The Applied Pi Calculus: Mobile Values, New Names, and Secure Communication

MARTÍN ABADI, Google Brain
BRUNO BLANCHET, Inria
CÉDRIC FOURNET, Microsoft Research

We study the interaction of the programming construct "new," which generates statically scoped names, with communication via messages on channels. This interaction is crucial in security protocols, which are the main motivating examples for our work; it also appears in other programming-language contexts.

We define the applied pi calculus, a simple, general extension of the pi calculus in which values can be formed from names via the application of built-in functions, subject to equations, and be sent as messages. (In contrast, the pure pi calculus lacks built-in functions; its only messages are atomic names.) We develop semantics and proof techniques for this extended language and apply them in reasoning about security protocols.

This article essentially subsumes the conference paper that introduced the applied pi calculus in 2001. It fills gaps, incorporates improvements, and further explains and studies the applied pi calculus. Since 2001, the applied pi calculus has been the basis for much further work, described in many research publications and sometimes embodied in useful software, such as the tool ProVerif, which relies on the applied pi calculus to support the specification and automatic analysis of security protocols. Although this article does not aim to be a complete review of the subject, it benefits from that further work and provides better foundations for some of it. In particular, the applied pi calculus has evolved through its implementation in ProVerif, and the present definition reflects that evolution.

CCS Concepts: • **Security and privacy** → **Formal methods and theory of security**; • **Theory of computation** → *Process calculi*;

Additional Key Words and Phrases: Security protocols

ACM Reference format:
Martín Abadi, Bruno Blanchet, and Cédric Fournet. 2017. The Applied Pi Calculus: Mobile Values, New Names, and Secure Communication. *J. ACM* 65, 1, Article 1 (October 2017), 41 pages.
https://doi.org/10.1145/3127586

1 A CASE FOR IMPURITY

Purity often comes before convenience and even before faithfulness in the lambda calculus, the pi calculus, and other foundational programming languages. For example, in the standard pi calculus, the only messages are atomic names (Milner 1999). This simplicity is extremely appealing from

This work was started while Martín Abadi was at Bell Labs Research and continued while he was at the University of California at Santa Cruz and at Microsoft Research.
Authors' addresses: M. Abadi, 1600 Amphitheatre Pkwy, Mountain View, CA 94043, USA; email: abadi@google.com; B. Blanchet, 2 rue Simone Iff, 75012 Paris, France; email: bruno.blanchet@inria.fr; C. Fournet, Microsoft Research, 21 Station road, Cambridge CB1 2FB, UK; email: fournet@microsoft.com.

ACM 0004-5411/2017/10-ART1 $15.00
https://doi.org/10.1145/3127586

a foundational viewpoint, and helps in developing the theory of the pi calculus. Furthermore, ingenious encodings demonstrate that it may not entail a loss of generality. In particular, integers, objects, and even higher-order processes can be represented in the pure pi calculus. Similarly, various encodings of cryptographic operations in the pi calculus have been considered (Abadi and Gordon 1999; Baldamus et al. 2004; Carbone and Maffeis 2003; Martinho and Ravara 2011).

On the other hand, this purity has a price. In applications, the encodings can be futile, cumbersome, and even misleading. For instance, in the study of programming languages based on the pi calculus (such as Pict (Pierce and Turner 2000), JoCaml (Conchon and Fessant 1999), or occam-pi (Welch and Barnes 2005)), there is little point in pretending that integers are not primitive. The encodings may also hinder careful reasoning about communication (e.g., because they require extra messages), and they may complicate static analysis and proofs.

These difficulties are often circumvented through on-the-fly extensions. The extensions range from quick punts ("for the next example, let's pretend that we have a data type of integers") to the laborious development of new calculi, such as the spi calculus (Abadi and Gordon 1999) (a calculus with cryptographic operations) and its variants. Generally, the extensions bring us closer to a realistic programming language or modeling language—that is not always a bad thing.

Although many of the resulting calculi are ad hoc and poorly understood, others are robust and uniform enough to have a rich theory and a variety of applications. In particular, impure extensions of the lambda calculus with function symbols and with equations among terms ("delta rules") have been developed systematically, with considerable success. Similarly, impure versions of CCS and CSP with value passing are not always deep but often neat and convenient (Milner 1989).

In this article, we introduce, study, and use an analogous uniform extension of the pi calculus, which we call the applied pi calculus (by analogy with "applied lambda calculus"). From the pure pi calculus, we inherit constructs for communication and concurrency, and for generating statically scoped new names ("new"). We add functions and equations, much as is done in the lambda calculus. Messages may then consist not only of atomic names but also of values constructed from names and functions. This embedding of names into the space of values gives rise to an important interaction between the "new" construct and value-passing communication, which appears in neither the pure pi calculus nor value-passing CCS and CSP. Further, we add an auxiliary substitution construct, roughly similar to a floating "let"; this construct is helpful in programming examples and especially in semantics and proofs, and serves to capture the partial knowledge that an environment may have of some values.

The applied pi calculus builds on the pure pi calculus and its substantial theory, but it shifts the focus away from encodings. In comparison with ad hoc approaches, it permits a general, systematic development of syntax, operational semantics, equivalences, and proof techniques.

Using the calculus, we can write and reason about programming examples where "new" and value passing appear. First, we can easily treat standard data types (integers, pairs, arrays, etc.). We can also model unforgeable capabilities as new names, then model the application of certain functions to those capabilities. For instance, we may construct a pair of capabilities. More delicately, the capabilities may be pointers to composite structures, and then adding an offset to a pointer to a pair may yield a pointer to its second component (e.g., as in Liblit and Aiken (2000)). Furthermore, we can study a variety of security protocols. For this purpose, we represent fresh channels, nonces, and keys as new names, and primitive cryptographic operations as functions, obtaining a simple but useful programming-language perspective on security protocols (much as in the spi calculus). A distinguishing characteristic of the present approach is that we need not craft a special calculus and develop its proof techniques for each choice of cryptographic operations. Thus, we can express and analyze fairly sophisticated protocols that combine several cryptographic primitives (encryptions, hashes, signatures, XORs, etc.). We can also describe attacks against the protocols that rely

on (equational) properties of some of those primitives. In our work to date, security protocols are our main source of examples.

The next section defines the applied pi calculus. Section 3 introduces some small, informal examples. Section 4 defines semantic concepts, such as process equivalence, and develops proof techniques. Sections 5 and 6 treat larger, instructive examples; they concern a Diffie-Hellman key exchange, cryptographic hash functions, and message authentication codes. (The two sections are independent.) Many other examples now appear in the literature, as explained below. Section 7 discusses related work, and Section 8 concludes. The body of the article contains some proofs and outlines others; many details of the proofs, however, are in appendices.

This article essentially subsumes the conference paper that introduced the applied pi calculus in 2001. It fills gaps, incorporates various improvements, and further explains and studies the applied pi calculus. Specifically, it presents a revised language, with a revised semantics, as explained in Sections 2 and 4. It also includes precise definitions and proofs; these address gaps in the conference paper, discussed in further detail in Section 4. Finally, some of the examples in Sections 3, 5, and especially 6 are polished or entirely new.

Since 2001, the applied pi calculus has been the basis for much further work, described in many research publications (some of which are cited below) and tutorials (Abadi 2007; Cortier and Kremer 2014; Ryan and Smyth 2011). This further work includes semantics, proof techniques, and applications in diverse contexts (key exchange, electronic voting, certified email, cryptographic file systems, encrypted web storage, website authorization, zero-knowledge proofs, and more). It is sometimes embodied in useful software, such as the tool ProVerif (Blanchet 2001, 2004, 2016; Blanchet et al. 2008). This tool, which supports the specification and automatic analysis of security protocols, relies on the applied pi calculus as input language. Other software that builds on ProVerif targets protocol implementations, web security mechanisms, or stateful systems such as hardware devices (Arapinis et al. 2011; Bansal et al. 2012; Bhargavan et al. 2008b). Finally, the applied pi calculus has also been implemented in other settings, such as the prover Tamarin (Kremer and Künnemann 2014; Meier et al. 2013).

