Hiding Names: Private Authentication in the Applied Pi Calculus

Cédric Fournet[1] and Martín Abadi[2]

[1] Microsoft Research
[2] University of California at Santa Cruz

Abstract. We present the analysis of a protocol for private authentication in the applied pi calculus. We treat authenticity and secrecy properties of the protocol. Although such properties are fairly standard, their formulation in the applied pi calculus makes an original use of process equivalences. In addition, we treat identity-protection properties, which are a delicate concern in several recent protocol designs.

1 Introduction

In recent years, the understanding of basic security properties such as integrity and confidentiality has become both deeper and wider. There has also been substantial progress in the design and verification of protocols that aim to guarantee these properties. On the other hand, fundamental tasks such as secure session establishment remain the subject of active, productive research. Moreover, properties beyond integrity and confidentiality have been studied rather lightly to date. These properties include, for example, protection of identity information and protection against denial-of-service attacks. They may seem secondary but they are sometimes important.

This paper contributes to the ongoing study of security protocols and of their properties. More specifically, this paper presents the analysis of a security protocol in the applied pi calculus [2], a recent variant of the pi calculus. The protocol in question is one for private authentication (the second protocol of [1]). Its analysis is worthwhile for several reasons:

- The protocol is for a standard purpose, namely establishing a session (with associated cryptographic keys), and it is concerned with standard security properties, such as authenticity and secrecy. Therefore, the analysis of the protocol exemplifies concepts and techniques relevant to many other protocols.
- In addition, the protocol is concerned with a privacy property: it aims to guarantee that third parties do not learn the identity of protocol participants. Although this property and similar ones appear prominently in several recent protocol designs, they have hardly been specified and proved precisely to date. Therefore, this paper develops an approach for stating and deriving those properties.

To appear in the proceedings of the International Symposium on Software Security (ISSS'02), Tokyo, Japan, November 8-10, 2002, LNCS. © Springer-Verlag Berlin Heidelberg 2002

– The protocol includes some delicate features, and is not a trivial example invented only in order to illustrate formal techniques. On the other hand, the protocol remains fairly simple, so we can give relatively concise treatments of its main properties.

In the applied pi calculus, the constructs of the classic pi calculus can be used to represent concurrent systems that communicate on channels, and function symbols can be used to represent cryptographic operations and other operations on data. Large classes of important attacks can also be expressed in the applied pi calculus, as contexts. These include the typical attacks for which a symbolic, mostly "black-box" view of cryptography suffices (but not for example some lower-level attacks that depend on timing behavior or on probabilities). Thus, in general, the applied pi calculus serves for describing and reasoning about many of the central aspects of security protocols. In particular, it is an appropriate setting for the analysis of the protocol for private authentication. Some of the properties of the protocol can be nicely captured in the form of equivalences between processes. Moreover, some of the properties are sensitive to the equations satisfied by the cryptographic functions upon which the protocol relies. The applied pi calculus is well-suited for expressing those equivalences and those equations.

In a sense, private authentication is about hiding the names (or identities) of protocol participants. The applied pi calculus permits hiding the names that represent private communication channels and secret cryptographic keys (through the restriction construct ν). Despite this superficial coincidence, the name hiding of private authentication and that of the applied pi calculus are rather different. We do not have a direct reduction of one to the other. However, the name hiding of the applied pi calculus is crucial for expressing the protocol under consideration and for deriving the equivalences that express its properties.

The next two sections explain private authentication and the applied pi calculus, respectively. Section 4 shows how to express a protocol for private authentication in the applied pi calculus. Section 5 treats the authenticity and secrecy properties of the protocol; section 6, its privacy properties. (We omit all proofs, because of space constraints.) Section 7 discusses some related work and concludes.

2 Private Authentication

Although we do not aim to provide a general definition of privacy (partly because one might have to be too vague or empty), we focus on the following frequent scenario in which privacy is a central concern: two or more mobile interlocutors wish to communicate securely, protecting their messages and also their identities from third parties. This scenario arises often in mobile telephony and mobile computing [7, 14, 12, 15, 6, 8]. In these contexts, roaming users may want to conceal their identities from others and even from infrastructure providers and operators. Furthermore, identity protection is a goal of several recent protocols for communication at the IP level [9, 5].

More specifically, suppose that a mobile principal A (a user or a computer) wishes to communicate with some other principals, and that A is willing to prove its identity to these principals. Suppose that B is one of them, and that B is willing to communicate with A and to prove its identity to A. After providing these proofs, in the subsequent session, A and B may make sensitive requests from each other and may reveal sensitive data to each other. We study a protocol (from [1]) that enables A and B to establish an authenticated communication channel. By following the protocol, A and B should not have to indicate their identity and presence to any third parties.

In this section, we review the protocol informally. We start by outlining its assumptions, then describe its message flow and (briefly) some of its properties and limitations. Later sections contain a formal development of these points.

2.1 Assumptions

The protocol assumes that messages do not automatically reveal the identity of their senders and receivers—for example, by mentioning them in headers. This assumption entails some difficulties in routing messages. Focusing on a relatively simple but important case, the protocol supposes that all messages are broadcast within some location, such as a physical building or a virtual chat room.

As in most security protocols (following Needham and Schroeder [13]), the communication infrastructure is untrusted. An attacker can interpose itself on all public communication channels, and thus can alter or copy parts of messages, delete messages, replay messages, or emit false material.

The protocol also assumes that each principal A has a public key K_A and a corresponding private key K_A^{-1} (e.g., [11]), and that the association between principals and public keys is known. This association can be implemented with the help of a mostly-off-line certification authority, and it is trivial when one identifies public keys with principal names. Public keys are used for encryption and private keys for the corresponding decryptions. Informally, when K is a public key, we write $\{M\}_K$ for the encryption of M using K. The protocol assumes some properties of the encryption scheme (not all entirely standard). Only a principal that knows the corresponding private key K^{-1} should be able to understand a message encrypted under a public key K. Furthermore, decrypting a message with a private key K^{-1} should succeed only if the message was encrypted under the corresponding public key K, and the success or failure of a decryption should be obvious to the principal who performs it. Finally, someone who sees a message encrypted under a public key K should not be able to tell that it is under K without knowledge of the corresponding private key K^{-1}, even with knowledge of K or other messages under K.

2.2 The Protocol

When a principal A wishes to talk to another principal B, and B is willing to talk to a set of principals S_B, the protocol specifies that A and B proceed as follows:

- A generates a fresh, unpredictable quantity N_A (a "nonce"), and sends out

$$\text{"hello"}, \{\text{"hello"}, N_A, K_A\}_{K_B}$$

- When B receives any message that consists of "hello" and (apparently) a ciphertext, B tries to decrypt the second component using K_B^{-1}. If the decryption succeeds, then B extracts the corresponding nonce N_A and key K_A, checks that $A \in S_B$, generates a nonce N_B, and sends out

$$\text{"ack"}, \{\text{"ack"}, N_A, N_B, K_B\}_{K_A}$$

  If the decryption fails, if the plaintext is not of the required form, or if $A \notin S_B$, then B instead sends out a "decoy" message. This message should basically look like B's other message. In particular, it may have the form

$$\text{"ack"}, \{N\}_K$$

  where N is a fresh nonce and only B knows K^{-1}, or it may be indistinguishable from a message of this form.
- When A receives a message that consists of "ack" and (apparently) a ciphertext, A tries to decrypt the second component using K_A^{-1}. If the decryption succeeds, then A extracts the corresponding nonces N_A and N_B and key K_B, and checks that it has recently sent N_A under K_B. If the decryption or the checks fail, then A does nothing.

Afterwards, A and B may use N_A and N_B as shared secrets. In particular, A and B may use N_B as a session key, or they may compute session keys by concatenating and hashing the two nonces.

