Wesleyan University The Honors College

Identity Authentication and Secrecy in the πCalculus and Prolog

by

Stefan Sundseth

A thesis submitted to the
faculty of Wesleyan University
in partial fulfillment of the requirements for the
Degree of Bachelor of Arts
with Departmental Honors in Mathematics and Computer Science

Middletown, Connecticut April 2013

Contents

CHAPTER 1

Introduction

In our current state of technology, more and more private communications are taking place over digital channels. The nature of these channels often makes them available to the public, so the secrecy of such communications is dependent upon algorithms which generate and utilize encryption schemes. These "communication protocols" allow parties to exchange sensitive information over public channels with little fear of interception. However, discovery of a flaw in these protocols can lead to computer fraud resulting in the loss of large sums of money. Clearly it is necessary to be able to prove the secrecy of existing protocols and detect errors in new ones.

Before properties of protocols could be proved, a rigorous method to describe them was needed. To this end process calculi(e.g. [**2**]) were developed which included ways to formalize the notions of secrecy, encryption, authentication, etc; as well as what forms an attack on protocols could take. Equipped with such calculi, researchers began to investigate ways to automate the process of proving the properties of protocols

To this end, several programming calculi have been developed for the study of protocols(e.g. [**1**] [**4**]...). These calculi establish security properties of the protocols which they describe by transforming their process calculus constructs into programs executable by a computer.

Another approach to analyzing protocols was developed by Michael Burroughs, Martin Abadi, and Roger Needham [**5**]. They introduce a 'Logic of Authentication'(BAN after the author's names) which contains logical connectives such as *believes, sees, said* to create statements of the form 'principal A believes that B said m'. Through a process they call *idealization* a protocol is transformed into a series of logical statements, and an execution of a protocol is a series of updates to the beliefs of the principals.

Though BAN provided new insights into reasoning about the security of protocols, it is not without flaws. Most important of these is the concern that *idealization* may transform an insecure protocol into one that is sound. An example of this is the fact that BAN as it was first written does not point out the flaw in the Needham-Schroeder protocol, which is known to be insecure to a man-in-the-middle attack.

In this thesis we examine a method of study developed by Martin Abadi and Bruno Blanchet [**3**]. They establish a translation between the πCalculus of Robin Milner [**8**](a process calculus for describing communicating systems), a type-system describing *public* and *secret* types, and logic-programming(of which the most notable language is Prolog).

To prove that their translation is correct they first show that if a protocol P is typed by an environment E(built by their type system), and if E maps a name s to the defined 'secrecy-type'; then P protects the secrecy of s. Abadi and Blanchet then prove that from the Prolog translation of a protocol can be built a suitable instance of their type system, which in turn shows that the protocol is secure.

In this thesis we take a different approach to proving their translation correct. Rather than using the type system, we show directly that if the Prolog program does not prove the attacker's knowledge of a secret, then the protocol from which the program was built is secure.

In chapter 2 of this thesis we describe the syntax and reductions of the πCalculus, using these to formally show the flaw in the Needham-Schroeder public key authentication protocol. We also introduce a modified Needham-Schroeder to present a way to transform its failure of identity authentication into a failure of secrecy. Rather than using BAN's logical rules for 'belief', we adopt the notion that if a principal A correctly authenticates his identity to principal B then B will release a secret to A. Thus if B is deceived then the failure of authentication leads to a secret being released to an attacker. In chapter 3 we present Abadi and Blanchet's translation of protocols into Prolog, and as an example give a translation of our modified Needham-Schroeder. Finally, in chapter 4 we prove Abadi and Blanchet's secrecy result which relates a protocol to its Prolog translation.

CHAPTER 2

Protocols in the πCalculus

1. πCalculus Introduction and Syntax

The πCalculus was created by Robin Milner [**8**] as a rigorous treatment of communication protocols between computational systems. It focuses on the concept of a name, and is unique in its notion of mobility: anything, even the names of channels, can be sent as messages. Thus it can describe networks which 'mobilize' by sending new channels to components, reconfiguring themselves in the process. As we will see, this treatment of communication adapts very well to describing security protocols.

The syntax of the πCalculus is simple, and protocols are built up as follows:

π	$::=$	$x(y)$	receive message y on channel x
	$\mid$	$\bar{x}\langle y\rangle$	send message y on channel x
	$\mid$	τ	unobservable computation performed by a component
	$\mid$	0	do nothing
P	$::=$	$\Sigma_{i\in I}\pi_i.P_i$	finite number of sequential computations
			$P+Q$ means execute either P or Q
	$\mid$	$P \mid P$	parallel computation
	$\mid$	$(\nu\ a)P$	bind the name a in P
	$\mid$	$!P$	parallel compose P with itself a finite number of times

In order to more easily model security protocols, the πCalculus is enriched with the following syntax:

- Constructors $\mathit{pencrypt}(x,k)$ and $\mathit{pk}(k)$ where $\mathit{pencrypt}(x,k)$ means 'the message x encrypted by key k' and $\mathit{pk}(k)$ means 'the public key generated from secret key k'.

- Destructor $\mathit{pdecrypt}(x, k)$ which means 'the decryption of message x with key k. Combining this with constructors, we have $m = \mathit{pdecrypt}(\mathit{pencrypt}(m, \mathit{pk}(k)), k)$.
- The local definition $\mathit{let}\ x = M$ in P executes P with the variable x bound to the term M.
- The conditional if $M = N$ then P else Q executes P if M and N reduce to the same term at runtime; otherwise it executes Q.
- The combination of the above two constructs with a destructor: $\mathit{let}\ x = \mathit{pdecrypt}(M, k)$ in P else Q. This executes P if M is of the form $(\mathit{pencrypt}(x, \mathit{pk}(k)))$, Q otherwise.

To formalize the notion of computation in the πCalculus, we have the following reaction rules which describe the relation $\rightarrow$ over protocols. This reduction relation is defined as the least relation closed under the following set of rules:

I/O	:	$x(y).P \mid \bar{x}\langle z\rangle.Q \rightarrow P\{z/y\} \mid Q$
Tau	:	$\tau.P \rightarrow P$
*let	:	let $x = \mathit{pdecrypt}(M, k)$ in P else $Q \rightarrow P\{M'/x\}$ where $\mathit{pdecrypt}(M, k) = M'$
*let	:	let $x = \mathit{pdecrypt}(M, k)$ in P else $Q \rightarrow Q$ when $M \notin \mathit{def}(\mathit{pdecrypt})$
		(the two *let rules are not formal rules of the πCalculus,
		but they should be intuitive reductions for our extension)

And finally we have rules for the structural congruence, $\equiv$, between protocols. $P \equiv Q$ if we can transform one into the other using the following:

(1) alpha-conversion (changing bound names)
(2) reordering terms in a summation
(3) $P \mid Q \equiv Q \mid P, P \mid (Q \mid R) \equiv (P \mid Q) \mid R$
(4) $(\nu\ a)P \mid Q \equiv (\nu\ a)(P \mid Q)$ if $a \notin fn(Q)$

We will tacitly use congruence rules when computing the reaction of terms. We write $\rightarrow^*$ for the reflexive transitive closure of $\rightarrow \cup \equiv$.

We call this enriched calculus the π^*Calculus[1] which, though simple, is powerful and useful for describing the security protocols we are interested in. The reduction rules stated above allow us to step through the execution of a protocol, observing what states it can reach and whether those states protect secrecy. The definition of insecure states is especially important, as proofs that a protocol is secure must show that those states cannot be reached.