Although this article does not aim to offer a complete review of the subject and its growth since 2001, it benefits from that further work and provides better foundations for some of it. In particular, the applied pi calculus has evolved through its implementation in ProVerif, and the present definition reflects that evolution.

2 THE APPLIED PI CALCULUS

In this section, we define the applied pi calculus: its syntax and informal semantics (Section 2.1), then its operational semantics (Section 2.2). We also discuss a few variants and extensions of our definitions (Section 2.3).

2.1 Syntax and Informal Semantics

A *signature* Σ consists of a finite set of function symbols, such as f, encrypt, and pair, each with an arity. A function symbol with arity 0 is a constant symbol.

Given a signature Σ, an infinite set of names, and an infinite set of variables, the set of *terms* is defined by the grammar:

$L, M, N, T, U, V ::=$	terms
$a, b, c, \ldots, k, \ldots, m, n, \ldots, s$	name
x, y, z	variable
$f(M_1, \ldots, M_l)$	function application,

where f ranges over the functions of Σ and l matches the arity of f.

Although names, variables, and constant symbols have similarities, we find it clearer to keep them separate. A term is ground when it does not contain variables (but it may contain names and constant symbols). We use metavariables u, v, w to range over both names and variables. We abbreviate tuples $u_1, \ldots, u_l$ and $M_1, \ldots, M_l$ to $\widetilde{u}$ and $\widetilde{M}$, respectively.

The grammar for *processes* is similar to the one in the pi calculus, but here messages can contain terms (rather than only names) and 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")
$\mathit{if}\ M = N\ \mathit{then}\ P\ \mathit{else}\ Q$	conditional
$N(x).P$	message input
$\overline{N}\langle M\rangle.P$	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, private name n and then behaves as P. The conditional construct $\mathit{if}\ M = N\ \mathit{then}\ P\ \mathit{else}\ Q$ is standard, but we should stress that $M = N$ represents equality, rather than strict syntactic identity. We abbreviate it $\mathit{if}\ M = N\ \mathit{then}\ P$ when Q is $\mathbf{0}$. Finally, $N(x).P$ is ready to input from channel N, then to run P with the actual message replaced for the formal parameter x, while $\overline{N}\langle M\rangle.P$ is ready to output M on channel N, 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
$\{^M/_x\}$	active substitution

We write $\{^M/_x\}$ for the substitution that replaces the variable x with the term M. Considered as a process, $\{^M/_x\}$ is like $\mathit{let}\ x = M\ \mathit{in}\ \ldots$, and is similarly useful. However, unlike a "let" definition, $\{^M/_x\}$ floats and applies to any process that comes into contact with it. To control this contact, we may add a restriction: $\nu x.(\{^M/_x\} \mid P)$ corresponds exactly to $\mathit{let}\ x = M\ \mathit{in}\ P$. The substitution $\{^M/_x\}$ typically appears when the term M has been sent to the environment, but the environment may not have the atomic names that appear in M; the variable x is just a way to refer to M in this situation. Although the substitution $\{^M/_x\}$ concerns only one variable, we can build bigger substitutions by parallel composition, and may write

$$\{^{M_1}/_{x_1}, \ldots, ^{M_l}/_{x_l}\} \quad \text{for} \quad \{^{M_1}/_{x_1}\} \mid \ldots \mid \{^{M_l}/_{x_l}\}.$$

We write σ, $\{^M/_x\}$, $\{^{\widetilde{M}}/_{\widetilde{x}}\}$ for substitutions; $x\sigma$ for the image of x by σ; and $T\sigma$ for the result of applying σ to the free variables of T. We identify the empty substitution and the null process $\mathbf{0}$.

As usual, names and variables have scopes, which are delimited by restrictions and by inputs. We write $\mathit{fv}(A)$ and $\mathit{fn}(A)$ for the sets of free variables and free names of A, respectively. These sets are inductively defined, as detailed in Figure 1. The domain $\mathit{dom}(A)$ of an extended process A is the set of variables that A exports (those variables x for which A contains a substitution $\{^M/_x\}$

$$
\begin{aligned}
fv(x) &\stackrel{\text{def}}{=} \{x\} \\
fv(n) &\stackrel{\text{def}}{=} \emptyset \\
fv(f(M_1, \ldots, M_l)) &\stackrel{\text{def}}{=} fv(M_1) \cup \cdots \cup fv(M_l) \\
fv(\mathbf{0}) &\stackrel{\text{def}}{=} \emptyset \\
fv(P \mid Q) &\stackrel{\text{def}}{=} fv(P) \cup fv(Q) \\
fv(!P) &\stackrel{\text{def}}{=} fv(P) \\
fv(\nu n.P) &\stackrel{\text{def}}{=} fv(P) \\
fv(\mathit{if}\ M = N\ \mathit{then}\ P\ \mathit{else}\ Q) &\stackrel{\text{def}}{=} fv(M) \cup fv(N) \cup fv(P) \cup fv(Q) \\
fv(N(x).P) &\stackrel{\text{def}}{=} fv(N) \cup (fv(P) \setminus \{x\}) \\
fv(\overline{N}\langle M\rangle.P) &\stackrel{\text{def}}{=} fv(N) \cup fv(M) \cup fv(P) \\
fv(A \mid B) &\stackrel{\text{def}}{=} fv(A) \cup fv(B) \\
fv(\nu n.A) &\stackrel{\text{def}}{=} fv(A) \\
fv(\nu x.A) &\stackrel{\text{def}}{=} fv(A) \setminus \{x\} \\
fv(\{^M/_x\}) &\stackrel{\text{def}}{=} fv(M) \cup \{x\}
\end{aligned}
$$

$fn(\cdot)$ is defined as $fv(\cdot)$, except that

$$
\begin{aligned}
fn(x) &\stackrel{\text{def}}{=} \emptyset \\
fn(n) &\stackrel{\text{def}}{=} \{n\} \\
fn(\nu n.P) &\stackrel{\text{def}}{=} fn(P) \setminus \{n\} \\
fn(N(x).P) &\stackrel{\text{def}}{=} fn(N) \cup fn(P) \\
fn(\nu n.A) &\stackrel{\text{def}}{=} fn(A) \setminus \{n\} \\
fn(\nu x.A) &\stackrel{\text{def}}{=} fn(A) \\
fn(\{^M/_x\}) &\stackrel{\text{def}}{=} fn(M)
\end{aligned}
$$

$$
\begin{aligned}
dom(P) &\stackrel{\text{def}}{=} \emptyset \\
dom(A \mid B) &\stackrel{\text{def}}{=} dom(A) \cup dom(B) \\
dom(\nu n.A) &\stackrel{\text{def}}{=} dom(A) \\
dom(\nu x.A) &\stackrel{\text{def}}{=} dom(A) \setminus \{x\} \\
dom(\{^M/_x\}) &\stackrel{\text{def}}{=} \{x\}
\end{aligned}
$$

Fig. 1. Free variables, free names, and domain.

not under a restriction on x). Figure 1 also defines $dom(A)$ formally. We consider that expressions (processes and extended processes) are equal modulo renaming of bound names and variables.