This protocol has some deliberate similarities with several previous ones [13, 10, 9]. However, unlike those protocols, it aims to preserve the privacy of the participants, first of all by not publishing their names in cleartext, and also for example through the decoy message. The inclusion of this message prevents an attack where a malicious principal $C \notin S_B$ computes and sends

$$\text{"hello"}, \{\text{"hello"}, N_C, K_A\}_{K_B}$$

and then deduces B's presence and $A \in S_B$ by noticing a response. Moreover, B's response to A when $A \notin S_B$ is a decoy message that any other principal could have sent, so that A cannot confirm B's presence in this case.

2.3 Properties and Limitations

Intuitively, the protocol is supposed to establish the shared secrets N_A and N_B. At the very least, we would expect that A and B, and only them, can derive a session key K from these secrets. We would expect, moreover, that this key be essentially independent of any other data. For example, it should not be possible for an attacker without access to K to compute a ciphertext under K from a record of the protocol messages. In short, K should behave much like a pre-established shared key. The only observable differences between running the protocol and having a pre-established shared key should be that an attacker can disrupt a protocol run, making it fail, and that an attacker can notice that the protocol generates some opaque messages. Our results of section 5 provide a more precise statement of this comparison, in the form of an equivalence.

The protocol is also supposed to assure A and B of each other's identity. However, the two participants have somewhat different states in this respect at the conclusion of a key exchange. The initiator, A, has evidence that it shares the session key K with the principal B that responded. On the other hand, B has evidence that it shares K at most with A, but cannot be certain that A initiated the protocol run. Any other principal C might have contacted B pretending to be A, but then C will not obtain the key. Only after further communication can B be sure of A's participation in the session.

In addition, the protocol is supposed to protect the identity of the participants. This should mean, in particular, that an attacker cannot learn anything when A wishes to communicate with B but not vice versa. It should also mean that an attacker cannot distinguish a run between A and B from a run between two other principals A' and B', under appropriate hypotheses. The hypotheses should say, for example, that B is not the attacker, since B learns A's identity. The hypotheses should also limit what the participants can do besides running the protocol. For example, if A were to broadcast "A knows some nonces!" after every protocol run, then A's identity would clearly not be protected. More generally, the hypotheses need to address possible leaks not caused by the protocol proper. Section 6 develops these hypotheses and gives our privacy results, also relying on equivalences.

3 The Applied Pi Calculus (Overview)

The applied pi calculus is a simple, general extension of the pi calculus with value passing, primitive function symbols, and equations between terms. In [2], we introduce this calculus, develop semantics and proof techniques, and apply those techniques in reasoning about some security protocols. This section gives only a brief overview.

3.1 Syntax and Informal Semantics

A *signature* Σ consists of a finite set of function symbols, such as h and $\mathsf{decrypt}$, each with an integer arity. Given a signature Σ, an infinite set of names, and an infinite set of variables, the set of *terms* is defined by the grammar:

$U, V ::=$	terms
$a, n, \ldots$	name
$x, y, \ldots$	variable
$f(U_1, \ldots, U_l)$	function application

where f ranges over the function symbols of Σ and l matches the arity of f. We use meta-variables u and v to range over both names and variables.

The grammar for *processes* is similar to the one in the pi calculus, except that here messages can contain terms (rather than only names) and that names need not be just channel names:

$P, Q, R ::=$	processes (or plain processes)
$\mathbf{0}$	null process
$P \mid Q$	parallel composition
$!P$	replication
$\nu n.P$	name restriction ("new")
if $U = V$ *then* P *else* Q	conditional
$u(x).P$	message input
$\overline{u}\langle V\rangle.P$	message output

The null process $\mathbf{0}$ does nothing; $P \mid Q$ is the parallel composition of P and Q; the replication $!P$ behaves as an infinite number of copies of P running in parallel. The process $\nu n.P$ makes a new name n then behaves as P. The conditional construct *if* $U = V$ *then* P *else* Q is standard, but we should stress that $U = V$ represents equality, rather than strict syntactic identity. We abbreviate it *if* $U = V$ *then* P when Q is $\mathbf{0}$. Finally, the input process $u(x).P$ is ready to input from channel u, then to run P with the actual message replaced for the formal parameter x, while the output process $\overline{u}\langle V\rangle.P$ is ready to output message V on channel u, then to run P. In both of these, we may omit P when it is $\mathbf{0}$.

Further, we extend processes with *active substitutions*:

$A, B, C ::=$	extended processes
P	plain process
$A \mid B$	parallel composition
$\nu n.A$	name restriction
$\nu x.A$	variable restriction
$\{x = V\}$	active substitution

We write $\{x = V\}$ for the substitution that replaces the variable x with the term V. The substitution $\{x = V\}$ typically appears when the term V has been sent to the environment, but the environment may not have the atomic names that appear in V; the variable x is just a way to refer to V in this situation. The substitution $\{x = V\}$ is active in the sense that it "floats" and applies to any process that comes into contact with it. In order to control this contact, we may add a variable restriction: $\nu x.(\{x = V\} \mid P)$ corresponds exactly to *let* $x = V$ *in* P. Although the substitution $\{x = V\}$ concerns only one variable, we can build bigger substitutions by parallel composition. We always assume that our substitutions are cycle-free. We also assume that, in an extended process, there is at most one substitution for each variable, and there is exactly one when the variable is restricted.

A *frame* is an extended process built up from active substitutions by parallel composition and restriction. Informally, frames represent the static knowledge gathered by the environment after communications with an extended process. We let φ range over frames, and let $\varphi(A)$ be the frame obtained from the extended process A by erasing all plain subprocesses of A. We write $(U = V)\varphi$ when U and V are equal up to φ [2, section 4.2]. An *evaluation context* $C[\_]$ is an extended process with a hole in the place of an extended process. As usual, names and variables have scopes, which are delimited by restrictions and by inputs. When E is any expression, $fv(E)$, $bv(E)$, $fn(E)$, and $bn(E)$ are the sets of free and bound variables and free and bound names of E, respectively.

We rely on a sort system for terms and extended processes [2, section 2]. We always assume that terms and extended processes are well-sorted and that substitutions and context applications preserve sorts.

3.2 Operational Semantics

Given a signature Σ, we equip it with an equational theory (that is, with an equivalence relation on terms with certain closure properties). We write $\Sigma \vdash U = V$ when the equation $U = V$ is in the theory associated with Σ. We usually keep the theory implicit, and abbreviate $\Sigma \vdash U = V$ to $U = V$ when Σ is clear from context or unimportant.

Structural equivalences, written $A \equiv B$, relate extended processes that are equal by any capture-avoiding rearrangements of parallel compositions, restrictions, and active substitutions, and by equational rewriting of any terms in processes. *Reductions*, written $A \rightarrow B$, represent silent steps of computation (in particular, internal message transmissions and branching on conditionals). *Labelled transitions*, written $A \xrightarrow{\alpha} B$, represent interactions with the environment. They consist of message inputs and message outputs, respectively written $A \xrightarrow{a(U)} B$ and $A \xrightarrow{\nu\tilde{u}.\overline{a}\langle U\rangle} B$. An output transition $A \xrightarrow{\nu\tilde{u}.\overline{a}\langle U\rangle} B$ is enabled only if the message U is a very simple term, typically a fresh variable x. Nonetheless, B may contain an active substitution that associates x with any term. An input transitions $A \xrightarrow{a(U)} B$ may use variables defined in A (typically from previous message outputs) to form the message U. Reductions and labelled transitions are closed by structural equivalence, hence by equational rewriting on terms.

3.3 Examples

We further explain the applied pi calculus with examples motivated by the protocol under consideration. We start with formatted messages. We then discuss one-way hash functions and encryption functions.

