice. *ACM Transactions on Computer Systems*, 10(4):265–310, 1992.

[25] S. S. Levente Buttyán and U. Wilhelm. A simple logic for authentication protocol design. In *Proceedings of the 11th Computer Security Foundations Workshop*. IEEE Computer Society Press, 1998.

[26] D. Longley and S. Rigby. An automatic search for security flaws in key management schemes. *Computers and Security*, 11(1):75–90, 1992.

[27] G. Lowe. Breaking and fixing the needham-schroeder public key authentication protocol. *Lecture Notes in Computer Science*, 1055:147–166, 1997.

[28] G. Lowe. A hierarchy of authentication specifications. In *Proceedings of The 10th Computer Security Foundations Workshop*. IEEE Computer Society Press, 1997.

[29] C. Meadows. The NRL protocol analyzer: An overview. *Journal of Logic Programming*, 26(2):113–131, 1996.

[30] C. Meadows. Analysis of the internet key exchange protocol using the nrl protocol analyzer. In *Proceedings of the 1999 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, 1999.

[31] C. Meadows. What makes a cryptographic protocol secure? the evolution of requirements specification in formal cryptographic protocol analysis. In *Proceedings of ESOP*. Springer-Verlag, 2003.

[32] C. A. Meadows. Formal verification of cryptographic protocols: A survey. In *ASIACRYPT: Advances in Cryptology – ASIACRYPT: International Conference on the Theory and Application of Cryptology*. LNCS, Springer-Verlag, 1995.

[33] M. A. Michael Burrows and R. Needham. A logic of authentication. *ACM Transactions in Computer Systems*, 8(1):18–36, 1990.

[34] R. Needham and M. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12), 1978.

[35] J. Postel. RFC 791: Internet Protocol, Sept. 1981.

[36] J. Postel. RFC 793: Transmission control protocol, Sept. 1981.

[37] PVS specification and verification system. http://pvs.csl.sri.com/, 2004. Last visited: September 2004.

[38] C. M. Richard Kemmerer and J. Millen. Three systems for cryptographic protocol analysis. *Journal of Cryptology*, 7(2), 1994.

[39] M. Sipser. *Introduction to the Theory of Computation*. PWS Publishing Company, Boston, 1997.

[40] P. Syverson. The use of logics in the analysis of cryptographic protocols. In *IEEE Symposium on Research in Security and Privacy*, Oakland, CA, 1991. IEEE Computer Society Press.

[41] P. Syverson. Limitations on design principles for public key protocols. In *IEEE Symposium on Security and Privacy*, pages 62–73, Oakland, CA, 1996. IEEE Computer Society Press.

[42] J. Thayer, J. Herzog, and J. Guttman. Strand spaces: Proving security protocols correct. *Journal of Computer Security*, 1999.

[43] D. B. West. *Introduction to Graph Theory - Second edition*. Prentice Hall, New York, 2001.

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, VA

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, CA

3. Dr. George Dinolt
   Computer Science Department
   Naval Postgraduate School
   Monterey, CA

4. Dr. Timothy Levin
   Computer Science Department
   Naval Postgraduate School
   Monterey, CA

5. Dr. Cynthia Irvine
   Computer Science Department
   Naval Postgraduate School
   Monterey, CA

6. Dr. Sylvan S. Pinsky
   NSA
   Fort Meade, MD

7. Catherine Meadows
   Naval Research Laboratory
   Washington, DC

8. Gautam Trivedi
   Naval Research Laboratory
   Washington, DC

9. Joshua D. Guttman
   The MITRE Corporation
   Bedford, MA

10. Jonathon Herzog
    The MITRE Corporation
    Bedford, MA

11. Dr. Shirley Wilson
    North Central College
    Naperville, IL

12. Jason Rogers
    Naval Postgraduate School
    Monterey, CA