We always assume that our substitutions are cycle-free; that is, by reordering, they can be written $\{^{M_1}/_{x_1}, \ldots, ^{M_l}/_{x_l}\}$, where $x_i \notin fv(M_j)$ for all $i \leq j \leq l$. For instance, we exclude substitutions

$$\frac{u : \tau}{\vdash u : \tau} \qquad \frac{f : \tau_1 \times \cdots \times \tau_l \to \tau \quad \vdash M_1 : \tau_1 \quad \ldots \quad \vdash M_l : \tau_l}{\vdash f(M_1, \ldots, M_l) : \tau}$$

$$\vdash 0 \qquad \frac{\vdash P \quad \vdash Q}{\vdash P \mid Q} \qquad \frac{\vdash P}{\vdash !P} \qquad \frac{\vdash P}{\vdash \nu n.P} \qquad \frac{\vdash M : \tau \quad \vdash N : \tau \quad \vdash P \quad \vdash Q}{\vdash \mathit{if}\ M = N\ \mathit{then}\ P\ \mathit{else}\ Q}$$

$$\frac{\vdash N : \mathsf{Channel} \quad \vdash P}{\vdash N(x).P} \qquad \frac{\vdash N : \mathsf{Channel} \quad \vdash M : \tau \quad \vdash P}{\vdash \overline{N}\langle M \rangle.P}$$

$$\frac{\vdash A \quad \vdash B}{\vdash A \mid B} \qquad \frac{\vdash A}{\vdash \nu u.A} \qquad \frac{x : \tau \quad \vdash M : \tau}{\vdash \{^{M}/_{x}\}}$$

Fig. 2. Sort system.

such as $\{^{f(y)}/_{x}, ^{f(x)}/_{y}\}$. 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, that is, $dom(A) \cap dom(B) = \emptyset$ in every extended process $A \mid B$, and $x \in dom(A)$ in every extended process $\nu x.A$. An extended process A is *closed* when its free variables are all defined by an active substitution, that is, $dom(A) = fv(A)$. We use the abbreviation $\nu\widetilde{u}$ for the (possibly empty) series of pairwise distinct restrictions $\nu u_1.\nu u_2.\ldots.\nu u_l$.

A *frame* is an extended process built up from $\mathbf{0}$ and active substitutions of the form $\{^{M}/_{x}\}$ by parallel composition and restriction. We let φ and ψ range over frames. Every extended process A can be mapped to a frame $\varphi(A)$ by replacing every plain process embedded in A with $\mathbf{0}$. The frame $\varphi(A)$ can be viewed as an approximation of A that accounts for the static knowledge exposed by A to its environment, but not for A's dynamic behavior. Assuming that all bound names and variables are pairwise distinct and do not clash with free ones, one can ignore all restrictions in a frame, thus obtaining an underlying substitution; we require that, for each extended process, this resulting substitution be cycle-free.

We rely on a sort system for terms and processes. It includes a sort Channel for channels. It may also include other sorts such as Integer, Key, or simply a universal sort for data Data. Each variable and each name comes with a sort; we write $u : \tau$ to mean that u has sort τ. There are an infinite number of variables and an infinite number of names of each sort. We typically use a, b, and c as names of sort Channel; s and k as names of some other sort (e.g., Data); and m and n as names of any sort. Function symbols also come with the sorts of their arguments and of their result. We write $f : \tau_1 \times \cdots \times \tau_l \to \tau$ to mean that f has arguments of sorts $\tau_1, \ldots, \tau_l$ and a result of sort τ. Figure 2 gives the rules of the sort system. It defines the following judgments: $\vdash M : \tau$ means that M is a term of sort τ; $\vdash P$ means that the process P is well sorted; $\vdash A$ means that the extended process A is well sorted. This sort system enforces that function applications are well sorted, that M and N are of the same sort in conditional expressions, that N has sort Channel in input and output expressions, that M is well sorted (with an arbitrary sort τ) in output expressions, and that active substitutions preserve sorts. We always assume that expressions are well sorted, and that substitutions preserve sorts.

2.2 Operational Semantics

We give an operational semantics for the applied pi calculus in the now customary "chemical style" (Berry and Boudol 1992; Milner 1992). At the center of this operational semantics is a reduction relation $\to$ on extended processes, which basically models the steps of computations.

For example, $\overline{a}\langle M\rangle \mid a(x).\overline{b}\langle x\rangle \rightarrow \overline{b}\langle M\rangle$ represents the transmission of the message M on the channel a to a process that will forward the message on the channel b; the formal x is replaced with its actual value M in this reduction. The axioms for the reduction relation $\rightarrow$, which are remarkably simple, rely on auxiliary rules for a structural equivalence relation $\equiv$ that permits the rearrangement of processes, for example, the use of commutativity and associativity of parallel composition. Furthermore, both structural equivalence and reduction depend on an underlying equational theory. Therefore, this section introduces equational theories, then defines structural equivalence and reduction.

Given a signature Σ, we equip it with an equational theory, that is, with a congruence relation on terms that is closed under substitution of terms for variables and names. (See, e.g., Mitchell's textbook (Mitchell 1996, Chapter 3) and its references for background on universal algebra and algebraic data types from a programming-language perspective.) We further require that this equational theory respect the sort system (i.e., two equal terms are of the same sort) and that it be nontrivial (i.e., there exist two different terms in each sort).

An equational theory may be generated from a finite set of equational axioms or from rewrite rules, but this property is not essential for us. We tend to ignore the mechanics of specifying equational theories, but give several examples in Section 3.

We write $\Sigma \vdash M = N$ when the equation $M = N$ is in the theory associated with Σ. Here we keep the theory implicit, and we may even abbreviate $\Sigma \vdash M = N$ to $M = N$ when Σ is clear from context or unimportant. We write $\Sigma \nvdash M = N$ for the negation of $\Sigma \vdash M = N$.

As usual, a context is an expression with a hole. An *evaluation context* is a context whose hole is not under a replication, a conditional, an input, or an output. A context $E[\_]$ *closes* A when $E[A]$ is closed.

Structural equivalence $\equiv$ is the smallest equivalence relation on extended processes that is closed by application of evaluation contexts, and such that:

$$
\begin{array}{lrl}
\textsc{Par-0} & A & \equiv A \mid \mathbf{0} \\
\textsc{Par-A} & A \mid (B \mid C) & \equiv (A \mid B) \mid C \\
\textsc{Par-C} & A \mid B & \equiv B \mid A \\
\textsc{Repl} & !P & \equiv P \mid !P \\
\\
\textsc{New-0} & \nu n.\mathbf{0} & \equiv \mathbf{0} \\
\textsc{New-C} & \nu u.\nu v.A & \equiv \nu v.\nu u.A \\
\textsc{New-Par} & A \mid \nu u.B & \equiv \nu u.(A \mid B) \quad \text{when } u \notin fv(A) \cup fn(A) \\
\\
\textsc{Alias} & \nu x.\{^M/_x\} & \equiv \mathbf{0} \\
\textsc{Subst} & \{^M/_x\} \mid A & \equiv \{^M/_x\} \mid A\{^M/_x\} \\
\textsc{Rewrite} & \{^M/_x\} & \equiv \{^N/_x\} \quad \text{when } \Sigma \vdash M = N
\end{array}
$$

The rules for parallel composition and restriction are standard. Alias enables the introduction of an arbitrary active substitution. Subst describes the application of an active substitution to a process that is in contact with it. Rewrite deals with equational rewriting. Subst implicitly requires that $x : \tau$ and $\vdash M : \tau$ for some sort τ. In combination, Alias and Subst yield $A\{^M/_x\} \equiv \nu x.(\{^M/_x\} \mid A)$ for $x \notin fv(M)$:

$$
\begin{array}{rll}
A\{^M/_x\} & \equiv A\{^M/_x\} \mid \mathbf{0} & \text{by Par-0} \\
& \equiv A\{^M/_x\} \mid \nu x.\{^M/_x\} & \text{by Alias}
\end{array}
$$

$$\begin{aligned} &\equiv \nu x.(A\{^{M}/_{x}\} \mid \{^{M}/_{x}\}) \text{ by } \textsc{New-Par} \\ &\equiv \nu x.(\{^{M}/_{x}\} \mid A\{^{M}/_{x}\}) \text{ by } \textsc{Par-C} \\ &\equiv \nu x.(\{^{M}/_{x}\} \mid A) \qquad \text{by } \textsc{Subst} \end{aligned}$$

Using structural equivalence, every closed extended process A can be rewritten to consist of a substitution and a closed plain process with some restricted names:

$$A \equiv \nu \widetilde{n}.(\{^{\widetilde{M}}/_{\widetilde{x}}\} \mid P),$$

where $fv(P) = \emptyset$, $fv(\widetilde{M}) = \emptyset$, and $\{\widetilde{n}\} \subseteq fn(\widetilde{M})$. In particular, every closed frame φ can be rewritten to consist of a substitution with some restricted names:

$$\varphi \equiv \nu \widetilde{n}.\{^{\widetilde{M}}/_{\widetilde{x}}\},$$

where $fv(\widetilde{M}) = \emptyset$ and $\{\widetilde{n}\} \subseteq fn(\widetilde{M})$. The set $\{\widetilde{x}\}$ is the domain of φ.