In our protocol, we use two kinds of formated messages ("hello" and "ack") with two and three variable fields, respectively. Accordingly, we introduce binary and ternary function symbols $\mathsf{hello}(\_,\_)$ and $\mathsf{ack}(\_,\_,\_)$ in the signature Σ; these symbols represent the message constructors. In addition, we introduce inverse, unary function symbols $\mathsf{hello.0}\,(\_)$, $\mathsf{hello.1}\,(\_)$, $\mathsf{ack.0}\,(\_)$, $\mathsf{ack.1}\,(\_)$, and $\mathsf{ack.2}\,(\_)$ in order to select particular fields in messages. Finally, we describe the intended behavior of formatted messages with the evident equations:

$$
\begin{aligned}
\mathsf{hello.0}\,(\mathsf{hello}(x_0, x_1)) &= x_0 \\
\mathsf{hello.1}\,(\mathsf{hello}(x_0, x_1)) &= x_1 \\
\mathsf{ack.0}\,(\mathsf{ack}(y_0, y_1, y_2)) &= y_0 \\
\mathsf{ack.1}\,(\mathsf{ack}(y_0, y_1, y_2)) &= y_1 \\
\mathsf{ack.2}\,(\mathsf{ack}(y_0, y_1, y_2)) &= y_2
\end{aligned}
$$

(A first equational theory may consists of these equations, and all equations obtained by reflexivity, symmetry, and transitivity and by substituting terms for the variables $x_0, \ldots, y_2$.)

In order to model the one-way hash computation of a session key out of the nonces N_A and N_B, we introduce a binary function symbol $\mathsf{h}(\_,\_)$ with no equations. The fact that $\mathsf{h}(N_A, N_B) = \mathsf{h}(N'_A, N'_B)$ only when $N_A = N'_A$ and $N_B = N'_B$ models that h is collision-free. The absence of an inverse for h models the one-wayness of h. In our protocol, these properties are important to guarantee that $\mathsf{h}(N_A, N_B)$ is indeed secret (as long as N_A or N_B is) and, further, that the attacker cannot recover N_A or N_B even if it obtains $\mathsf{h}(N_A, N_B)$.

In order to model symmetric cryptography (that is, shared-key cryptography), we may introduce binary function symbols $\mathsf{encrypt}(\_,\_)$ and $\mathsf{decrypt}(\_,\_)$ for encryption and decryption, respectively, with the equation:

$$\mathsf{decrypt}(\mathsf{encrypt}(x, y), y) = x \tag{1}$$

Here x represents the plaintext and y the key. We often use the notation $\{U\}_V$ instead of $\mathsf{encrypt}(U, V)$. For instance, the (useless) process $\nu K.\bar{c}\langle\{U\}_K\rangle$ sends the term U encrypted under a fresh key K on channel c. It is only slightly harder to model asymmetric (public-key) cryptography, where the keys for encryption and decryption are different. In addition to $\mathsf{encrypt}(\_,\_)$ and $\mathsf{decrypt}(\_,\_)$, we introduce the unary function symbol $\mathsf{pk}(\_)$ for deriving a public key from a private key. Instead of (1), we use the equation:

$$\mathsf{decrypt}(\mathsf{encrypt}(x, \mathsf{pk}(y)), y) = x \tag{2}$$

Since there is no inverse for $\mathsf{pk}(\_)$, the public key $\mathsf{pk}(s)$ can be passed to the environment without giving away the capability to decrypt messages encrypted under $\mathsf{pk}(s)$.

For instance, a principal B with public key K_B can be represented as a process in a context $P_B[\_] \stackrel{\text{def}}{=} \nu s.\,(\{K_B = \mathsf{pk}(s)\} \mid [\_])$ that binds a decryption key s and exports the associated encryption key as a variable K_B. As this example indicates, we essentially view ν as a generator of unguessable seeds. In some cases, those seeds may be directly used as passwords or keys; in others, some transformations are needed.

3.4 Observational Equivalences

In the analysis of protocols, we frequently argue that two given processes cannot be distinguished by any context, that is, that the processes are observationally equivalent. As in the spi calculus, the context represents an active attacker, and equivalences capture security properties in the presence of the attacker. The applied pi calculus has a useful, general theory of observational equivalence parameterized by Σ and its equational theory [2]. Specifically, the following three relations are defined for any Σ and equational theory:

- *Static equivalence*, written $\approx_s$, relates frames that cannot be distinguished by any term comparison. In the presence of the "new" construct, the relation $\approx_s$ is somewhat delicate and interesting.
  For instance, we have
  $$\nu N.\{x = \mathsf{h}(N, K_B)\} \approx_s \nu N.\{x = \mathsf{h}(N, K_C)\}$$
  for any K_B and K_C, since the nonce N guarantees that both terms substituted for x have the same (null) equational properties, but
  $$\nu N.\{x = \mathsf{hello}(N, K_B)\} \not\approx_s \nu N.\{x = \mathsf{hello}(N, K_C)\}$$
  as soon as K_B and K_C differ, since the comparison $\mathsf{hello.1}\,(x) = K_B$ succeeds only with the first frame.
- More generally, *contextual equivalence* relates extended processes that cannot be distinguished by any evaluation context in the applied pi calculus, with any combination of messaging and term comparisons.
- *Labelled bisimilarity*, written $\approx_l$, coincides with contextual equivalence, but it is defined in terms of labelled transitions instead of arbitrary evaluation contexts, and it is the basis for standard, powerful proof techniques.

4 The Protocol in the Applied Pi Calculus

In this section we give a precise model for the protocol described in section 2.2: we first choose an adequate equational theory, then detail our representation of principals and attackers, and finally give processes that express the protocol.

4.1 An Equational Theory

The following grammar of terms indicates the function symbols and notation conventions that we use:

$T, U, V, V_0, \cdots ::=$	terms
$A, B, K, x_1, x_2, \ldots$	variable
$c_1, c_2, \mathit{init}_A, \mathit{accept}_B, \mathit{connect}_A, \ldots$	name (for a channel)
$N, N_A, K_A^{-1}, \ldots$	name (typically for nonces and keys)
$\mathsf{h}(U, V)$	cryptographic hash
$\mathsf{pk}(U)$	public-key derivation
$\{T\}_V$	public-key encryption
$\mathsf{decrypt}(W, U)$	private-key decryption

$\mathsf{hello}(U_0, U_1), \mathsf{ack}(V_0, V_1, V_2)$	constructor for protocol message
$\mathsf{hello.0}\,(U)\,, \ldots, \mathsf{ack.2}\,(V)$	field selector for protocol message
$\emptyset$	empty set
$U.V$	set extension

This grammar includes primitives for constructing sets ($\emptyset$ and .) but not a set membership relation. We write $V \in W$ as an abbreviation for $W.V = W$.

Our equational theory is fairly standard. The equations on terms are:

$\mathsf{decrypt}(\{x\}_{\mathsf{pk}(z)}, z) = x$	private-key decryption
$\mathsf{hello.j}\,(\mathsf{hello}(x_0, x_1)) = x_j$	field selection in "hello" message
$\mathsf{ack.j}\,(\mathsf{ack}(x_0, x_1, x_2)) = x_j$	field selection in "ack" message
$(\emptyset.x).x = \emptyset.x$	idempotence of set extension
$(x.y).z = (x.z).y$	associativity of set extension

The equational theory implicitly assumes that encryption is "which-key concealing", in the sense that someone who sees a message encrypted under a public key K should not be able to tell that it is under K without knowledge of the corresponding private key K^{-1}. On the other hand, it would be easy to add functions and equations that negate this property, in order to model additional capabilities of an attacker. In particular, for the benefit of the attacker, we could add the function symbols get-key, test-key, or same-key, with respective equations:

$$\mathsf{get\text{-}key}(\{x\}_z) = z$$
$$\mathsf{test\text{-}key}(\{x\}_z, z) = \mathsf{true}$$
$$\mathsf{same\text{-}key}(\{x\}_z, \{y\}_z) = \mathsf{true}$$

These additions would not affect authentication and secrecy properties, but they would compromise privacy properties.