Intuitively, an insecure state is one in which a secret is sent on a channel known to an attacker, or as an encrypted message to which the attacker has the key. However, to simplify our definition, we assume that once the attacker has the means to access a secret he will publish it on a public channel. Thus we know a protocol is insecure if a secret is sent out on a public channel. Now we can state the definition of secrecy.

Definition. Let S be a set of names, let $c \in S$, and let $s \notin S$. A protocol P protects the secrecy of s in S if there exists no protocol Q such that $P \mid Q \to^* (\nu\ a_1)...(\nu\ a_n)\bar{c}\langle s\rangle.P' \mid Q'$.

Here the set S is all public names used by a protocol, so this definition means that if a protocol P uses names from a set S then there is no attacker Q which can lead P to a state where the secret is output on a public channel. A public channel is one whose name is in S, we often use the name c to denote the public channel of a protocol. The protocol may also have some number n of variables bound by (ν). We assume that Q does know all the names in S, but does not know the secret initially. For the rest of this thesis, we refer to such a state as *bad*. In the next section, we provide an example of how a protocol can reduce to *bad*, thus revealing its failure of secrecy.

2. Needham-Schroeder

A popular protocol for study is Needham-Schroeder due to its simplicity and its flaw in security, namely to a man-in-the-middle attack. It is an *identity-authentication* protocol comprising two principals who we will call A and B. Each seeks to establish

[1]For the rest of this paper, we refer to the π^*Calculus as the πCalculus

the identity of the other, thus forming a sort of 'trust' between them based only on the contents of messages they receive. In order to confirm each other's identity they send a series of encrypted messages which, if proper responses are sent, culminate in a belief that the other principal is who he says he is.

Assume A and B have public keys pkA and pkB, to which correspond secret keys skA and skB. Recall that we denote the encryption and decryption of messages through the *pencrypt* constructor and *pdecrypt* destructor. where $pencrypt(x, y)$ means "x encrypted with the key y" and $pdecrypt(x, y)$ means "the decryption of x using key y". For A and B, a message encrypted with their public key can only be decrypted by the corresponding secret key. Thus $pdecrypt(pencrypt(x, pkA), ska) = x$. Before we describe Needham-Schroeder in the πCalculus let's examine it in the common syntax to get an idea of what messages are sent and received. We describe formally the correspondance between A and B with statements of the form $A \rightarrow B : m$, meaning "A sends B the message M".

$$\begin{array}{lll} (1) & A \rightarrow B & : & pencrypt((N_a, pkA), pkB) \\ (2) & B \rightarrow A & : & pencrypt((N_a, N_b), pkA) \\ (3) & A \rightarrow B & : & pencrypt(N_b, pkB) \end{array}$$

(1) A sends to B a nonce, a value he creates and no one else knows, along with his public key; the entire message is encrypted with B's public key.
(2) B decrypts A's message and returns A's nonce along with a nonce of his own.
(3) A decrypts this message and sends B his nonce back, at which point they are both convinced of the other's identity.

Part of this establishment of trust is the belief that "If I encrypt a message with X's public key then only X can decrypt and read that message." However, we see in the following description of a man-in-the-middle attack that this is not the case. This attack was published in 1995 by Gavin Lowe [**7**], over a dozen years after Needham-Schroeder had been put into common practice. Let us assume that A and B have a colleague E who they trust but is secretly dishonest. The attack begins when A initiates Needham-Schroeder with E, who then initiates a session with B by impersonating A:

$$\begin{array}{llcl}
(1) & A \rightarrow E & : & pencrypt((N_a, pkA), pkE) \\
(2) & E \rightarrow B & : & pencrypt((N_a, pkA), pkB) \\
(3) & B \rightarrow E & : & pencrypt((N_a, N_b), pkA) \\
(4) & E \rightarrow A & : & pencrypt((N_a, N_b), pkA) \\
(5) & A \rightarrow E & : & pencrypt(N_b, pkE) \\
(6) & E \rightarrow B & : & pencrypt(N_b, pkB)
\end{array}$$

(1) A initiates Needham-Schroeder with E, sending his public key along with a nonce.
(2) E begins his deception by decrypting the message and sending it to B encrypted with B's public key.
(3) B sees A's public key in the message, so he sends back A's nonce and his own encrypted with A's public key.
(4) E cannot decrypt this message, so he sends it straight to A as is.
(5) A decrypts the message and sees his own nonce, so he sends B's nonce to E encrypted with E's public key.
(6) E can then decrypt A's message and send B his nonce back. At this point B believes he is talking to A when in fact it is E who sent him his nonce back.

Examining these lines shows us some drawbacks to the common syntax: it tells nothing about where the contents of messages came from (e.g. N_a and N_b), it does not describe what processing is performed upon a received message, and it lacks a way to say "B believes he is talking to A". The πCalculus, however, provides syntax for the rigorous treatment of the role each principal plays if sent certain messages.

Writing Needham-Schroeder in this calculus, we adopt some conventions. Variables begin with capital letters and bound names begin with lowercase letters, Prolog style. For ease of reading, variables in *input* or *let* statements are named to represent what message is expected.

Thus to illustrate the flaw in Needham-Schroeder using the πCalculus, we present two modifications of its sequence of messages. Both modifications consist of B showing trust in his interlocutor by sending out a secret, s, on some channel. In the first modification, NS1, A sends B his nonce back along with the name of a channel, if B confirms that the nonce is his own he then sends out the secret along that channel. Table 1 shows the description of this protocol, P, in the πCalculus. It is composed of three parts: a server P_0 which creates keys and tells the other parts who they will be communicating with, and participants A and B. Thus for a complete protocol we have $P = P_0 \mid A \mid B$.

The second modification, NS2, makes use of the πCalculus' rule that everything is a name, including channels. So instead of A sending B a channel on which to output s, B simply uses his nonce as a channel. Since A should be the only other person who knows B's nonce, only A can listen on that channel and receive the secret. This version of Needham-Schroeder is shown as protocol $P' = P'_0 \mid A' \mid B'$ in Table 2.

In both tables there are protocols E and E', which represent attacks on protocols P and P' respectively. The attacks can be written in the πCalculus as $P \mid E$ and $P' \mid E'$. We illustrate a trace of the attack on P' in Table 3 to demonstrate how the protocol reduces to a state where the attacker possesses a secret and is thus insecure. Of note is the fact that E' uses $B's$ trust to obtain a secret and proceeds to publish that secret on a public channel. We also make use of the congruence rule $(\nu\ a)P \mid Q \equiv (\nu\ a)(P \mid Q)$ where $a \notin fn(Q)$. In Table 3 we leave the binding constructs to improve readability, but remember that they would be extruded from the scope of $P' \mid E'$ to $(\nu\ \ldots)(P' \mid E')$. By this attack we have shown how a failure in identity authentication can lead to a failure of secrecy.