Internal reduction $\rightarrow$ is the smallest relation on extended processes closed by structural equivalence and application of evaluation contexts such that

$$\begin{array}{lrcl} \textsc{Comm} & \overline{N}\langle x\rangle.P \mid N(x).Q & \rightarrow & P \mid Q \\ \textsc{Then} & \mathit{if}\ M = M\ \mathit{then}\ P\ \mathit{else}\ Q & \rightarrow & P \\ \textsc{Else} & \mathit{if}\ M = N\ \mathit{then}\ P\ \mathit{else}\ Q & \rightarrow & Q \end{array}$$

for any ground terms M and N such that $\Sigma \nvdash M = N$.

Communication (Comm) is remarkably simple because the message concerned is a variable; this simplicity entails no loss of generality because Alias and Subst can introduce a variable to stand for a term:

$$\begin{aligned} \overline{N}\langle M\rangle.P \mid N(x).Q &\equiv \nu x.(\{^{M}/_{x}\} \mid \overline{N}\langle x\rangle.P \mid N(x).Q) \\ &\rightarrow \nu x.(\{^{M}/_{x}\} \mid P \mid Q) \quad \text{by } \textsc{Comm} \\ &\equiv P \mid Q\{^{M}/_{x}\}. \end{aligned}$$

(This derivation assumes that $x \notin fv(N) \cup fv(M) \cup fv(P)$, which can be established by renaming as needed.)

Comparisons (Then and Else) directly depend on the underlying equational theory. Using Else sometimes requires that active substitutions in the context be applied first, to yield ground terms M and N. For example, rule Else does not allow us to reduce $\{^{n}/_{x}\} \mid \mathit{if}\ x = n\ \mathit{then}\ P\ \mathit{else}\ Q$.

This use of the equational theory may be reminiscent of initial algebras. In an initial algebra, the principle of "no confusion" dictates that two elements are equal only if this is required by the corresponding equational theory. Similarly, $\mathit{if}\ M = N\ \mathit{then}\ P\ \mathit{else}\ Q$ reduces to P only if this is required by the equational theory, and reduces to Q otherwise. Initial algebras also obey the principle of "no junk," which says that all elements correspond to terms built exclusively from function symbols of the signature. In contrast, a fresh name need not equal any such term in the applied pi calculus.

2.3 Variants and Extensions

Several variants of the syntax of the applied pi calculus appear in the literature, and further variants may be considered. We discuss a few:

- In the conference paper, there are several sorts for channels: the sort Channel$\langle\tau\rangle$ is the sort of channels that convey messages of sort τ. The sort Channel without argument is more general, in the sense that all processes well sorted with Channel$\langle\tau\rangle$ are also well sorted with Channel. Having a single sort for channels simplifies some models, for instance, when all

public messages are sent on the same channel, even if they have different types. Moreover, by using Channel as only sort, we can encode an untyped version of the applied pi calculus. The tool ProVerif also uses the sort Channel without argument.

- In a more refined version of the sort system, we could allow names only in a distinguished set of sorts. For instance, we could consider a sort of Booleans, containing as only values the constants true and false. Such a sort would not contain names. Sorts without names would have to be treated with special care in proofs, since our proofs often use fresh names.

  On the other hand, letting all sorts contain names does not prevent modeling Booleans by a sort. For example, we can treat as false all terms of the sort different from true, including not only the constant false but also all names. Analogous treatments apply to other common data types.
- In the conference paper, channels in inputs and outputs are names or variables rather than any term. Allowing any term as channel yields a more general calculus and avoids some side conditions in theorems. It is also useful for some encodings (Abadi and Blanchet 2005b). Finally, it is in line with the syntax of ProVerif, where this design choice was adopted in order to simplify the untyped version of the calculus.

  Nevertheless, the sort system can restrict the terms that appear as channels: if no function symbol returns a result of sort Channel, then channels can be only names or variables.
- Function symbols can also be defined by rewrite rules instead of an equational theory. This approach is taken in ProVerif (Blanchet 2009): a destructor g is a partial function defined by rewrite rules $g(M_1, \ldots, M_l) \to M$; the destructor application $g(N_1, \ldots, N_l)$ fails when no rewrite rule applies, and this failure can be tested in the process calculus.

  A destructor $g : \tau_1 \times \cdots \times \tau_l \to \tau$ with rewrite rule $g(M_1, \ldots, M_l) \to M$ can be encoded in the applied pi calculus by function symbols $g : \tau_1 \times \cdots \times \tau_l \to \tau$ and $\mathrm{test}_g : \tau_1 \times \cdots \times \tau_l \to$ bool with the equations

$$g(M_1, \ldots, M_l) = M$$
$$\mathrm{test}_g(M_1, \ldots, M_l) = \mathsf{true}.$$

  The function test_g allows one to test whether $g(N_1, \ldots, N_l)$ is defined, by checking whether $\mathrm{test}_g(N_1, \ldots, N_l) = \mathsf{true}$ holds. (See Section 3 for examples of such test functions.) The function g may be applied even when its arguments are not instances of $(M_1, \ldots, M_l)$, thus yielding terms $g(N_1, \ldots, N_l)$ that do not exist in the calculus with rewrite rules. These "stuck" terms may be simulated with distinct fresh names in that variant of the calculus.

  Destructors are easy to implement in a tool. They also provide a built-in error-handling construct: the error handling is triggered when no rewrite rule applies. However, they complicate the semantics because they require a notion of evaluation of terms. Moreover, many useful functions can be defined by equations but not as destructors (for instance, encryption without redundancy, XOR, and modular exponentiation, which we use in the rest of this article). Therefore, ProVerif supports both destructors and equations (Blanchet et al. 2008). Thus, the language of ProVerif is a superset of the applied pi calculus as defined in this article (Blanchet 2016, Chapter 4), with the caveat that ProVerif does not support all equational theories and that it considers only plain processes.
- An extension that combines the applied pi calculus with ambients and with a built-in construct for evaluating messages as programs has also been studied (Blanchet and Aziz 2003). This extended calculus mixes many notions, so the corresponding proofs are complex. Considering a single notion at a time yields a simpler and more elegant calculus. Furthermore, although the applied pi calculus has few primitives, it supports various other constructs via

encodings; in particular, the message evaluation construct could be represented by defining an interpreter in the calculus.

- Our equational theories are closed under substitution of terms for names. This property yields a simple and uniform treatment of variables and names. An alternative definition, which may suffice, assumes only that equational theories are closed under one-to-one renaming and do not equate names. That definition makes it possible to define a function that tests whether a term is a name.