4.2 The Principals

We model arbitrary configurations of principals. Each principal may run any number of sessions, as initiator and responder, and may perform other operations after session establishment or even independently of the protocol. Only some of these principals are trustworthy. We are interested in the security properties that hold for them.

Our model of a principal A has two parts: an implementation of the protocol, written P_A, and a "user process" (or "user protocol"), written U_A. The user process defines any additional behavior, such as when protocol runs are initiated and what happens after each session establishment. It consumes the shared secrets produced during the establishment of sessions and uses these secrets, perhaps to do something useful. According to the user process, each principal may run several sessions of the protocol, possibly playing both the role of initiator and that of responder. Of course, security properties depend on both P_A and U_A. We define P_A below in section 4.4; on the other hand, we treat U_A as a parameter.

We use the following control interface between the (abstract) user process and the (specific) session-establishment protocol. The interface concerns both the roles of session initiator and responder.

***init*:** Principal A sends $\overline{init_A}\langle B\rangle$ to trigger a session-establishment attempt with principal B.

***accept*:** The responder part of the protocol for principal B sends $\overline{accept_B}\langle A, K\rangle$ to notify principal B that it has accepted a session apparently from principal A, with session key K.

***connect*:** The initiator part of the protocol for principal A sends $\overline{connect_A}\langle B, K\rangle$ to notify principal A that its attempt to contact B succeeded, with session key K.

In addition, for each principal B, the set S_B represents all acceptable interlocutors for B. For simplicity, we do not provide an interface for updating this set, so it remains constant. Thus, the interface between the session protocol and the user process for each principal X consists of the communication channels $\mathcal{V}_X \stackrel{\text{def}}{=} \{init_X,\ accept_X,\ connect_X\}$ plus a (constant) set of principals S_X.

Note that the interface provides a key K to the user process, rather than nonces N_A and N_B. We prefer to define K in such a way that N_A and N_B cannot be computed from K (for example, $K = \mathsf{h}(N_A, N_B)$). Our results can thus be independent of how the user process applies K.

As suggested by the informal description of the protocol, we represent the identity of each principal as its public key, using variables A, B, E, X, ... for both identities and public keys (rather than A, B, K_A, and K_B as in section 2.2). For the present purposes, the essence of a principal lies in its ability to decrypt any message encrypted under its public key. Accordingly, we associate a context of the form

$$PK_A\,[\_] \stackrel{\text{def}}{=} \nu K_A^{-1}.(\{A = \mathsf{pk}(K_A^{-1})\} \mid [\_])$$

with every principal identity A. This context restricts the use of the decryption key K_A^{-1} to the process in the context and it exports the corresponding public key. Whenever we put a process R in this context, our intent is that R never communicates K_A^{-1} to the environment.

By definition of well-formed configurations in the applied pi calculus, a process of the form $C[PK_A\,[R]]$ exports A, only R can access K_A^{-1}, and we cannot apply a context that would redefine A. On the other hand, $C[\_]$ can define any number of other principals. Thus, we obtain a fairly generous and convenient model when we represent an attacker by an arbitrary context.

For example, the process $PK_A\,[\mathbf{0}]$ indicates that A is a principal whose decryption key is never used. This process concisely models an absent principal.

4.3 The Network and the Attacker

In our model of the protocol, network messages are transmitted on the channels named c_1 and c_2. These represent two public communication channels, or a single public channel, perhaps the ether, in which tags serve for differentiating traffic flows.

As explained in section 2, we assume that an attacker can interpose itself on all public communication channels. In our model, an arbitrary environment (an arbitrary evaluation context) represents the attacker. This environment can interact with the configuration of principals using labelled transitions on any free channel name. We obtain an attractively simple representation of broadcast communication: each message is simply made available to the attacker, on a public channel, and the attacker may then decide to transmit the message, again on a public channel, to one or more principals.

In addition, we sometimes model a weaker, passive attacker. An attack step—that is, eavesdropping on a message—amounts to a message interception (formally, with an output label) followed by a re-emission of the same message (with an input label). We write $A \xrightarrow{\nu\widetilde{u}.c[\widetilde{V}]} A'$ as a shorthand for the sequence of two transitions $A \xrightarrow{\nu\widetilde{u}.\bar{c}\langle\widetilde{V}\rangle}\xrightarrow{c(\widetilde{V})} A'$.

4.4 The Protocol

In this section we give a formal counterpart to the description of message flows of section 2.2.

Messages We rely on substitutions in order to define the protocol messages and the key derivation, as follows.

$$\begin{aligned}
\sigma_1 &\stackrel{\text{def}}{=} \{x_1 = \{\mathsf{hello}(N_A, A)\}_B\} \\
\sigma_2 &\stackrel{\text{def}}{=} \{x_2 = \{\mathsf{ack}(N_A, N_B, B)\}_A\} \\
\sigma_2^\circ &\stackrel{\text{def}}{=} \{x_2 = N_B\} \\
\sigma_K &\stackrel{\text{def}}{=} \{K = \mathsf{h}(N_A, N_B)\}
\end{aligned}$$

Although N_A and N_B are free here, they represent fresh nonces. They will be bound in any process that introduces these substitutions. The substitution σ_2° corresponds to the responder's decoy message, in which here we use a name rather than a ciphertext, for simplicity.

Syntactic sugar We sometimes use the following abbreviations.

For testing, we write *if* $U_1 = V_1$ *and* $U_2 = V_2$ *then* P *else* Q for the process *if* $U_1 = V_1$ *then* (*if* $U_2 = V_2$ *then* P *else* Q) *else* Q, and rely on other similar abbreviations.

For decryption, we use pattern matching on message contents. Specifically, we write

$$\textit{if } x = \{\mathsf{ack}(N_A, \nu N_B, B)\}_A \textit{ using } K_A^{-1} \textit{ then } P \textit{ else } Q$$

for the process

$$\nu N_B.\begin{pmatrix}\{N_B = \mathsf{ack.1}\left(\mathsf{decrypt}(x, K_A^{-1})\right)\} \mid \\ \mathit{if}\ x = \{\mathsf{ack}(N_A, N_B, B)\}_A\ \mathit{then}\ P\ \mathit{else}\ Q\end{pmatrix}$$

with the assumption that $N_B \notin fv(Q)$, and we use analogous abbreviations with νA and νN_A. Here, we use the identifiers N_A and N_B as variables rather than names, locally.

For filtering duplicate messages, we write

$$!c_1(x \setminus V).\mathit{if}\ x\ \mathit{fresh}\ \mathit{then}\ P\ \mathit{else}\ Q$$

for the process

$$\nu c.\,(\overline{c}\langle V\rangle \mid !c_1(x).c(s).(\overline{c}\langle s.x\rangle \mid \mathit{if}\ x \in s\ \mathit{then}\ Q\ \mathit{else}\ P))$$

where c is a fresh channel name and s is a fresh variable. We use channel c for maintaining a set V of previously received messages; Q is triggered instead of P when one of those messages is received again.