$$
\begin{array}{lll}
P_0 & \triangleq & (\nu\ skA)(\nu\ skB) \\
 & & \text{let } pkA = pk(skA) \text{ in} \\
 & & \text{let } pkB = pk(skB) \text{ in} \\
 & & \bar{c}\langle pkA\rangle.\bar{c}\langle pkB\rangle \\
\text{A} & \triangleq & c(PKX). \\
 & & (\nu\ n_a). \\
 & & \bar{c}\langle pencrypt((n_a, pkA), PKX)\rangle \\
 & & c(Nonces). \\
 & & \text{let } (N, N_b) = pdecrypt(Nonces, skA) \text{ in} \\
 & & \text{if } N = n_a \text{ then } (\nu\ c_a). \\
 & & \bar{c}\langle pencrypt((N_b, c_a), PKX)\rangle \\
\text{B} & \triangleq & (\nu\ n_b). \\
 & & c(PKY). \\
 & & c(MessA). \\
 & & \text{let } (N_a, PKZ) = pdecrypt(MessA, skB) \text{ in} \\
 & & \text{if } PKZ = PKY \text{ then} \\
 & & \bar{c}\langle pencrypt((N_a, n_b), PKY)\rangle. \\
 & & c(NC). \\
 & & \text{let } (N, C) = pdecrypt(NC, skB) \text{ in} \\
 & & \text{if } N = n_b \text{ then } \bar{C}\langle s\rangle \\
\text{E} & \triangleq & c(PKA).c(PKB). \\
 & & \bar{c}\langle pkE\rangle.\bar{c}\langle PKA\rangle. \\
 & & c(MessA). \\
 & & \text{let } (N_a, PKA) = pdecrypt(MessA, skE) \text{ in} \\
 & & \bar{c}\langle pencrypt((N_a, PKA), PKB)\rangle. \\
 & & c(MessB). \\
 & & \bar{c}\langle MessB\rangle. \\
 & & c(NC). \\
 & & \text{let } (N_b, C) = pdecrypt(NC, skE) \text{ in} \\
 & & \bar{c}\langle pencrypt((N_b, C), PKB)\rangle.C(Secret).\bar{c}\langle Secret\rangle
\end{array}
$$

TABLE 1. NS1 with an attacker

$$\begin{aligned}
P'_0 &\triangleq (\nu\ skA)(\nu\ skB)\\
&\quad \text{let } pkA = pk(skA) \text{ in}\\
&\quad \text{let } pkB = pk(skB) \text{ in}\\
&\quad \bar{c}\langle pkA\rangle.\bar{c}\langle pkB\rangle\\
A' &\triangleq c(PKX).\\
&\quad (\nu\ n_a).\\
&\quad \bar{c}\langle pencrypt((n_a, pkA), PKX)\rangle\\
&\quad c(Nonces).\\
&\quad \text{let } (N, N_b) = pdecrypt(Nonces, skA) \text{ in}\\
&\quad \text{if } N = n_a \text{ then}\\
&\quad \bar{c}\langle pencrypt(N_b, PKX)\rangle\\
B' &\triangleq (\nu\ n_b).\\
&\quad c(PKY).\\
&\quad c(MessA).\\
&\quad \text{let } (N_a, PKZ) = pdecrypt(MessA, skB) \text{ in}\\
&\quad \text{if } PKZ = PKY \text{ then}\\
&\quad \bar{c}\langle pencrypt((N_a, n_b), PKY)\rangle.\\
&\quad c(NC).\\
&\quad \text{let } N = pdecrypt(NC, skB) \text{ in}\\
&\quad \text{if } N = n_b \text{ then } \bar{n_b}\langle s\rangle\\
E' &\triangleq c(PKA).c(PKB).\\
&\quad \bar{c}\langle pkE\rangle.\bar{c}\langle PKA\rangle.\\
&\quad c(MessA).\\
&\quad \text{let } (N_a, PKA) = pdecrypt(MessA, skE) \text{ in}\\
&\quad \bar{c}\langle pencrypt((N_a, PKA), PKB)\rangle.\\
&\quad c(MessB).\\
&\quad \bar{c}\langle MessB\rangle.\\
&\quad c(NC).\\
&\quad \text{let } N_b = pdecrypt(NC, skE) \text{ in}\\
&\quad \bar{c}\langle pencrypt(N_b, PKB\rangle.N_b(Secret).\bar{c}\langle Secret\rangle
\end{aligned}$$

TABLE 2. NS2 with an attacker

TABLE 3. Attack on Option 2: $P' \mid E'$

$P'_0 \triangleq (\nu\ skA)(\nu\ skB)$ let $pkA = pk(skA)$ in let $pkB = pk(skB)$ in $\bar{c}\langle pkA\rangle.\bar{c}\langle pkB\rangle$	E' $\triangleq c(PKA).c(PKB).$ $\bar{c}\langle pkE\rangle.\bar{c}\langle pkA\rangle.$ $c(MessA).$ let $(N_a, PKA) = pdecrypt(MessA, skE)$ in $\bar{c}\langle pencrypt((N_a, PKA), PKB)\rangle.$ $c(MessB).$ $\bar{c}\langle MessB\rangle.$ $c(NC).$ let $N_b = pdecrypt(NC, skE)$ in $\bar{c}\langle pencrypt(N_b, PKB\rangle.N_b(Secret).\bar{c}\langle Secret\rangle$
A' $\triangleq c(PKX).$ $(\nu\ n_a).$ $\bar{c}\langle pencrypt((n_a, pkA), PKX)\rangle$ $c(Nonces).$ let $(N, N_b) = pdecrypt(Nonces, skA)$ in if $N = n_a$ then $\bar{c}\langle pencrypt(N_b, PKX)\rangle$	B' $\triangleq (\nu\ n_b).$ $c(PKY).$ $c(MessA).$ let $(N_a, PKZ) = pdecrypt(MessA, skB)$ in if $PKZ = PKY$ then $\bar{c}\langle pencrypt((N_a, n_b), PKY)\rangle.$ $c(NC).$ let $N = pdecrypt(NC, skB)$ in if $N = n_b$ then $\bar{n_b}\langle s\rangle$

$$\Downarrow \quad \text{I/0}$$

E' substitutes pkA, pkB for PKA, PKB. A' substitutes pkE for PKX. B' substitues pkA for PKY

$$P'_0 \triangleq (\nu\ skA)(\nu\ skB).0$$

$$\begin{aligned}\text{E'} \triangleq\ & c(MessA). \\ & \text{let } (N_a, pkA) = pdecrypt(MessA, skE) \text{ in} \\ & \bar{c}\langle pencrypt((N_a, pkA), pkB)\rangle. \\ & c(MessB). \\ & \bar{c}\langle MessB\rangle. \\ & c(NC). \\ & \text{let } N_b = pdecrypt(NC, skE) \text{ in} \\ & \bar{c}\langle pencrypt(N_b, pkB\rangle.N_b(Secret).\bar{c}\langle Secret\rangle\end{aligned}$$

$$\begin{aligned}\text{A'} \triangleq\ & (\nu\ n_a). \\ & \bar{c}\langle pencrypt((n_a, pkA), pkE)\rangle \\ & c(Nonces). \\ & \text{let } (N, N_b) = pdecrypt(Nonces, skA) \text{ in} \\ & \text{if } N = n_a \text{ then} \\ & \bar{c}\langle pencrypt(N_b, pkE)\rangle\end{aligned}$$

$$\begin{aligned}\text{B'} \triangleq\ & (\nu\ n_b). \\ & c(MessA). \\ & \text{let } (N_a, PKZ) = pdecrypt(MessA, skB) \text{ in} \\ & \text{if } PKZ = pkA \text{ then} \\ & \bar{c}\langle pencrypt((N_a, n_b), pkA)\rangle. \\ & c(NC). \\ & \text{let } N = pdecrypt(NC, skB) \text{ in} \\ & \text{if } N = n_b \text{ then } \bar{n_b}\langle s\rangle\end{aligned}$$

$$\Downarrow \quad \text{I/0}$$

E' substitutes $pencrypt((n_a, pkA), pkE)$ for $MessA$.