Some other variations concern the definition of the semantics:

- As in other papers (Blanchet et al. 2008; Liu 2011), we can handle the replication by a reduction step $!P \rightarrow P \mid !P$ instead of the structural equivalence rule $!P \equiv P \mid !P$. This modification prevents transforming $P \mid !P$ into $!P$, and thus simplifies some proofs.
- As Section 2.2 indicates, we can rewrite extended processes by pulling restrictions to the top, so that every closed extended process A becomes an extended process A° such that

$$A \equiv A^\circ = \nu \widetilde{n}.(\{^{\widetilde{M}}/_{\widetilde{x}}\} \mid P_1 \mid \ldots \mid P_l),$$

where $fv(P_1 \mid \ldots \mid P_l) = \emptyset$, $fv(\widetilde{M}) = \emptyset$, and $P_1, \ldots, P_l$ are replication, conditional, input, or output expressions. We can then modify the definitions of structural equivalence and internal reduction to act on processes in the form above. Structural equivalence says that the parallel composition $P_1 \mid \ldots \mid P_l$ is associative and commutative and that the names in $\widetilde{n}$ can be reordered. Internal reduction is the smallest relation on closed extended processes, closed by structural equivalence, such that

$$\begin{array}{rcll} E[\overline{N}\langle M\rangle.P \mid N'(x).Q] & \rightarrow & E[P \mid Q\{^{M}/_{x}\}]^\circ & \text{if } \Sigma \vdash N = N' \\ E[\mathit{if}\ M = N\ \mathit{then}\ P\ \mathit{else}\ Q] & \rightarrow & E[P]^\circ & \text{if } \Sigma \vdash M = N \\ E[\mathit{if}\ M = N\ \mathit{then}\ P\ \mathit{else}\ Q] & \rightarrow & E[Q]^\circ & \text{if } \Sigma \nvdash M = N \\ E[!P] & \rightarrow & E[P \mid !P]^\circ & \end{array}$$

for any evaluation context E. A similar idea appears in the intermediate applied pi calculus of Delaune et al. (2010) and Liu et al. (2011, 2012). There, all restrictions not under replication are pulled to the top of processes, over conditionals, inputs, outputs, and parallel compositions; the processes $P_1, \ldots, P_l$ may be $\mathbf{0}$; and channels are names or variables.
- Pushing the previous idea further, we can represent the extended process

$$A \equiv \nu \widetilde{n}.(\{^{\widetilde{M}}/_{\widetilde{x}}\} \mid P_1 \mid \ldots \mid P_l)$$

as a configuration $(\mathcal{N}, \sigma, \mathcal{P}) = (\{\widetilde{n}\}, \{^{\widetilde{M}}/_{\widetilde{x}}\}, \{P_1, \ldots, P_l\})$, where $\mathcal{N}$ is a set of names, σ is a substitution, and $\mathcal{P}$ is a multiset of processes. We can then define internal reduction on such configurations, without any structural equivalence. (Sets and multisets allow us to ignore the ordering of restrictions and parallel processes.) This idea is used in semantics of the calculus of ProVerif (Abadi and Blanchet 2005b; Allamigeon and Blanchet 2005; Blanchet 2009, 2016).

Semantics based on global configurations are closer to abstract machines. Such semantics simplify proofs, because they leave only few choices in reductions. They also make it easier to define further extensions of the calculus, such as tables and phases in ProVerif (Blanchet 2016). However, our compositional semantics is more convenient in order to model interactions between a process and a context. It is also closer to the traditional semantics of the pi calculus. The two kinds of semantics

are of course connected. In particular, Blanchet (2016, Chapter 4) formally relates the semantics of ProVerif based on configurations to our semantics.

3 BRIEF EXAMPLES

This section collects several examples, focusing on signatures, equations, and some simple processes. We start with pairs; this trivial example serves to introduce some notations and issues. We then discuss lists, cryptographic hash functions, encryption functions, digital signatures, and the XOR function (Menezes et al. 1996; Schneier 1996), as well as a form of multiplexing, which demonstrates the use of channels that are terms rather than names. Further examples appear in Sections 5 and 6. More examples, such as blind signatures (Kremer and Ryan 2005) and zero-knowledge proofs (Backes et al. 2008), have appeared in the literature since 2001.

Of course, at least some of these functions appear in most formalizations of cryptography and security protocols. In comparison with the spi calculus, the applied pi calculus permits a more uniform and versatile treatment of these functions, their variants, and their properties. Like the spi calculus, however, the applied pi calculus takes advantage of notations, concepts, and techniques from programming languages.

Pairs. Algebraic data types such as pairs, tuples, arrays, and lists occur in many examples. Encoding them in the pure pi calculus is not hard, but neither is representing them as primitive. For instance, the signature Σ may contain the binary function symbol pair and the unary function symbols fst and snd, with the abbreviation (M, N) for $\mathsf{pair}(M, N)$, and with the evident equations:

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

$$\mathsf{snd}((x, y)) = y. \tag{2}$$

(So the equational theory consists of these equations, and all the equations obtained by reflexivity, symmetry, transitivity, applications of function symbols, and substitutions of terms for variables.) These function symbols may, for instance, be sorted as follows:

$$\begin{aligned} \mathsf{pair} &: \mathsf{Data} \times \mathsf{Data} \to \mathsf{Data} \\ \mathsf{fst} &: \mathsf{Data} \to \mathsf{Data} \\ \mathsf{snd} &: \mathsf{Data} \to \mathsf{Data} \end{aligned}$$

We may use the test $(\mathsf{fst}(M), \mathsf{snd}(M)) = M$ to check that M is a pair before using the values of $\mathsf{fst}(M)$ and $\mathsf{snd}(M)$. Alternatively, we may add a Boolean function is_pair that recognizes pairs, defined by the equation

$$\mathsf{is\_pair}((x, y)) = \mathsf{true}.$$

With this equation, the conditional *if* $\mathsf{is\_pair}(M) = \mathsf{true}$ *then* P *else* Q runs P if M is a pair and Q otherwise. Using pairs, we may, for instance, define the process:

$$\nu s.\left(\overline{a}\langle(M, s)\rangle \mid a(z).\mathit{if}\ \mathsf{snd}(z) = s\ \mathit{then}\ \overline{b}\langle\mathsf{fst}(z)\rangle\right).$$

One of its components sends a pair consisting of a term M and a fresh name s on a channel a. The other receives a message on a and, if its second component is s, forwards the first component on a channel b. Thus, we may say that s serves as a capability (or password) for the forwarding. However, this capability is not protected from eavesdroppers when it travels on a. Any other process can listen on a and can apply snd to the message received, thus learning s. We can represent such an attacker within the calculus, for example, by the following process:

$$a(z).\overline{a}\langle(N, \mathsf{snd}(z))\rangle,$$

which may receive (M, s) on a and send (N, s) on a. Composing this attacker in parallel with the process, we may obtain N instead of M on b.

Such attacks can be thwarted by the use of restricted channel names, as in the process

$$va.vs.\left(\overline{a}\langle(M,s)\rangle \mid a(z).\mathit{if}\ \mathsf{snd}(z) = s\ \mathit{then}\ \overline{b}\langle\mathsf{fst}(z)\rangle\right).$$

Alternatively, they can be thwarted by the use of cryptography, as discussed below.

Lists. We may treat lists similarly, with the following function symbols and corresponding sorts:

$$\begin{aligned}\mathsf{nil} &: \mathsf{List}\\ \mathsf{cons} &: \mathsf{Data} \times \mathsf{List} \to \mathsf{List}\\ \mathsf{hd} &: \mathsf{List} \to \mathsf{Data}\\ \mathsf{tl} &: \mathsf{List} \to \mathsf{List}\end{aligned}$$

The constant nil is the empty list; $\mathsf{cons}(x, y)$ represents the concatenation of the element x at the beginning of the list y, and we write it with infix notation as $x :: y$, where the symbol :: associates to the right, and hd and tl are head and tail functions with the equations

$$\mathsf{hd}(x :: y) = x \qquad \mathsf{tl}(x :: y) = y. \tag{3}$$

Further, we write $M +\!\!+ N$ for the concatenation of an element N at the end of a list M, where the function $+\!\!+ : \mathsf{List} \times \mathsf{Data} \to \mathsf{List}$ associates to the left, and satisfies the equations

$$\mathsf{nil} +\!\!+ x = x :: \mathsf{nil} \qquad (x :: y) +\!\!+ z = x :: (y +\!\!+ z). \tag{4}$$

Cryptographic Hash Functions. We represent a cryptographic hash function as a unary function symbol h with no equations. The absence of an inverse for h models the one-wayness of h. The fact that $\mathsf{h}(M) = \mathsf{h}(N)$ only when $M = N$ models that h is collision-free.