Processes The following code represents the protocol. It includes definitions of processes both for the initiator role (with A as initiator) and for the responder role (with B as responder).

$$\begin{aligned}
&P_A \stackrel{\text{def}}{=} I_A \mid R_A \\
&I_A \stackrel{\text{def}}{=} !\mathit{init}_A(B).\nu N_A.\,(\overline{c}_1\langle x_1\sigma_1\rangle \mid I'_A) \\
&I'_A \stackrel{\text{def}}{=} c_2(x_2).\mathit{if}\ x_2 = \{\mathsf{ack}(N_A, \nu N_B, B)\}_A\ \mathit{using}\ K_A^{-1}\ \mathit{then}\ \overline{\mathit{connect}_A}\langle B, K\sigma_K\rangle
\end{aligned}$$

$$\begin{aligned}
R_B \stackrel{\text{def}}{=}\ & !c_1(x_1 \setminus \emptyset). \\
&\quad \mathit{if}\ x_1\ \mathit{fresh}\ \mathit{and}\ x_1 = \{\mathsf{hello}(\nu N_A, \nu A)\}_B\ \mathit{using}\ K_B^{-1}\ \mathit{and}\ A \in S_B \\
&\quad \mathit{then}\ \nu N_B.\,\left(\overline{c}_2\langle x_2\sigma_2\rangle \mid \overline{\mathit{accept}_B}\langle A, K\sigma_K\rangle\right)\ \mathit{else}\ \nu N_B.\overline{c}_2\langle x_2\sigma_2^\circ\rangle
\end{aligned}$$

Here, I_A shows the initiator receiving an initiation request on channel init_A and sending the first protocol message; I'_A then shows the initiator receiving and checking a response, and passing a session key on channel $\mathit{connect}_A$ if the response is satisfactory. On the other hand, R_B shows the responder receiving a message, processing it, responding, and in some cases passing a session key on channel accept_B. Both I_A and R_B are replicated processes. According to R_B, the responder filters duplicate messages. This filtering is not suggested by the informal descriptions of the protocol, but we believe that it is a reasonable refinement, with useful consequences.

As coded, the protocol has little resistance to multiplexing errors. In particular, the initiator fails if the first response that it receives is not the expected one. We could add retries without much difficulty, but this aspect of the protocol is mostly irrelevant in the study of safety properties.

4.5 Configurations of Principals

In our statements of security properties (not in the definition of the protocol itself), we distinguish a particular finite, non-empty set $\mathcal{C}$ of compliant principals A, B, ... A compliant principal A is one in which the decryption key K_A^{-1} is used exclusively in the session-establishment protocol. The initial configuration of a single compliant principal A with user process U_A is therefore an extended process of the form:

$$Q_A \stackrel{\text{def}}{=} \nu\mathcal{V}_A.\big(U_A \mid PK_A\,[P_A]\big)$$

This extended process is parameterized by the set S_A, and (at least) exports the variable A and has free channels c_1 and c_2. In Q_A, by definition, U_A does not have access to K_A^{-1}.

Combining several such extended processes, we obtain a global configuration of the form $\prod_{A\in\mathcal{C}} Q_A$ for any set of compliant principals $\mathcal{C}$. Sometimes, however, we do not need to distinguish the user processes of several compliant principals. We can instead group them in a single expression U, letting $U = \prod_{A\in\mathcal{C}} U_A$. Then, letting $\mathcal{V} = \bigcup_{A\in\mathcal{C}} \mathcal{V}_A$, we consider configurations of the form:

$$\begin{aligned} P &\stackrel{\text{def}}{=} \textstyle\prod_{A\in\mathcal{C}} PK_A\,[P_A] \\ Q &\stackrel{\text{def}}{=} \nu\mathcal{V}.\big(U \mid P\big) \end{aligned}$$

We assume that the user processes of compliant principals (U_A and U) never communicate control channels ($\mathcal{V}$) in messages. For instance, the process $\overline{c}_1\langle \mathit{connect}_A\rangle$ cannot be the user process of a compliant principal. This assumption can easily be enforced by the sort system.

We use P in section 5 when we establish security properties that do not depend on U, thus effectively regarding U as part of the attacker. We use Q in section 6, with additional restrictions on U, when we study privacy.

5 Authentication and Secrecy Properties

We begin our analysis of the protocol with traditional properties, namely responder authentication and session-key secrecy. Such standard properties are important, and often a prerequisite for privacy properties. Moreover, their formulation in the applied pi calculus illustrates the use of observational equivalence for expressing security properties. In contrast, many other formalisms for similar purposes rely only on properties of traces, rather than on equivalences.

For a given set of compliant principals $\mathcal{C}$, we study runs of the protocol in the presence of an active attacker, by examining transitions $P \xrightarrow{\eta} P'$ from the configuration P defined above to some configuration P', where η is an arbitrary sequence of labels.

In our statements, we let ω and φ abbreviate the series of actions and the equational "net effect" of a successful run of the protocol, and let ω^- and φ^-

abbreviate the series of actions and the equational “net effect” of a non-accepted run of the protocol:

$$\begin{aligned}
\xrightarrow{\omega} &\overset{\text{def}}{=} \xrightarrow{init_A(B)}\xrightarrow{\nu x_1.c_1[x_1]}\rightarrow_{*}\xrightarrow{\nu x_2.c_2[x_2]}\rightarrow\xrightarrow{\nu K.\overline{accept}_B\langle A,K\rangle}\xrightarrow{\overline{connect}_A\langle B,K\rangle}\\
\xrightarrow{\omega^-} &\overset{\text{def}}{=} \xrightarrow{init_A(B)}\xrightarrow{\nu x_1.c_1[x_1]}\rightarrow_{*}\xrightarrow{\nu x_2.c_2[x_2]}\rightarrow\\
\varphi &\overset{\text{def}}{=} \nu N_A.\,(\sigma_1 \mid \nu N_B.(\sigma_2 \mid \sigma_K))\\
\varphi^- &\overset{\text{def}}{=} (\nu N_A.\sigma_1) \mid (\nu N_B.\sigma_2^{\circ})
\end{aligned}$$

We have that if $A \in S_B$ then

$$P \xrightarrow{\omega} P_{x1} \mid \varphi$$

else

$$P \xrightarrow{\omega^-} P_{x1} \mid \varphi^-$$

where P_{x1} is P updated so that R_B holds an element x_1 in the set of messages it has received. Thus, P may perform a complete run of the protocol, and this run succeeds if authorized by the responder and fails otherwise. More generally, for any P' such that $P \xrightarrow{\eta} P'$, we have that if $A \in S_B$ then

$$P' \xrightarrow{\omega} P'_{x1} \mid \varphi$$

else

$$P' \xrightarrow{\omega^-} P'_{x1} \mid \varphi^-$$

where P'_{x1} is a corresponding update of P'. These results express the functional correctness of the protocol. They hold independently of whether encryption is which-key concealing.

The following theorem relates the two possible outcomes of an actual run to a “magical” outcome $\varphi^{\circ} \overset{\text{def}}{=} \nu N_1.\{x_1 = N_1\} \mid \nu N_2.\{x_2 = N_2\}$ where the two intercepted messages are trivially independent of the principals A and B and of the established key.

Theorem 1 (Secrecy for complete runs). *Let $A, B \in \mathcal{C}$.*

1. *(Success:) If $P \xrightarrow{\eta} P'$ and $A \in S_B$, then $P' \xrightarrow{\omega}\approx_l P' \mid \varphi^{\circ} \mid \nu N.\{K = N\}$.*
   (Failure:) If $P \xrightarrow{\eta} P'$ and $A \notin S_B$, then $P' \xrightarrow{\omega^-}\approx_l P' \mid \varphi^{\circ}$.
2. *Conversely, if $P \xrightarrow{\omega} P''$, then $A \in S_B$ and $P'' \approx_l P \mid \varphi^{\circ} \mid \nu N.\{K = N\}$.*

The active substitution $\nu N.\{K = N\}$ exports the simplest definition of a fresh secret key, a fresh name, rather than an expression computed from x_1 and x_2. Interestingly, φ° and $\nu N.\{K = N\}$ do not depend on A and B at all, so this theorem implies a first privacy guarantee. The equivalences $\approx_l$ are used for rewriting $P'_{x1} \mid \varphi$ and $P'_{x1} \mid \varphi^-$, by simplifying φ and φ^- and by erasing x_1 from the set of messages that R_B has received, returning to the process P' and hiding that a run has occurred. These equivalences hold only if encryption is which-key-concealing. Otherwise, we obtain only:

$$P'_{x1} \mid \varphi \approx_l P'_{x1} \mid (\nu N_A.\sigma_1) \mid (\nu N_A N_B.\sigma_2) \mid (\nu N.\{K = N\})$$

On the right-hand side, we are left with messages x_1 and x_2 that contain the public keys of A and B. Nonetheless, N_A and N_B are bound around σ_1 and σ_2, so the independence and secrecy of the session key are still guaranteed.