$$P'_0 \triangleq (\nu\ skA)(\nu\ skB).0$$

$$\begin{aligned}\text{E'} \triangleq\ & \text{let } (N_a, pkA) = \\ & \ pdecrypt(pencrypt((n_a, pkA), pkE), skE) \text{ in} \\ & \bar{c}\langle pencrypt((N_a, pkA), pkB)\rangle. \\ & c(MessB). \\ & \bar{c}\langle MessB\rangle. \\ & c(NC). \\ & \text{let } N_b = pdecrypt(NC, skE) \text{ in} \\ & \bar{c}\langle pencrypt(N_b, pkB\rangle.N_b(Secret).\bar{c}\langle Secret\rangle\end{aligned}$$

$$\begin{aligned}\text{A'} \triangleq\ & (\nu\ n_a). \\ & c(Nonces). \\ & \text{let } (N, N_b) = pdecrypt(Nonces, skA) \text{ in} \\ & \text{if } N = n_a \text{ then} \\ & \bar{c}\langle pencrypt(N_b, pkE)\rangle\end{aligned}$$

$$\begin{aligned}\text{B'} \triangleq\ & (\nu\ n_b). \\ & c(MessA). \\ & \text{let } (N_a, PKZ) = pdecrypt(MessA, skB) \text{ in} \\ & \text{if } PKZ = pkA \text{ then} \\ & \bar{c}\langle pencrypt((N_a, n_b), pkA)\rangle. \\ & c(NC). \\ & \text{let } N = pdecrypt(NC, skB) \text{ in} \\ & \text{if } N = n_b \text{ then } \bar{n_b}\langle s\rangle\end{aligned}$$

$$\Downarrow \quad \text{*let}$$

E' decrypts A's message and substitutes n_a for N_a.

$$P_0' \triangleq (\nu\ skA)(\nu\ skB).0$$

$$\begin{aligned} \text{E'} \triangleq\ & \bar{c}\langle pencrypt((n_a, pkA), pkB)\rangle.\\ & c(MessB).\\ & \bar{c}\langle MessB\rangle.\\ & c(NC).\\ & \text{let } N_b = pdecrypt(NC, skE) \text{ in}\\ & \bar{c}\langle pencrypt(N_b, pkB\rangle.N_b(Secret).\bar{c}\langle Secret\rangle \end{aligned}$$

$$\begin{aligned} \text{A'} \triangleq\ & (\nu\ n_a).\\ & c(Nonces).\\ & \text{let } (N, N_b) = pdecrypt(Nonces, skA) \text{ in}\\ & \text{if } N = n_a \text{ then}\\ & \bar{c}\langle pencrypt(N_b, pkE)\rangle \end{aligned}$$

$$\begin{aligned} \text{B'} \triangleq\ & (\nu\ n_b).\\ & c(MessA).\\ & \text{let } (N_a, PKZ) = pdecrypt(MessA, skB) \text{ in}\\ & \text{if } PKZ = pkA \text{ then}\\ & \bar{c}\langle pencrypt((N_a, n_b), pkA)\rangle.\\ & c(NC).\\ & \text{let } N = pdecrypt(NC, skB) \text{ in}\\ & \text{if } N = n_b \text{ then } \bar{n_b}\langle s\rangle \end{aligned}$$

$$\Downarrow \quad \text{I/0}$$

B' substitutes $pencrypt((n_a, pkA), pkB)$ for $MessA$.

$$P_0' \triangleq (\nu\ skA)(\nu\ skB).0$$

$$\begin{aligned} \text{E'} \triangleq\ & c(MessB).\\ & \bar{c}\langle MessB\rangle.\\ & c(NC).\\ & \text{let } N_b = pdecrypt(NC, skE) \text{ in}\\ & \bar{c}\langle pencrypt(N_b, pkB\rangle.N_b(Secret).\bar{c}\langle Secret\rangle \end{aligned}$$

$$\begin{aligned} \text{A'} \triangleq\ & (\nu\ n_a).\\ & c(Nonces).\\ & \text{let } (N, N_b) = pdecrypt(Nonces, skA) \text{ in}\\ & \text{if } N = n_a \text{ then}\\ & \bar{c}\langle pencrypt(N_b, pkE)\rangle \end{aligned}$$

$$\begin{aligned} \text{B'} \triangleq\ & (\nu\ n_b).\\ & \text{let } (N_a, PKZ) =\\ & \ pdecrypt(pencrypt((n_a, pkA), pkB), skB) \text{ in}\\ & \text{if } PKZ = pkA \text{ then}\\ & \bar{c}\langle pencrypt((N_a, n_b), pkA)\rangle.\\ & c(NC).\\ & \text{let } N = pdecrypt(NC, skB) \text{ in}\\ & \text{if } N = n_b \text{ then } \bar{n_b}\langle s\rangle \end{aligned}$$

$$\Downarrow \quad \text{*let}$$

B' decrypts the message and substitutes n_a, pkA for N_a, PKZ.

$$\begin{array}{lll|lll}
P_0' & \triangleq & (\nu\ skA)(\nu\ skB).0 & \text{E'} & \triangleq & c(MessB). \\
 & & & & & \bar{c}\langle MessB\rangle. \\
 & & & & & c(NC). \\
 & & & & & \text{let } N_b = pdecrypt(NC, skE) \text{ in} \\
 & & & & & \bar{c}\langle pencrypt(N_b, pkB\rangle.N_b(Secret).\bar{c}\langle Secret\rangle \\
\hline
\text{A'} & \triangleq & (\nu\ n_a). & \text{B'} & \triangleq & (\nu\ n_b). \\
 & & c(Nonces). & & & \text{if } pkA = pkA \text{ then} \\
 & & \text{let } (N, N_b) = pdecrypt(Nonces, skA) \text{ in} & & & \bar{c}\langle pencrypt((n_a, n_b), pkA)\rangle. \\
 & & \text{if } N = n_a \text{ then} & & & c(NC). \\
 & & \bar{c}\langle pencrypt(N_b, pkE)\rangle & & & \text{let } N = pdecrypt(NC, skB) \text{ in} \\
 & & & & & \text{if } N = n_b \text{ then } \bar{n}_b\langle s\rangle
\end{array}$$

$$\Downarrow \quad \text{I/0}$$

E' substitutes $pencrypt((n_a, n_b), pkA)$ for $MessB$.

$$\begin{array}{lll|lll}
P_0' & \triangleq & (\nu\ skA)(\nu\ skB).0 & \text{E'} & \triangleq & \bar{c}\langle pencrypt((n_a, n_b), pkA)\rangle. \\
 & & & & & c(NC). \\
 & & & & & \text{let } N_b = pdecrypt(NC, skE) \text{ in} \\
 & & & & & \bar{c}\langle pencrypt(N_b, pkB\rangle.N_b(Secret).\bar{c}\langle Secret\rangle \\
\hline
\text{A'} & \triangleq & (\nu\ n_a). & \text{B'} & \triangleq & (\nu\ n_b). \\
 & & c(Nonces). & & & c(NC). \\
 & & \text{let } (N, N_b) = pdecrypt(Nonces, skA) \text{ in} & & & \text{let } N = pdecrypt(NC, skB) \text{ in} \\
 & & \text{if } N = n_a \text{ then} & & & \text{if } N = n_b \text{ then } \bar{n}_b\langle s\rangle \\
 & & \bar{c}\langle pencrypt(N_b, pkE)\rangle & & &
\end{array}$$

$$\Downarrow \quad \text{I/0}$$

A' substitutes $pencrypt((n_a, n_b), pkA)$ for $Nonces$.