Modifying our first example, we may now write the process:

$$vs.\left(\overline{a}\langle(M, \mathsf{h}((s, M)))\rangle \mid a(x).\mathit{if}\ \mathsf{h}((s, \mathsf{fst}(x))) = \mathsf{snd}(x)\ \mathit{then}\ \overline{b}\langle\mathsf{fst}(x)\rangle\right).$$

Here the value M is authenticated by pairing it with the fresh name s and then hashing the pair. Although $(M, \mathsf{h}((s, M)))$ travels on the public channel a, no other process can extract s from this message or produce $(N, \mathsf{h}((s, N)))$ for some other N using the available functions. Therefore, we may reason that this process will forward only the intended term M on channel b.

This example is a typical cryptographic application of hash functions. In light of the practical importance of those applications, our treatment of hash functions is attractively straightforward. Still, we may question whether our formal model of these functions is not too strong and simplistic in comparison with the properties of actual implementations based on algorithms such as SHA. In Section 6, we consider a somewhat weaker, subtler model for hash functions.

Symmetric Encryption. In order to model symmetric cryptography (i.e., shared-key cryptography), we take binary function symbols enc and dec for encryption and decryption, respectively, with the equation

$$\mathsf{dec}(\mathsf{enc}(x, y), y) = x.$$

Here x represents the plaintext and y the key. We often use fresh names as keys in examples; for instance, the (useless) process

$$vk.\overline{a}\langle\mathsf{enc}(M, k)\rangle$$

sends the term M encrypted under a fresh key k.

In applications of encryption, it is frequent to assume that each encrypted message comes with sufficient redundancy so that decryption with the "wrong" key is evident. Accordingly, we can test whether the decryption of M with the key k succeeds by testing whether $\mathsf{enc}(\mathsf{dec}(M,k),k) = M$. Alternatively, we could also add a test function $\mathsf{test}_{\mathsf{dec}}$ with the equation

$$\mathsf{test}_{\mathsf{dec}}(\mathsf{enc}(x,y),y) = \mathsf{true}.$$

Provided that we check that decryption succeeds before using the decrypted message, this model of encryption basically yields the spi calculus (Abadi and Gordon 1999).

On the other hand, in modern cryptology, such redundancy is not usually viewed as part of the encryption function proper, but rather an addition. The redundancy can be implemented with message authentication codes. We can model an encryption scheme without redundancy with the following two equations:

$$\mathsf{dec}(\mathsf{enc}(x,y),y) = x$$
$$\mathsf{enc}(\mathsf{dec}(z,y),y) = z.$$

These equations model that decryption is the inverse bijection of encryption, a property that is typically satisfied by block ciphers.

Asymmetric Encryption. It is only slightly harder to model asymmetric (public-key) cryptography, where the keys for encryption and decryption are different. We introduce two new unary function symbols pk and sk for generating public and private keys from a seed, and the equation

$$\mathsf{dec}(\mathsf{enc}(x,\mathsf{pk}(y)),\mathsf{sk}(y)) = x.$$

We may now write the process

$$\nu s.\left(\overline{a}\langle \mathsf{pk}(s)\rangle \mid b(x).\overline{c}\langle \mathsf{dec}(x,\mathsf{sk}(s))\rangle\right).$$

The first component publishes the public key $\mathsf{pk}(s)$ by sending it on a. The second receives a message on b, uses the corresponding private key $\mathsf{sk}(s)$ to decrypt it, and forwards the resulting plaintext on c. As this example indicates, we essentially view name restriction (νs) 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.

Some encryption schemes have additional properties. In particular, enc and dec may be the same function. This property matters in implementations, and sometimes permits attacks. Moreover, certain encryptions and decryptions commute in some schemes. For example, we have $\mathsf{dec}(\mathsf{enc}(x,y),z) = \mathsf{enc}(\mathsf{dec}(x,z),y)$ if the encryptions and decryptions are performed using RSA with the same modulus. The treatment of such properties is left open in the spi calculus (Abadi and Gordon 1999). In contrast, it is easy to express the properties in the applied pi calculus and to study the protocols and attacks that depend on them.

Nondeterministic ("Probabilistic") Encryption. Going further, we may add a third argument to enc, so that the encryption of a plaintext with a key is not unique. This nondeterminism is an essential property of probabilistic encryption (Goldwasser and Micali 1984). The equation for decryption becomes

$$\mathsf{dec}(\mathsf{enc}(x,\mathsf{pk}(y),z),\mathsf{sk}(y)) = x.$$

With this variant, we may write the process

$$a(x).\left(\nu m.\overline{b}\langle \mathsf{enc}(M,x,m)\rangle \mid \nu n.\overline{c}\langle \mathsf{enc}(N,x,n)\rangle\right),$$

which receives a message x and uses it as an encryption key for two messages, $\mathsf{enc}(M,x,m)$ and $\mathsf{enc}(N,x,n)$. An observer who does not have the corresponding decryption key cannot tell whether

the underlying plaintexts M and N are identical by comparing the ciphertexts, because the ciphertexts rely on different fresh names m and n. Moreover, even if the observer learns x, M, and N (but not the decryption key), it cannot verify that the messages contain M and N because it does not know m and n.

Public-Key Digital Signatures. Like public-key encryption schemes, digital signature schemes rely on pairs of public and private keys. In each pair, the private key serves for computing signatures and the public key for verifying those signatures. In order to model key generation, we use again the two unary function symbols pk and sk for generating public and private keys from a seed. For signatures and their verification, we use a new binary function symbol sign, a ternary function symbol check, and a constant symbol ok, with the equation

$$\mathsf{check}(x, \mathsf{sign}(x, \mathsf{sk}(y)), \mathsf{pk}(y)) = \mathsf{ok}.$$

(Several variants are possible.)

Modifying once more our first example, we may now write the process

$$\begin{aligned}&\left(\nu s.\{^{\mathsf{pk}(s)}/_{y}\} \mid \overline{a}\langle(M, \mathsf{sign}(M, \mathsf{sk}(s)))\rangle\right) \mid \\ &a(x).\mathit{if}\ \mathsf{check}(\mathsf{fst}(x), \mathsf{snd}(x), y) = \mathsf{ok}\ \mathit{then}\ \overline{b}\langle\mathsf{fst}(x)\rangle.\end{aligned}$$

Here the value M is signed using the private key $\mathsf{sk}(s)$. Although M and its signature travel on the public channel a, no other process can produce N and its signature for some other N. Therefore, again, we may reason that only the intended term M will be forwarded on channel b. This property holds despite the publication of $\mathsf{pk}(s)$ (but not $\mathsf{sk}(s)$), which is represented by the active substitution that maps y to $\mathsf{pk}(s)$. Despite the restriction on s, processes outside the restriction can use $\mathsf{pk}(s)$ through y. In particular, y refers to $\mathsf{pk}(s)$ in the process that checks the signature on M.

XOR. We may model the XOR function, some of its uses in cryptography, and some of the protocol flaws connected with it. Some of these flaws (e.g., Ryan and Schneider (1998)) stem from the intrinsic equational properties of XOR, such as associativity, commutativity, the existence of a neutral element, and the cancellation property that we may write as follows:

$$\begin{aligned}\mathsf{xor}(\mathsf{xor}(x, y), z) &= \mathsf{xor}(x, \mathsf{xor}(y, z))\\ \mathsf{xor}(x, y) &= \mathsf{xor}(y, x)\\ \mathsf{xor}(x, 0) &= x\\ \mathsf{xor}(x, x) &= 0.\end{aligned}$$

Others arise because of the interactions between XOR and other operations (e.g., Core SDI S.A. (1998); Stubblebine and Gligor (1992)). For example, CRCs (cyclic redundancy checks) can be poor proofs of integrity, partly because of the equation

$$\mathsf{crc}(\mathsf{xor}(x, y)) = \mathsf{xor}(\mathsf{crc}(x), \mathsf{crc}(y)).$$

Multiplexing. Finally, we illustrate a possible usage of channels that are not names. Consider, for instance, a pairing function for building channels $\mathsf{pair} : \mathsf{Data} \times \mathsf{Port} \to \mathsf{Channel}$ with its associated projections $\mathsf{fst} : \mathsf{Channel} \to \mathsf{Data}$ and $\mathsf{snd} : \mathsf{Channel} \to \mathsf{Port}$, and Equations (1) and (2) from our first example. We may use this function for multiplexing as follows:

$$\begin{aligned}&\nu s.(\overline{\mathsf{pair}(s, \mathsf{port}_1)}\langle M_1\rangle \mid \overline{\mathsf{pair}(s, \mathsf{port}_2)}\langle M_2\rangle\\ &\quad \mid \mathsf{pair}(s, \mathsf{port}_1)(x_1) \mid \mathsf{pair}(s, \mathsf{port}_2)(x_2)).\end{aligned}$$

In this process, the first output can be received only by the first input, and the second output can be received only by the second input.