A direct corollary concerns two instances P_A and P_B of the protocol in the initial state. This corollary emphasizes the transitions observed by an environment with no access to the control channels.

$$\begin{aligned}
&P_A \mid P_B \mid \overline{init}_A\langle B\rangle \ \rightarrow \xrightarrow{\nu x_1.c_1[x_1]} \rightarrow^* \xrightarrow{\nu x_2.c_2[x_2]} \rightarrow \approx_l \\
&P_A \mid P_B \mid \varphi^\circ \mid \begin{cases} \nu N.\left(\overline{accept}_B\langle A,N\rangle \mid \overline{connect}_A\langle B,N\rangle\right) & \text{if } A \in S_B \\ \mathbf{0} & \text{if } A \notin S_B \end{cases}
\end{aligned}$$

Intuitively, when we erase control messages, we obtain the same trace and equational effect whether or not $A \in S_B$.

Finally, we also obtain an authentication property:

Theorem 2 (Responder authentication). *Suppose that $P \xrightarrow{\eta} P'$ and (1) η has no internal communication step on c_1 and c_2; (2) P' has no immediate output on channel $accept_B$.*
If $\overline{connect}_A\langle B,K\rangle$ occurs in η, then $P \xrightarrow{\omega}\xrightarrow{\eta'} P'$ for some permutation $\omega\eta'$ of η.

In the statement of the theorem, we rely on α-conversion and assume that the names and variables in processes and labels never clash. With this standard assumption, the commutation of two transition steps (when enabled) can be written simply as the commutation of their labels. Conditions 1 and 2 in the theorem are technically convenient, but not essential. Condition 1 rules out traces where a message is not intercepted by the attacker, and is instead transmitted internally. Any internal communication $A \rightarrow A'$ on channel c_i can be decomposed into $A \xrightarrow{\nu x_i.c_i[x_i]} A''$ with $A' \equiv \nu x_i.A''$. Condition 2 rules out traces where the transition $accept_B$ in ω has not occurred and is enabled in P'.

In light of the results above, we can interpret this theorem partly as a correspondence assertion: whenever A receives a connection message after a protocol run, apparently with B, we have that

1. A initiated the session with B;
2. B accepted the session with A;
3. both parties are now sharing a fresh key K, as good as a fresh shared name;
4. intercepted messages x_1 and x_2 are seemingly unrelated to A, B, and K.

6 Privacy Properties

In this section, we focus on privacy properties. For a given set of compliant principals $\mathcal{C}$, we consider the question of whether an attacker can distinguish two user processes U_1 and U_2 when we place these processes in the context $\nu\mathcal{V}.([\_]|P)$ that provides local access to the session-establishment protocol. Therefore, indistinguishability for user processes depends on the identity-protection features of the protocol, and it is coarser than ordinary observational equivalence $\approx_l$ (that is, indistinguishability in all evaluation contexts).

For instance, if U_1 and U_2 each contain a message $\overline{init}_{A_1}\langle B_1\rangle$ and $\overline{init}_{A_2}\langle B_2\rangle$, and if U_1 and U_2 "behave similarly" once a session is established, then U_1 and U_2 are indistinguishable in this specific sense. On the other hand, we have $\overline{init}_{A_1}\langle B_1\rangle \approx_l \overline{init}_{A_2}\langle B_2\rangle$ only if $A_1 = A_2$ and $B_1 = B_2$.

In order to capture this notion of indistinguishability, we introduce a special labelled transition system and a notion of bisimulation. We obtain a general result in terms of that notion of bisimulation, then derive some privacy properties as corollaries. Thus, for the study of a particular protocol, we develop a special notion of observation of user processes. In contrast, in recent, related work [4, 3], we take a standard notion of observation, and develop communication protocols that are secure with respect to it (and which, for instance, rely on "noise" messages in order to hide communication patterns between compliant principals).

We adopt the following notation convention. We write A, B for principals in the set of compliant principals $\mathcal{C}$, and E for a principal not in $\mathcal{C}$.

6.1 A Labelled Transition System

Next we define labelled transitions for user processes with control state. The control state records the sets S_B of acceptable interlocutors and abstractly keeps track of the sessions being negotiated. The labelled transitions reflect only what the environment can observe about these sessions, filtering out identity information.

Formally, a control state ρ consists of two functions, one that maps each principal $B \in \mathcal{C}$ to a set S_B, and the other a finite map from integers to entries t. The entries are of four kinds:

- $A\,B$: a session offer from A to B not yet considered by B.
- $A\ B\ K_i$: a session offer from A to B accepted by B with key K_i (when $A \in S_B$).
- $A\,B\,-$: a session offer from A to B rejected by B (when $A \notin S_B$).
- $A\,E$: a session offer from A to some non-compliant principal E.

For any ρ and any integer i not in ρ's domain, we let $\rho[i \mapsto t]$ be the control state that extends ρ by mapping i to t. We assume that the keys K_i are all distinct. We let $\mathcal{V}_\rho$ be the union of $\mathcal{V}$ with the keys K_i for all integers i in the domain of ρ.

We pair a process with a control state, with the notation $\rho : U$. We assume that K_i is free in U only if ρ maps i to an entry of the form $A\ B\ K_i$. (In Q, the user process U may have free variables defined by P, such as variables A and B that represent compliant principals, or K_i for a computed key. When we consider transitions of U or $\rho : U$, we treat these variables as names.)

Such a pair $\rho : U$ may have the three sorts of transitions $\rho : U \xrightarrow{\gamma} \rho' : U'$ that we define next: ordinary transitions, blinded transitions, and external transitions.

– Ordinary transitions are essentially those of the process U. For any label α that does not contain control channels or keys K_i, we have:

$$\textsc{Lift}\frac{U \xrightarrow{\alpha} U'}{\rho : U \xrightarrow{\alpha} \rho : U'} \text{ when} \begin{array}{l} \mathit{fn}(\alpha) \cap \mathcal{V}_\rho = \emptyset \\ \mathit{bn}(\alpha) \cap (\mathcal{C} \cup \mathcal{V}_\rho) = \emptyset \end{array}$$

– The attacker can blindly intercept all messages sent on public channels by the principals in $\mathcal{C}$ and resend any of these messages later. Specifically, the attacker can detect new session attempts, make responders consider session offers (either genuine or fake), and make initiators consider intercepted "ack" messages. We reflect these actions using blinded transitions $\xrightarrow{\overline{\mathit{init}}\,\nu i}$, $\xrightarrow{\mathit{accept}\,i}$, $\xrightarrow{\mathit{accept}_B(A)}$, and $\xrightarrow{\mathit{connect}\,i}$.