$$\begin{array}{lll|lll}
P_0' & \triangleq & (\nu\ skA)(\nu\ skB).0 & \text{E'} & \triangleq & c(NC). \\
 & & & & & \text{let } N_b = pdecrypt(NC, skE) \text{ in} \\
 & & & & & \bar{c}\langle pencrypt(N_b, pkB\rangle.N_b(Secret).\bar{c}\langle Secret\rangle \\
\hline
\text{A'} & \triangleq & (\nu\ n_a). & \text{B'} & \triangleq & (\nu\ n_b). \\
 & & \text{let } (N, N_b) = & & & c(NC). \\
 & & \ pdecrypt(pencrypt((n_a, n_b), pkA), skA) \text{ in} & & & \text{let } N = pdecrypt(NC, skB) \text{ in} \\
 & & \text{if } N = n_a \text{ then} & & & \text{if } N = n_b \text{ then } \bar{n}_b\langle s\rangle \\
 & & \bar{c}\langle pencrypt(N_b, pkE)\rangle & & &
\end{array}$$

$$\Downarrow \quad \text{*let}$$

A' decrypts the message and substitutes n_a, n_b for N, N_b.

$P_0' \triangleq (\nu\ skA)(\nu\ skB).0$	E' $\triangleq$ $c(NC)$. let $N_b = pdecrypt(NC, skE)$ in $\bar{c}\langle pencrypt(N_b, pkB)\rangle.N_b(Secret).\bar{c}\langle Secret\rangle$
A' $\triangleq$ $(\nu\ n_a)$. if $n_a = n_a$ then $\bar{c}\langle pencrypt(n_b, pkE)\rangle$	B' $\triangleq$ $(\nu\ n_b)$. $c(NC)$. let $N = pdecrypt(NC, skB)$ in if $N = n_b$ then $\bar{n_b}\langle s\rangle$

$$\Downarrow \quad \text{I/0}$$

E' substitutes $pencrypt(n_b, pkE)$ for NC.

$P_0' \triangleq (\nu\ skA)(\nu\ skB).0$	E' $\triangleq$ let $N_b =$ $pdecrypt(pencrypt(n_b, pkE), skE)$ in $\bar{c}\langle pencrypt(N_b, pkB)\rangle.N_b(Secret).\bar{c}\langle Secret\rangle$
A' $\triangleq$ $(\nu\ n_a).0$	B' $\triangleq$ $(\nu\ n_b)$. $c(NC)$. let $N = pdecrypt(NC, skB)$ in if $N = n_b$ then $\bar{n_b}\langle s\rangle$

$$\Downarrow \quad \text{*let}$$

E' decrypts the message and substitutes n_b for N.

$P_0' \triangleq (\nu\ skA)(\nu\ skB).0$	E' $\triangleq$ $\bar{c}\langle pencrypt(n_b, pkB)\rangle.n_b(Secret).\bar{c}\langle Secret\rangle$
A' $\triangleq$ $(\nu\ n_a).0$	B' $\triangleq$ $(\nu\ n_b)$. $c(NC)$. let $N = pdecrypt(NC, skB)$ in if $N = n_b$ then $\bar{n_b}\langle s\rangle$

$$\Downarrow \quad \text{I/0}$$

B' substitutes $pencrypt(n_b, pkB)$ for NC.

$P_0' \triangleq (\nu\ skA)(\nu\ skB).0$	E' $\triangleq$ $n_b(Secret).\bar{c}\langle Secret\rangle$
A' $\triangleq$ $(\nu\ n_a).0$	B' $\triangleq$ $(\nu\ n_b)$. let $N =$ $pdecrypt(pencrypt(n_b, pkB), skB)$ in if $N = n_b$ then $\bar{n_b}\langle s\rangle$

$$\Downarrow \quad \text{*let}$$

B' decrypts the message and substitutes n_b for N.

P_0' $\triangleq$ $(\nu\ skA)(\nu\ skB).0$	E' $\triangleq$ $n_b(Secret).\bar{c}\langle Secret\rangle$
A' $\triangleq$ $(\nu\ n_a).0$	B' $\triangleq$ $(\nu\ n_b)$. if $n_b = n_b$ then $\bar{n_b}\langle s\rangle$

$\Downarrow$ I/0

E' substitutes s for *Secret*.

P_0' $\triangleq$ $(\nu\ skA)(\nu\ skB).0$	E' $\triangleq$ $\bar{c}\langle s\rangle$
A' $\triangleq$ $(\nu\ n_a).0$	B' $\triangleq$ $(\nu\ n_b).0$

Here we have reached a state $\bar{c}\langle s\rangle$, thus we have seen how the πCalculus protocol presented above can be reduced to an insecure state as was defined earlier. Clearly it can be quite tedious to trace even a simple protocol such as Needham-Schroeder, which was the motivation for Abadi and Blanchet to create their translation into Prolog [**3**]. An automated process which can determine the security of such a protocol would surely be useful, so in the next chapter we describe the steps of their translation and show how it works for our Needham-Schroeder protocol.

CHAPTER 3

Translating Protocols into Logic Programs

We saw in the last chapter how to trace a πCalculus protocol to determine the states of its execution. Now we present Abadi and Blanchet's translation of protocols to logic programs. This translation is essentially a protocol-checker, it can be used to prove secrecy properties. Given a process P and a set of names S the translation builds a sequence of Horn clauses based on the following.

There are two predicates: *attacker* and *message*. Facts take the form of either $attacker(p)$ meaning the attacker knows p, or $message(c, p)$ meaning the message p was sent on channel c. The contents of a message are called patterns, which can be either variables(x, y, z), names$(a[], b[], c[])$, or constructor applications$(f(p_1, ...p_n))$. Names that are created by the ν binding are treated as a function of the inputs that came before them in order to avoid collision. For example, in $c(x).(\nu\ a).P$ we write the pattern $a[x]$ to denote that a is a fresh name.

1. Rules for Attacker

The attacker initially has all the names in the set S, so the translation has the rules $attacker(a[])$ for each $a \in S$. We also include rules based on the attacker model developed by Dolev and Yao [**6**]. In this model, the attacker has the following abilities:

- Copy, destroy, delay, or reorder any set of messages.
- Split messages into their pieces.
- Read any message for which he has the decryption key.
- Create messages from parts of any he has observed and address it to any principal as being from any other principal.
- Appear as a legitimate principal to any other principal.

- Be simultaneously involved in as many instances of a protocol as he desires.
- Generate as many messages as he wants.
- Remember messages from previous instances of protocols.

To encapsulate these abilities in Prolog, the translation of any protocol includes the following statements:

```
(Rf)    For each constructor f of arity n,
          attacker(f(X1,...,Xn)) :- attacker(X1),...,attacker(Xn).
(Rg)    For each destructor g,
          for each equation g(M1,...,Mn) in def(g),
        attacker(M) :- attacker(M1),...,attacker(Mn).
(Rl)    attacker(Y) :- message(X, Y), attacker(X).
(Rs)    message(X, Y) :- attacker(X), attacker(Y).
```

Rules (Rf) and (Rg) mean the attacker can apply all operations to all terms he has. Rule (Rl) means the attacker can receive messages on any channel he has, and rule (Rs) means the attacker can construct messages from any terms he has on any channel he has.