4 EQUIVALENCES AND PROOF TECHNIQUES

In examples, we frequently argue that two given processes cannot be distinguished by any context, that is, that the processes are observationally equivalent. The spi calculus developed the idea that the context represents an active attacker, and equivalences capture authenticity and secrecy properties in the presence of the attacker. More broadly, a wide variety of security properties can be expressed as equivalences.

In this section, we define observational equivalence for the applied pi calculus. We also introduce a notion of static equivalence for frames, a labeled semantics for processes, and a labeled equivalence relation. We prove that labeled equivalence and observational equivalence coincide, obtaining a convenient proof technique for observational equivalence.

4.1 Observational Equivalence

We write $A \Downarrow a$ when A can send a message on name a, that is, when $A \rightarrow^* \equiv E[\overline{a}\langle M\rangle.P]$ for some evaluation context $E[\_]$ that does not bind a.

Definition 4.1. An *observational bisimulation* is a symmetric relation $\mathcal{R}$ between closed extended processes with the same domain such that $A\ \mathcal{R}\ B$ implies:

(1) if $A \Downarrow a$, then $B \Downarrow a$;
(2) if $A \rightarrow^* A'$ and A' is closed, then $B \rightarrow^* B'$ and $A'\ \mathcal{R}\ B'$ for some B';
(3) $E[A]\ \mathcal{R}\ E[B]$ for all closing evaluation contexts $E[\_]$.

Observational equivalence ($\approx$) is the largest such relation.

For example, when h is a unary function symbol with no equations, we obtain that $\nu s.\overline{a}\langle s\rangle \approx \nu s.\overline{a}\langle \mathsf{h}(s)\rangle$.

These definitions are standard in the pi calculus, where $\Downarrow a$ is called a *barb* on a, and where $\approx$ is one of the two usual notions of weak barbed bisimulation congruence. (See Section 4.5 and Fournet and Gonthier (1998) for a detailed discussion.) In the applied pi calculus, one could also define barbs on arbitrary terms, not just on names; we do not need that generalization for our purposes. The set of closing evaluation contexts for A depends only on A's domain; hence, in Definition 4.1, A and B have the same closing evaluation contexts. In Definition 4.1(2), since $\mathcal{R}$ is a relation between closed extended processes, we require that A' also be closed. Being closed is not preserved by all reductions, since structural equivalence may introduce free unused variables. For instance, we have $\mathbf{0} \equiv \nu x.\{^{y}/_{x}\}$ by Alias and $\{^{M}/_{x}\} \equiv \{^{\mathsf{fst}((M,y))}/_{x}\}$ by Rewrite using the equation $\mathsf{fst}((x,y)) = x$.

Although observational equivalence is undecidable in general, various tools support certain automatic proofs of observational equivalence and other equivalence relations, in the applied pi calculus and related languages (e.g., Baudet (2005), Blanchet et al. (2008), Chadha et al. (2012), and Cheval et al. (2013)).

4.2 Static Equivalence

Two substitutions may be seen as equivalent when they behave equivalently when applied to terms. We write $\approx_s$ for this notion of equivalence and call it static equivalence. In the presence of the "new" construct, defining $\approx_s$ is somewhat delicate and interesting. For instance, consider two functions f and g with no equations (intuitively, two independent hash functions), and the three

frames

$$\varphi_0 \stackrel{\text{def}}{=} \nu k.\{^{k}/_{x}\} \mid \nu s.\{^{s}/_{y}\}$$
$$\varphi_1 \stackrel{\text{def}}{=} \nu k.\{^{\mathsf{f}(k)}/_{x}, ^{\mathsf{g}(k)}/_{y}\}$$
$$\varphi_2 \stackrel{\text{def}}{=} \nu k.\{^{k}/_{x}, ^{\mathsf{f}(k)}/_{y}\}.$$

In φ_0, the variables x and y are mapped to two unrelated values that are different from any value that the context may build (since k and s are new). These properties also hold, but more subtly, for φ_1; although $\mathsf{f}(k)$ and $\mathsf{g}(k)$ are based on the same underlying fresh name, they look unrelated. (Analogously, it is common to derive apparently unrelated keys by hashing from a single underlying secret, as in SSL and TLS (Dierks and Rescorla 2008; Freier et al. 1996).) Hence, a context that obtains the values for x and y cannot distinguish φ_0 and φ_1. On the other hand, the context can discriminate φ_2 by testing the predicate $\mathsf{f}(x) = y$. Therefore, we would like to define static equivalence so that $\varphi_0 \approx_s \varphi_1 \not\approx_s \varphi_2$.

This example relies on a concept of equality of terms in a frame, which the following definition captures.

Definition 4.2. Two terms M and N are equal in the frame φ, written $(M = N)\varphi$, if and only if $fv(M) \cup fv(N) \subseteq dom(\varphi)$, $\varphi \equiv \nu\widetilde{n}.\sigma$, $M\sigma = N\sigma$, and $\{\widetilde{n}\} \cap (fn(M) \cup fn(N)) = \emptyset$ for some names $\widetilde{n}$ and substitution σ.

In Definition 4.2, the equality $M\sigma = N\sigma$ is independent of the representative $\nu\widetilde{n}.\sigma$ chosen for the frame φ such that $\varphi \equiv \nu\widetilde{n}.\sigma$ and $\{\widetilde{n}\} \cap (fn(M) \cup fn(N)) = \emptyset$. (Lemma D.1 in Appendix D establishes this property.)

Definition 4.3. Two closed frames φ and ψ are *statically equivalent*, written $\varphi \approx_s \psi$, when $dom(\varphi) = dom(\psi)$ and when, for all terms M and N, we have $(M = N)\varphi$ if and only if $(M = N)\psi$.

Two closed extended processes are statically equivalent, written $A \approx_s B$, when their frames are statically equivalent.

For instance, in our example, we have $(\mathsf{f}(x) = y)\varphi_2$ but not $(\mathsf{f}(x) = y)\varphi_1$, and hence, $\varphi_1 \not\approx_s \varphi_2$.

Depending on Σ, static equivalence can be quite hard to check, but at least it does not depend on the dynamics of processes. Some simplifications are possible in common cases, in particular when terms can be put in normal forms (e.g., in the proof of Theorems 6.1 and 6.3). Decision procedures exist for static equivalence in large classes of equational theories (Abadi and Cortier 2006), some implemented in tools (Baudet et al. 2009a; Ciobâcă et al. 2012).

The next lemma establishes closure properties of static equivalence: it shows that static equivalence is invariant by structural equivalence and reduction, and closed by application of closing evaluation contexts. Its proof appears in Appendix A.

Lemma 4.4. *Let A and B be closed extended processes. If $A \equiv B$ or $A \to B$, then $A \approx_s B$. If $A \approx_s B$, then $E[A] \approx_s E[B]$ for all closing evaluation contexts $E[\_]$.*

As the next two lemmas demonstrate, static equivalence coincides with observational equivalence on frames, but is coarser on extended processes.

Lemma 4.5. *Observational equivalence and static equivalence coincide on frames.*

This lemma is an immediate corollary of Theorem 4.8 below. (See Corollary C.14 in Appendix C.3.)