$$\textsc{Init}\frac{U \xrightarrow{\overline{\mathit{init}_A}\langle B\rangle} U'}{\rho : U \xrightarrow{\overline{\mathit{init}}\,\nu i} \rho[i \mapsto A\,B] : U'}$$

$$\textsc{Accept}$$
$$\rho[i \mapsto A\,B] : U \xrightarrow{\mathit{accept}\,i} \begin{cases} \rho[i \mapsto A\,B\,K_i] : U \mid \overline{\mathit{accept}_B}\langle A, K_i\rangle & \text{if } A \in S_B \\ \rho[i \mapsto A\,B\,-\,] : U & \text{if } A \notin S_B \end{cases}$$

$$\textsc{Accept-Fake}$$
$$\rho : U \xrightarrow{\mathit{accept}_B(A)} \begin{cases} \rho : U \mid \nu N.\overline{\mathit{accept}_B}\langle A, N\rangle & \text{if } A \in S_B \\ \rho : U & \text{if } A \notin S_B \end{cases}$$

$$\textsc{Connect}$$
$$\rho[i \mapsto A\,B\,K_i] : U \xrightarrow{\mathit{connect}\,i} \rho : \nu K_i.\, \big(U \mid \overline{\mathit{connect}_A}\langle B, K_i\rangle\big)$$
$$\rho[i \mapsto A\,B\,-\,] : U \xrightarrow{\mathit{connect}\,i} \rho : U$$

– In addition, compliant principals may be willing to open sessions with non-compliant ones. These sessions are also mediated by the protocol, even if they are transparent to the attacker who can in principle decrypt all messages in these sessions. We reflect these actions using external transitions $\xrightarrow{\nu i E.\overline{\mathit{init}_A}\langle i,E\rangle}$, $\xrightarrow{\mathit{accept}_B(W,V)}$, and $\xrightarrow{\mathit{connect}_A(i,E,V)}$, where E is a variable and V and W are terms such that $\mathit{fn}(V) \cap \mathcal{V}_\rho = \mathit{fn}(W) \cap \mathcal{V}_\rho = \emptyset$.

$$\textsc{Init-E}\frac{U \xrightarrow{\nu E.\overline{\mathit{init}_A}\langle E\rangle} U'}{\rho : U \xrightarrow{\nu i E.\overline{\mathit{init}_A}\langle i,E\rangle} \rho[i \mapsto A\,E] : U'} \begin{array}{l} \text{when } (E \neq B)\varphi(U') \\ \text{for all } B \in \mathcal{C} \end{array}$$

$$\textsc{Accept-E} \quad \rho : U \xrightarrow{\mathit{accept}_B(W,V)} \rho : U \mid \overline{\mathit{accept}_B}\langle W, V\rangle \quad \begin{array}{l} \text{when } (W = A)\varphi(U) \\ \text{for some } A \in S_B \setminus \mathcal{C} \end{array}$$

$$\textsc{Connect-E} \quad \rho[i \mapsto A\,E] : U \xrightarrow{\mathit{connect}_A(i,E,V)} \rho : U \mid \overline{\mathit{connect}_A}\langle E, V\rangle$$

6.2 Private Bisimulation

In order to express hypotheses on the observable properties of user processes, we define an ad hoc notion of bisimulation:

Definition 1. Private bisimilarity *($\approx_{\mathcal{C}}$) is the largest symmetric relation $\mathcal{R}$ on extended processes with control state such that, whenever $T_1 \mathrel{\mathcal{R}} T_2$ with $T_1 = \rho_1 : U_1$ and $T_2 = \rho_2 : U_2$, we have:*

1. $\nu\mathcal{V}_{\rho_1}.U_1 \approx_s \nu\mathcal{V}_{\rho_2}.U_2$,
2. *if* $T_1 \rightarrow T_1'$, *then* $T_2 \rightarrow^* T_2'$ *and* $T_1' \mathrel{\mathcal{R}} T_2'$ *for some* T_2',
3. *if* $T_1 \xrightarrow{\gamma} T_1'$ *and* $fv(\gamma) \subseteq dom(\nu\mathcal{V}_{\rho_1}.U_1)$ *and* $bn(\gamma) \cap fn(\nu\mathcal{V}_{\rho_2}.U_2) = \emptyset$, *then* $T_2 \rightarrow^* \xrightarrow{\gamma} \rightarrow^* T_2'$ *and* $T_1' \mathrel{\mathcal{R}} T_2'$ *for some* T_2'.

This definition is an adaptation of that of weak labelled bisimilarity for the applied pi calculus [2, definition 4]. The three clauses are exactly analogous to those for the applied pi calculus, with different transitions in clause 3.

We also let ε range over initial control states, that is, control states that have no session entries and only define sets S_B for $B \in \mathcal{C}$. We write $P(\varepsilon)$ for the protocol P with these sets S_B. When ε is clear from context, we may write (as usual) P instead of $P(\varepsilon)$.

Our main privacy result states that, if two user processes are privately bisimilar (under our new notion of bisimulation), then the two corresponding configurations are observationally equivalent from the environment's point of view. As we show below, this result provides an effective proof technique for privacy properties.

Lemma 1 (Privacy). *If* $\varepsilon_1 : U_1 \approx_{\mathcal{C}} \varepsilon_2 : U_2$, *then*

$$\nu\mathcal{V}.\big(U_1 \mid P(\varepsilon_1)\big) \approx_l \nu\mathcal{V}.\big(U_2 \mid P(\varepsilon_2)\big)$$

The hypothesis $\varepsilon_1 : U_1 \approx_{\mathcal{C}} \varepsilon_2 : U_2$ deals with arbitrary user processes and sets S_B, and is typically not difficult to establish in particular cases. Importantly, its statement does not depend on any detail of the session protocol, only on its control interface. The conclusion $\nu\mathcal{V}.\big(U_1 \mid P(\varepsilon_1)\big) \approx_l \nu\mathcal{V}.\big(U_2 \mid P(\varepsilon_2)\big)$ then says that two composite systems, each with a user process, are indistinguishable.

The converse of Lemma 1 does not quite hold, at least because the definition of labelled transitions is conservative in some respects. (For instance, in that definition, we safely presume that the attacker has a private key associated with any value E that U employs to identify a non-compliant principal.) Thus, user processes that are not privately bisimilar may still be part of undistinguishable systems. Such user processes can easily be excluded with additional hypotheses.

6.3 Applications of the Privacy Lemma

One may formulate and prove many specific privacy properties for the protocol. The various properties may differ, in particular, on which user processes and sets S_B they consider. We give a series of simple examples of such properties. In the examples, the hypotheses can usually be made less demanding, and more specific and complicated. The proofs follow directly from Lemma 1.

We begin with a basic example that concerns the anonymity of failed sessions. Provided that U never inputs on channels $init_X$, if $A \notin S_B$ and $A' \notin S_{B'}$, then replacing $\overline{init}_A\langle B\rangle$ with $\overline{init}_{A'}\langle B'\rangle$ in U does not affect Q (up to observational equivalence).

The next result deals with a single initial session attempt, and states that the session attempt may not compromise any private bisimilarity that would hold after establishing the session.

Theorem 3 (Equivalent sessions). *For $j = 1, 2$, let*

$$U_j \stackrel{def}{=} \overline{init}_{A_j}\langle B_j\rangle \mid connect_{A_j}(B_j, K).V_j$$
$$U'_j \stackrel{def}{=} \nu K.\left(\overline{accept}_{B_j}\langle A_j, K\rangle \mid V_j\right)$$

with $A_j, B_j \in \mathcal{C}$ and $A_j \in S_{B_j}$ in ε_j. If $\varepsilon_1 : U'_1 \approx_{\mathcal{C}} \varepsilon_2 : U'_2$, then $\varepsilon_1 : U_1 \approx_{\mathcal{C}} \varepsilon_2 : U_2$.

For any V_1 and V_2 that do not use the control channels, the private bisimilarity hypothesis holds as soon as $\nu K.V_1 \approx_l \nu K.V_2$. With this additional assumption and Lemma 1, we have a corollary expressed in terms of standard labelled bisimilarity: we obtain that if $\nu K.V_1 \approx_l \nu K.V_2$ then $\nu\mathcal{V}.\big(U_1 \mid P(\varepsilon_1)\big) \approx_l \nu\mathcal{V}.\big(U_2 \mid P(\varepsilon_2)\big)$.