2. Rules for Protocol

Used in the translation are a partial function ρ which maps names and variables to patterns, and a sequence of facts h of the form $message(p, p')$. Now we can finally describe the translation inductively on the structure of the πCalculus. The translation $[\![P]\!]_{\rho h}$ of a process P is a set of Prolog statements built by the rules shown in Figure 1, but informally the rules are as follows:

- The translation of 0 is the empty set since nothing happens.
- The translation of a parallel composition is the union of their translations.
- The translation of a replication is the same as that of the protocol being replicated.

$$\begin{aligned}
[\![0]\!]_{\rho h} &= \emptyset \\
[\![P \mid Q]\!]_{\rho h} &= [\![P]\!]_{\rho h} \cup [\![Q]\!]_{\rho h} \\
[\![!P]\!]_{\rho h} &= [\![P]\!]_{\rho h} \\
[\![(\nu\ a)P]\!]_{\rho h} &= [\![P]\!]_{\rho[a \mapsto a[p'_1,\ldots,p'_n]]h} \\
&\quad \text{if } h = message(p_1, p'_1) \wedge \ldots \wedge \\
&\quad message(p_n, p'_n) \\
[\![c(X).P]\!]_{\rho h} &= [\![P]\!]_{\rho[X \mapsto X]h \wedge message(\rho(c),X)} \\
[\![\bar{c}\langle m\rangle.P]\!]_{\rho h} &= [\![P]\!]_{\rho h} \cup \{message(\rho(c), \rho(m)) :- h.\} \\
[\![let\ x = g(m_1, \ldots, m_n)\ in\ P\ else\ Q]\!]_{\rho h} &= \cup\{[\![P]\!]_{((\sigma\rho)[x \mapsto \sigma' p'])(\sigma h)}\} \cup [\![Q]\!]_{\rho h} \\
&\quad \text{where } g(p'_1, \ldots, p'_n) = p' \text{ and } (\sigma, \sigma') \text{ is} \\
&\quad \text{a most general pair of substitutions such} \\
&\quad \text{that } \sigma\rho(m_1) = \sigma' p'_1, \ldots, \sigma\rho(m_n) = \sigma' p'_n
\end{aligned}$$

FIGURE 1. Rules for the Translation of a Protocol

- Binding a variable a with $(\nu\ a)$ replaces the name a with pattern $a[\ldots]$ based on the messages previously received.
- For an input clause, the sequence h is extended by that input.
- An output statement adds a clause to the final translation which is dependent upon the inputs previously processed.
- The translation of a destructor unions all cases where the destructor succeeds, applying substitutions to the patterns in question, along with the case where the destructor fails. Thus the translation avoids determining whether the destructor succeeds or fails.

3. Summary

Let P_0 be a πCalculus protocol, and let $\rho = \{a \mapsto a[] | a \in fn(P_0)\}$. Then the translation $B_{P_0,S}$ of P_0 into Prolog is defined to be:

$$B_{P_0,S} = [\![P]\!]_{\rho\emptyset} \cup \{attacker(a[]) | a \in S\} \cup \{(Rf), (Rg), (Rl), (Rs)\}$$

Given this translation, we have the following secrecy result. If $s \in fn(P_0)$ and the goal $attacker(s)$ cannot be proved from $B_{P_0,S}$, then P_0 preserves the secrecy of s in S. This holds even though whether a fact can be derived from $B_{P_0,S}$ may be undecidable. Abadi and Blanchet prove this result using a type system. They describe a type *Public* and show that s cannot be in the set of patterns with type *Public*. In the next chapter, we prove this result directly by induction on the structure of the πCalculus. But first, let's look at an example.

4. Needham-Schroeder Translated

Figure 2 shows our NS2 protocol from chapter 2 translated into Prolog using the method we have just defined. Since the protocol is relatively short, avoiding collision is simple and we write patterns such as $c[]$ as just c.

```
attacker(c).

attacker(skE).

attacker(Mess) :- attacker(pencrypt(Mess, pk(Key))), attacker(Key).

attacker(Mess) :- message(Chan, Mess), attacker(Chan).

attacker(pencrypt(Mess, Key)) :- attacker(Mess), attacker(Key).

attacker(pk(Key)) :- attacker(Key).

message(c, pk(skA)).

message(c, pk(skB)).

message(Chan, Mess) :- attacker(Chan), attacker(Mess).

message(c, pencrypt(nA, PKX)) :- message(c, PKX).

message(c, pencrypt(pk(skA), PKX)) :- message(c, PKX).

message(c, pencrypt(Nb, PKX)) :-

    message(c, PKX),

    message(c, pencrypt(nA, pk(skA))),

    message(c, pencrypt(Nb, pk(skA))).

message(c, pencrypt(Na, PKX)) :-

    message(c, PKX),

    message(c, pencrypt(Na, pk(skB))),

    message(c, pencrypt(PKY, pk(skB))),

    PKX = PKY.

message(c, pencrypt(nB, PKX)) :-

    message(c, PKX),

    message(c, pencrypt(Na, pk(skB))),

    message(c, pencrypt(PKY, pk(skB))),

    PKX = PKY.

message(nB, s) :-

    message(c, PKX),

    message(c, pencrypt(Na, pk(skB))),

    message(c, pencrypt(PKY, pk(skB))),

    PKX = PKY,

    message(c, pencrypt(nB, pk(skB))).
```

FIGURE 2. Translation of NS2 into Prolog

CHAPTER 4

Proof of Secrecy

In this chapter we prove the secrecy result of Abadi and Blanchet's translation directly. Informally, if the translation cannot prove $attacker(s)$ then the protocol will not reduce to the insecure state described in chapter 2.

LEMMA 1. *Given a set of horn-clauses Γ, a goal G, and names b, c not in G; we have $\Gamma \nvdash G \rightarrow \Gamma\{c/b\} \nvdash G$. Proof in appendix A.*

LEMMA 2. *If $c(x).R \mid W \rightarrow^* bad$ then this sequence of reductions must be of the form $c(x).R \mid W \rightarrow^* c(x).R \mid \bar{c}\langle y\rangle W' \rightarrow R\{y/x\} \mid W' \rightarrow^* bad$. In a similar manner, if $\bar{c}\langle x\rangle.R \mid W \rightarrow^* bad$ then this sequence must be of the form $\bar{c}\langle x\rangle.R \mid W \rightarrow^* \bar{c}\langle x\rangle.R \mid c(y).W' \rightarrow R \mid W'\{x/y\} \rightarrow^* bad$. Proof in appendix A.*

LEMMA 3. *Let protocol P preserve the secrecy of s in S. Let T be the union of names in S and the closure of constructors applied to those names. Then $P\{y_1/x_1, y_2/x_2, ..., y_n/x_n\}$ where $y_1, ..., y_n \in T$ preserves the secrecy of s in S. We write $P\{y/x\}$ unless it is necessary to show multiple substitutions. Proof in appendix A.*

THEOREM (SECRECY). *Given a protocol P_0, its translation into a logic program $B_{P_0,S}$, and a secret s; we have that if $B_{P_0,S} \nvdash attacker(s)$ then P_0 preserves the secrecy of s in S.*

PROOF. By induction on P_0, we show for all cases that if $B_{P_0,S} \nvdash attacker(s)$ then P_0 is secure. We also assume that the partial function $\rho : S \rightarrow S$ and the conjunction of atoms h built by the translation do not affect the secrecy of a protocol, i.e. if a protocol is secure it is secure for any ρ and any h. For the base case, clearly the empty protocol preserves the secrecy of s in S. Assume that the theorem holds for any subterm of P_0

(Parallel Composition) Suppose $P_0 = R \mid Q$. Since $B_{P_0,S} = B_{R,S} \cup B_{Q,S}$ we can assume that $B_{R,S} \nvdash \mathit{attacker}(s)$ and $B_{Q,S} \nvdash \mathit{attacker}(s)$. So by the induction hypothesis, R and Q preserve the secrecy of s in S. The definition of secrecy states there is no W such that $R \mid W \to^* \mathit{bad}$ or $Q \mid W \to^* \mathit{bad}$. Now suppose that there exists W such that $(R \mid Q) \mid W \to^* \mathit{bad}$. By the associativity of parallel composition we can rewrite this as $R \mid (Q \mid W) \to^* \mathit{bad}$, which contradicts the fact that R preserves secrecy. Therefore no such W exists and we can conclude that P_0 preserves secrecy.

(Replication) Suppose $P_0 = !R$. Since $B_{P_0,S} = B_{R,S}$ and $!R \equiv R \mid !R$, it follows by the same argument as above that P_0 also preserves secrecy.

(Binding) Suppose $P_0 = (\nu a)R$. We have that $B_{P_0,S} = B_{R,S}$ with the fresh name a added to ρ. Since a does not appear in the term $\mathit{attacker}(s)$, Lemma 1 gives us that $B_{R,S} \nvdash \mathit{attacker}(s)$. So by the induction hypothesis we can assume that R preserves secrecy. Now suppose that there is a W such that $(\nu a)R \mid W \to^* \mathit{bad}$. Since we may assume that a does not occur in W, this can be rewritten $(\nu a)(R \mid W)$. From here we have $(\nu a)(R \mid W) \to^* \mathit{bad}'$ implies $R \mid W \to^* \mathit{bad}$, where $\mathit{bad} = (\nu a)\mathit{bad}'$. This contradicts our assumption, so no such W exists and P_0 preserves secrecy.

(Input) Suppose $P_0 = c(x).R$. Let the set $\bar{A}$ be the axioms included in the translation of P_0. Then we have $\bar{A} \cup [\![c(x).R]\!]_{\rho h} \nvdash \mathit{attacker}(s)$. By the rules of the translation this becomes $\bar{A} \cup [\![R]\!]_{\rho' h'} \nvdash \mathit{attacker}(s)$, where $\rho' = \rho[x \to x]$ and $h' = h \wedge \mathit{mess}(\rho(c), \rho(x))$. Since an attacker only has names in S, x can be either a name or a constructor applied to a name. Thus since x cannot occur in the goal $\mathit{attacker}(s)$, we have by Lemma 1 that $B_{R,S} \nvdash \mathit{attacker}(s)$. Thus R preserves secrecy by the induction hypothesis, and we must show that $c(x).R$ also preserves secrecy. For contradiction, suppose it doesn't preserve secrecy. Then there is a W such that $c(x).R \mid W \to^* \mathit{bad}$. Therefore by Lemma 2 this sequence of reactions is of the form $c(x).R \mid W \to^* c(x).R \mid \bar{c}\langle y\rangle W' \to R\{y/x\} \mid W \to^* \mathit{bad}$. So $c(x).R \mid W \to^* \mathit{bad}$

rewrites to $c(x).R \mid \bar{c}\langle y\rangle.W' \rightarrow^* \mathit{bad}$, where y must be in S because these are the only names an attacker has. Thus by the I/O reduction rule of the π-calculus we have $c(x).R \mid \bar{c}\langle y\rangle.W' \rightarrow R\{y/x\} \mid W'$. Since we already showed that R preserves secrecy, Lemma 2 tells us that $R\{y/x\}$ also preserves secrecy so we have a contradiction. Therefore P_0 preserves secrecy.

(Output) Suppose $P_0 = \bar{c}\langle M\rangle.R$ where M is a term. We have $\bar{A} \cup [\![\bar{c}\langle M\rangle.R]\!]_{\rho h} \not\vdash \mathit{attacker}(s)$. The translation rule for output gives $\bar{A} \cup [\![R]\!]_{\rho h} \cup h \rightarrow \mathit{message}(\rho(c), \rho(M))$. Here we know that if the union of several Horn-clauses does not prove $\mathit{attacker}(s)$, then neither does any subset of the union. Thus $B_{R,S} \not\vdash \mathit{attacker}(s)$ so by induction R preserves the secrecy of s from S. Now let's assume that there exists W such that $\bar{c}\langle x\rangle.R \mid W \rightarrow^* \mathit{bad}$, which Lemma 2 states is of the form $\bar{c}\langle x\rangle.R \mid W \rightarrow^* \bar{c}\langle x\rangle.R \mid c(y).W' \rightarrow R \mid W\{y/x\} \rightarrow^* \mathit{bad}$. If W does not rewrite to an input term, then the outermost term of P_0 will never be processed and bad cannot be reached. But the induction hypothesis states that $R \mid W\{y/x\}$ cannot reduce to bad since R preserves secrecy. Therefore we have a contradiction and P_0 must also preserve secrecy.

(Destruction) Suppose $P_0 = \text{let } x = g(M_1, ..., M_n) \text{ in } R \text{ else } Q$. Thus we have $\bar{A} \cup [\![\text{let } x = g(M_1, ..., M_n) \text{ in } R \text{ else } Q]\!]_{\rho h} \not\vdash \mathit{attacker}(s)$. The translation rule for this term requires substitutions on ρ and h, but does not add anything to them or the translation. Thus we can write it as $\bar{A} \cup [\![R]\!]_{\rho' h'} \cup [\![Q]\!]_{\rho h}$. This whole union does not prove $\mathit{attacker}(s)$ so neither do its parts and thus we have that R, Q preserves secrecy by the induction hypothesis. Now suppose that P_0 does not preserve secrecy, then there exists W such that $P_0 \mid W \rightarrow^* \mathit{bad}$. To reduce this protocol by the rules of the π-calculus we choose either $R\{g(\vec{M})/x\} \mid W$ or $Q \mid W$, so we must show that both produce a contradiction. For the first case $g(\vec{M})$ comes from S, so Lemma 2 tells us that $R\{g(\vec{M})/x\}$ preserves secrecy. Thus such a W cannot exist and we have a contradiction. For the second case we immediately know this cannot reduce to bad so we again have a contradiction. Therefore P_0 must also preserve the secrecy of s.

We have covered all forms that P_0 can take, therefore we can conclude that in all cases we have $B_{P_0,S} \nvdash attacker(s) \Longrightarrow P_0$ preserves the secrecy of s in S. $\square$

CHAPTER 5

Conclusion

In this thesis we have seen how security protocols can be written in the πCalculus, thus giving a more thorough treatment of the communications and computations that take place. We have also seen two ways to show a security flaw in a protocol: by manually tracing through the reductions of the calculus, or by using Abadi and Blanchet's translation to create a logic-program which can be queried with secrecy properties. Though the translation cannot definitively say that a protocol is secure(a proof of the goal $attacker(s)$ may not terminate), the contrapositive of our proven theorem states that if a flaw exists the translation will find it.

This relationship between process calculi and logic-programs provides an efficient way to study security protocols. With so much important information being communicated over the internet, further analysis of these protocols will hopefully lead to safer and more efficient communication systems.