A further privacy property concerns compliant principals that attempt to open sessions with one another but do not perform any action observable by the attacker after establishing a session. (They may for instance use private channels, or public channels with adequate decoys.) We may describe any such configuration of principals in $\mathcal{C}$ by a parallel compositions of $init_A$ messages with $A \in \mathcal{C}$, plus the sets $(S_B)_{B\in\mathcal{C}}$. In this special case, we easily characterize the equivalence of two configurations:

Theorem 4 (Silent sessions). *Let U_1 and U_2 be parallel compositions of messages $\overline{init}_A\langle X\rangle$ with $A \in \mathcal{C}$. If*

1. *U_1 and U_2 contain the same number of messages,*
2. *U_1 and U_2 contain exactly the same messages $\overline{init}_A\langle W\rangle$ for $W \notin \mathcal{C}$, and*
3. *the sets $S_B \setminus \mathcal{C}$ are identical in ε_1 and ε_2,*

then $\nu\mathcal{V}.\big(U_1 \mid P(\varepsilon_1)\big) \approx_l \nu\mathcal{V}.\big(U_2 \mid P(\varepsilon_2)\big)$.

In order to prove the theorem, we may establish $\varepsilon_1 : U_1 \approx_{\mathcal{C}} \varepsilon_2 : U_2$ by enumerating a few blinded and external transitions, then apply Lemma 1. Conversely, these conditions seem clearly necessary: the attacker can count the number of "hello" messages, can decrypt "hello" messages sent to principals outside $\mathcal{C}$ (as long as W is a public key not in $\mathcal{C}$), and can attempt to establish a session with any $B \in \mathcal{C}$.

We can derive other similar privacy results for uniform families of user processes, such as processes that do not use any principal identity after establishing sessions.

Our final result relates a configuration with a present but silent principal to a configuration with an absent principal. (This theorem does not require Lemma 1; it has a simple, direct bisimulation proof.)

Theorem 5 (Absent principal). *Assume that $|\mathcal{C}| > 1$, and let $X \notin \mathcal{C}$ and $S_X = \emptyset$. We have:*

$$Q \mid \nu \mathcal{V}_X.PK_X\,[P_X] \approx_l Q \mid PK_X\,[\mathbf{0}]$$

The process on the left-hand side is structurally equivalent to a configuration Q' with compliant principals $\mathcal{C} \cup \{X\}$; the process on the right-hand side includes an absent principal (a principal X whose decryption key is never used). Hence, one may first use private bisimilarity to show that X is apparently not involved in any session in Q', then apply Theorem 5 to substitute an absent principal for X. (Conversely, if $\mathcal{C} = \{\}$ or $\mathcal{C} = \{A\}$, then the addition of any instance of the protocol is observable.)

7 Conclusions

This paper reports on the analysis of a protocol in the applied pi calculus, covering standard authenticity and secrecy properties and also privacy (identity protection) properties. The formulation of these properties mainly relies on equivalences, which express indistinguishability in an arbitrary context. Our analysis concerns not only the core of the protocol but also its composition with a user process, since this composition may endanger privacy properties. Thus, we examine the protocol under several hypotheses on user processes. We obtain several related results that transfer hypotheses on user processes to security properties of complete systems.

The literature contains many other formal treatments of protocols and of their security properties. We will not attempt to survey that work here, but mention only the two most relevant papers. One of them is our original paper on the applied pi calculus [2], which considers session establishment and some of its properties, and which includes additional references. The other is a recent paper by Shmatikov and Hughes that defines several notions of anonymity and privacy [16], and sketches—in just a few sentences—an analysis of the protocol that is the subject of this paper. Shmatikov and Hughes develop a special formal framework for protocols, communication graphs. Despite some thematic overlap, the applied pi calculus appears to be richer than communication graphs. In particular, communication graphs do not include an account of user processes. While the definitions of anonymity and privacy seem appropriate and useful for communication graphs, it is not yet entirely clear whether and how they would carry over to the applied pi calculus and other settings.

Acknowledgments

We are grateful to Vitaly Shmatikov for helpful discusions, and to the editors for their patience. Part of Martín Abadi's work was done at Microsoft Research, Silicon Valley. Martín Abadi's work was also partly supported by the National Science Foundation under Grants CCR-0204162 and CCR-0208800.

References

1. Martín Abadi. Private authentication. In *Proceedings of the Workshop on Privacy Enhancing Technologies (PET 2002)*, LNCS. Springer-Verlag, 2002. To appear.
2. Martín Abadi and Cédric Fournet. Mobile values, new names, and secure communication. In *Proceedings of the 28th ACM Symposium on Principles of Programming Languages (POPL 2001)*, pages 104–115. ACM, January 2001.
3. Martín Abadi, Cédric Fournet, and Georges Gonthier. Authentication primitives and their compilation. In *Proceedings of the 27th ACM Symposium on Principles of Programming Languages (POPL 2000)*, pages 302–315. ACM, January 2000.
4. Martín Abadi, Cédric Fournet, and Georges Gonthier. Secure implementation of channel abstractions. *Information and Computation*, 174(1):37–83, April 2002.
5. William Aiello, Steven M. Bellovin, Matt Blaze, Ran Canetti, John Ionnidis, Angelos D. Keromytis, and Omer Reingold. Efficient, DoS-resistant, secure key exchange for internet protocols. In Vijay Atluri, editor, *Proceedings of the 9th ACM Conference on Computer and Communications Security (CCS 2002)*, pages 48–58. ACM, November 2002.
6. Giuseppe Ateniese, Amir Herzberg, Hugo Krawczyk, and Gene Tsudik. On traveling incognito. *Computer Networks*, 31(8):871–884, 1999.
7. Hannes Federrath, Anja Jerichow, and Andreas Pfitzmann. MIXes in mobile communication systems: Location management with privacy. In Ross J. Anderson, editor, *Information hiding: First international workshop*, volume 1174 of *LNCS*, pages 121–135. Springer-Verlag, 1996.
8. Markus Jakobsson and Susanne Wetzel. Security weaknesses in Bluetooth. In *Topics in Cryptology - CT-RSA 2001, Proceedings of the Cryptographer's Track at RSA Conference 2001*, volume 2020 of *LNCS*, pages 176–191. Springer-Verlag, 2001.
9. Hugo Krawczyk. SKEME: A versatile secure key exchange mechanism for internet. In *Proceedings of the Internet Society Symposium on Network and Distributed Systems Security*, February 1996. Available at `http://bilbo.isu.edu/sndss/sndss96.html`.
10. Butler Lampson, Martín Abadi, Michael Burrows, and Edward Wobber. Authentication in distributed systems: Theory and practice. *ACM Transactions on Computer Systems*, 10(4):265–310, November 1992.
11. Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography.* CRC Press, 1996.
12. Refik Molva, Didier Samfat, and Gene Tsudik. Authentication of mobile users. *IEEE Network*, 8(2):26–35, March/April 1994.
13. Roger M. Needham and Michael D. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12):993–999, December 1978.
14. Michael G. Reed, Paul F. Syverson, and David M. Goldschlag. Protocols using anonymous connections: Mobile applications. In B. Christianson, B. Crispo, M. Lomas, and M. Roe, editors, *Security Protocols: 5th International Workshop*, volume 1361 of *LNCS*, pages 13–23. Springer-Verlag, 1997.
15. Didier Samfat, Refik Molva, and N. Asokan. Untraceability in mobile networks. In *Proceedings of the First Annual International Conference on Mobile Computing and Networking (MobiCom 1995)*, pages 26–36, 1995.
16. Vitaly Shmatikov and Dominic Hughes. Defining anonymity and privacy (extended abstract). In *Workshop on Issues in the Theory of Security (WITS' 02)*, January 2002.