APPENDIX A

LEMMA 1. *Given a set of Horn-clauses* Γ*, a goal* G*, and names* b, c *not in* G*; we have* $\Gamma \nvdash G \rightarrow \Gamma\{c/b\} \nvdash G$.

PROOF. Suppose $\Gamma \nvdash G$ and $\Gamma\{c/b\} \vdash G$. (*)
Then we have $\vdash \Gamma\{c/b\} \rightarrow G$. Now for a variable y not in G we have $\vdash \forall y(\Gamma\{y/b\} \rightarrow G)$. Thus since b is not in G we have $\vdash \forall y(\Gamma\{y/b\}\{b/y\} \rightarrow G)$. This simplifies to $\vdash \Gamma \rightarrow G$, which contradicts (*). □

LEMMA 2. *If* $c(x).R \mid W \rightarrow^* bad$ *then this sequence of reductions must be of the form* $c(x).R \mid W \rightarrow^* c(x).R \mid \bar{c}\langle y\rangle W' \rightarrow R\{y/x\} \mid W' \rightarrow^* bad$*. In a similar manner, if* $\bar{c}\langle x\rangle.R \mid W \rightarrow^* bad$ *then this sequence must be of the form* $\bar{c}\langle x\rangle.R \mid W \rightarrow^* \bar{c}\langle x\rangle.R \mid c(y).W' \rightarrow R \mid W'\{x/y\} \rightarrow^* bad$.

PROOF. We handle the first statement by induction on the number of reductions, n, required to reach *bad*. Suppose $n = 1$, then W must be $\bar{c}\langle y\rangle.W'$ and $(x).R \mid \bar{c}\langle y\rangle.W' \rightarrow R\{y/x\} \mid W'$ where $R\{y/x\}$ is of the form $\bar{c}\langle s\rangle.R'$. Now suppose $c(x).R \mid W \rightarrow N \rightarrow^n \bar{c}\langle s\rangle.R \mid W'$. Here either N is $c(x).R \mid W'$ where $W \rightarrow W'$, in which case the induction hypothesis gives us that $N \rightarrow^n bad$ is of the stated form; or W is of the form $\bar{c}\langle y\rangle.W'$, in which case $c(x).R \mid \bar{c}\langle y\rangle.W' \rightarrow R\{y/x\} \mid W' \rightarrow^n bad$ and the conclusion is satisfied in one step.

A similar proof shows the second statement is also true. □

LEMMA 3. *Let protocol* P *preserve the secrecy of* s *in* S*. Let* T *be the union of names in* S *and the closure of constructors applied to those names. Then* $P\{y_1/x_1, y_2/x_2, ..., y_n/x_n\}$ *where* $y_1, ..., y_n \in T$ *preserves the secrecy of* s *in* S*. We write* $P\{y/x\}$ *unless it is necessary to show multiple substitutions.*

Proof. By induction on P, assume for all cases that P is secure: there is no W such that $P \mid W \rightarrow^* \mathit{bad}$. We write $P\{y/x\}$ unless it is necessary to show multiple substitutions and show that there is no W' where $P\{y/x\} \mid W' \rightarrow^* \mathit{bad}$. The base case clearly holds, so for induction assume that the lemma holds for any subterm of P.

(Parallel Composition) Suppose $P = R \mid Q$ where the lemma holds for R, Q, and suppose for a contradiction that there exists W such that $(R \mid Q)\{y/x\} \mid W \rightarrow^* \mathit{bad}$. Substitution distrubutes so we have $(R\{y/x\} \mid Q\{y/x\}) \mid W \rightarrow^* \mathit{bad}$. If we rewrite this as $R\{y/x\} \mid (Q\{y/x\}) \mid W) \rightarrow^* \mathit{bad}$ then we have a contradiction since $R\{y/x\}$ is known to be secure. Thus no such W exists and $P\{y/x\}$ is also secure.

(Replication) Suppose $P =!R$ where R preserves secrecy. Since $!R \equiv R \mid !R$ the argument is the same as above.

(Binding) Suppose $P = (\nu a)R$ where the lemma holds for R. Suppose for contradiction that $(\nu a)R\{y/x\} \mid W \rightarrow^* \mathit{bad}$. If a is chosen not in W then this can be rewritten $(\nu a)(R\{y/x\} \mid W) \rightarrow^* \mathit{bad}$. For this to be hold, it must be true that $R\{y/x\} \mid W \rightarrow^* \mathit{bad}'$ where $\mathit{bad} = (\nu a)\mathit{bad}'$. This is a contradiction of the lemma holding for R, thus $P\{y/x\}$ preserves secrecy.

(Input) Suppose $P = c(m).R$ and suppose that $c(m).R\{y/x\} \mid W \rightarrow^* \mathit{bad}$. Here W must rewrite to an output term, otherwise the input before R is never reduced and bad cannot be reached. So we have $c(m).R\{y/x\} \mid \bar{c}\langle n\rangle.W' \rightarrow^* \mathit{bad}$. By the I/O rule of the π-calculus this reduces to $R\{y/x, n/m\} \mid W' \rightarrow^* \mathit{bad}$. Here we have a contradiction since the lemma holds for R, so $P\{y/x\}$ is also secure.

(Output) Suppose $P = \bar{c}\langle n\rangle.R$ and suppose that $\bar{c}\langle n\rangle.R\{y/x\} \mid W \rightarrow^* \mathit{bad}$. W must rewrite to an input term, otherwise the output before R cannot be reached and the protocol cannot reduce to bad. Thus we write $\bar{c}\langle n\rangle.R\{y/x\} \mid c(m).W'$, which reduces to $R\{y/x\} \mid W\{n/m\} \rightarrow^* bad$. This is a contradiction of the induction hypothesis, so $P\{y/x\}$ is secure.

(Destruction) Suppose $P =$ let $x = g(M_1, ...M_n)$ in R else Q. Suppose for contradiction that (let $n = g(M_1, ...M_n)$ in R else $Q)\{y/x\} \mid W \rightarrow^* bad$. This reduces to $R\{g(\vec{M})/n, x/y\} + Q\{y/x\} \mid W \rightarrow^* bad$. Here the protocol 'chooses' whether to progress in R or in Q, but note that both are subterms of P so the above term produces a contradiction regardless of the choice. Thus P is also secure.
We have covered all forms P could take, therefore the lemma holds for any secure protocol. $\square$

Bibliography

[1] M. Abadi and C. Fournet. Mobile values, new names, and secure communication. *Proceedings of the 28th Annual ACM Symposium on Principles of Programming Languages*, pages 104–115, 2001.

[2] M. Abadi and A.D. Gordon. A calculus for cryptographic protocols: the spi-calculus. *Inf. Comput.*, 148:1–70, Jan 1999.

[3] Martin Abadi and Bruno Blanchet. Analyzing security protocols with secrecy types and logic programs. *29th Annual ACM Symposium on Principles of Programming Languages(POPL 2002)*, pages 33–44, Jan 2002.

[4] R.M. Amadio and D. Lugiez. On the reachability problem in cryptographic protocols. *CONCUR 2000: Concurrency Theory*, 2000.

[5] M. Burroughs, M. Abadi, and R. Needham. A logic of authentication. *Proceedings of the Royal Society of London*, A(426):233–271, 1989.

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

[7] Gavin Lowe. An attack on the needham-schroeder public key authentication protocol. *Information Processing Letters*, 56(3):131–136, Nov 1995.

[8] Robin Milner. *Communicating and Mobile Systems: The Pi-Calculus*. Cambridge University Press, 1999.