Lemma 4.6. *Observational equivalence is strictly finer than static equivalence on extended processes: $\approx \subset \approx_s$.*

To see that observational equivalence implies static equivalence, note that if A and B are observationally equivalent, then $A \mid C$ and $B \mid C$ have the same barbs for every C with $fv(C) \subseteq dom(A)$, and that they are statically equivalent when $A \mid C$ and $B \mid C$ have the same barb $\Downarrow a$ for every C of the special form *if* $M = N$ *then* $\overline{a}\langle n\rangle$, where a does not occur in A or B and $fv(C) \subseteq dom(A)$. (See Lemma C.9 in Appendix C.3.) The converse does not hold, as the following counterexample shows: letting $A = \overline{a}\langle n\rangle$ and $B = \overline{b}\langle n\rangle$, we have $A \not\approx B$, but $A \approx_s B$ because $\varphi(A) = \varphi(B) = \mathbf{0}$.

4.3 Labeled Operational Semantics and Equivalence

A labeled operational semantics extends the chemical semantics of Section 2.2, enabling us to reason about processes that interact with their context while keeping it implicit. The labeled semantics defines a relation $A \xrightarrow{\alpha} A'$, where α is a label of one of the following forms:

- A label $N(M)$, which corresponds to an input of M on N
- A label $\nu x.\overline{N}\langle x\rangle$, where x is a variable that must not occur in N, which corresponds to an output of x on N

The variable x is bound in the label $\nu x.\overline{N}\langle x\rangle$, so we define the bound variables of labels by $bv(N(M)) \stackrel{\text{def}}{=} \emptyset$ and $bv(\nu x.\overline{N}\langle x\rangle) \stackrel{\text{def}}{=} \{x\}$. The free variables of labels are defined by $fv(N(M)) \stackrel{\text{def}}{=} fv(N) \cup fv(M)$ and $fv(\nu x.\overline{N}\langle x\rangle) \stackrel{\text{def}}{=} fv(N)$ (since x does not occur in N in the latter label).

In addition to the rules for structural equivalence and reduction of Section 2, we adopt the following rules:

$$\textsc{In} \qquad N(x).P \xrightarrow{N(M)} P\{^{M}/_{x}\}$$

$$\textsc{Out-Var} \qquad \frac{x \notin fv(\overline{N}\langle M\rangle.P)}{\overline{N}\langle M\rangle.P \xrightarrow{\nu x.\overline{N}\langle x\rangle} P \mid \{^{M}/_{x}\}}$$

$$\textsc{Scope} \qquad \frac{A \xrightarrow{\alpha} A' \qquad u \text{ does not occur in } \alpha}{\nu u.A \xrightarrow{\alpha} \nu u.A'}$$

$$\textsc{Par} \qquad \frac{A \xrightarrow{\alpha} A' \qquad bv(\alpha) \cap fv(B) = \emptyset}{A \mid B \xrightarrow{\alpha} A' \mid B}$$

$$\textsc{Struct} \qquad \frac{A \equiv B \qquad B \xrightarrow{\alpha} B' \qquad B' \equiv A'}{A \xrightarrow{\alpha} A'}.$$

According to In, a term M may be input. On the other hand, Out-Var permits output for terms "by reference": a fresh variable is associated with the term in question and output.

For example, using the signature and equations for symmetric encryption, and the new constant symbol oops, we have the sequence of transitions of Figure 3. The first two transitions do not directly reveal the term M. However, they give enough information to the environment to compute M as $\mathrm{dec}(x, y)$ and to input it in the third transition.

The labeled operational semantics leads to an equivalence relation:

Definition 4.7. A *labeled bisimulation* is a symmetric relation $\mathcal{R}$ on closed extended processes such that $A \mathcal{R} B$ implies:

(1) $A \approx_s B$;
(2) if $A \to A'$ and A' is closed, then $B \to^* B'$ and $A' \mathcal{R} B'$ for some B';

$$
\begin{array}{rl}
 & \nu k.\overline{a}\langle \mathrm{enc}(M,k)\rangle.\overline{a}\langle k\rangle.a(z).\mathit{if}\ z = M\ \mathit{then}\ \overline{c}\langle\,\mathrm{oops}\,!\,\rangle \\
\xrightarrow{\nu x.\overline{a}\langle x\rangle} & \nu k.\left(\{^{\mathrm{enc}(M,k)}/_{x}\} \mid \overline{a}\langle k\rangle.a(z).\mathit{if}\ z = M\ \mathit{then}\ \overline{c}\langle\,\mathrm{oops}\,!\,\rangle\right) \\
\xrightarrow{\nu y.\overline{a}\langle y\rangle} & \nu k.\left(\{^{\mathrm{enc}(M,k)}/_{x}\} \mid \{^{k}/_{y}\} \mid a(z).\mathit{if}\ z = M\ \mathit{then}\ \overline{c}\langle\,\mathrm{oops}\,!\,\rangle\right) \\
\xrightarrow{a(\mathrm{dec}(x,y))} & \nu k.\left(\{^{\mathrm{enc}(M,k)}/_{x}\} \mid \{^{k}/_{y}\} \mid \mathit{if}\ \mathrm{dec}(x,y) = M\ \mathit{then}\ \overline{c}\langle\,\mathrm{oops}\,!\,\rangle\right) \\
\rightarrow & \nu k.\left(\{^{\mathrm{enc}(M,k)}/_{x}\} \mid \{^{k}/_{y}\}\right) \mid \overline{c}\langle\,\mathrm{oops}\,!\,\rangle
\end{array}
$$

Fig. 3. Example transitions.

(3) if $A \xrightarrow{\alpha} A'$, A' is closed, and $fv(\alpha) \subseteq dom(A)$, then $B \rightarrow^* \xrightarrow{\alpha} \rightarrow^* B'$ and $A' \mathcal{R} B'$ for some B'.

Labeled bisimilarity ($\approx_l$) is the largest such relation.

Conditions 2 and 3 are standard; condition 1, which requires that bisimilar processes be statically equivalent, is necessary, for example, in order to distinguish the frames φ_0 and φ_2 of Section 4.2. As in Definition 4.1, we explicitly require that A' be closed and $fv(\alpha) \subseteq dom(A)$ in order to exclude transitions that introduce free unused variables.

Our main result is that this relation coincides with observational equivalence. Although such results are fairly common in process calculi, they are important and nontrivial.

Theorem 4.8. *Observational equivalence is labeled bisimilarity:* $\approx = \approx_l$.

The proof of this theorem is outlined in Section 4.5 and completed in the appendix.

The theorem implies that $\approx_l$ is closed by application of closing evaluation contexts. However, unlike the definition of $\approx$, the definition of $\approx_l$ does not include a condition about contexts. It therefore permits simpler proofs.

In addition, labeled bisimilarity can probably be established via standard "bisimulation up to context" techniques (Sangiorgi 1998), which enable useful on-the-fly simplifications in frames after output steps. We do not develop the theory of "up to context" techniques, since we do not use them in this article.

The following lemmas provide methods for simplifying frames:

Lemma 4.9 (Alias elimination). *Let A and B be closed extended processes, M be a term such that $fv(M) \subseteq dom(A)$, and x be a variable such that $x \notin dom(A)$. We have $A \approx_l B$ if and only if*

$$\{^{M}/_{x}\} \mid A \approx_l \{^{M}/_{x}\} \mid B.$$

Proof. Both directions follow from context closure of $\approx_l$ for the contexts $\{^{M}/_{x}\} \mid \_$ and $\nu x.\_$, respectively. In the converse direction, since x is not free in A or B, we have $A \equiv \nu x.(\{^{M}/_{x}\} \mid A)$, $\nu x.(\{^{M}/_{x}\} \mid A) \approx_l \nu x.(\{^{M}/_{x}\} \mid B)$, and $\nu x.(\{^{M}/_{x}\} \mid B) \equiv B$, and hence $A \approx_l B$. □

Lemma 4.10 (Name disclosure). *Let A and B be closed extended processes and x be a variable such that $x \notin dom(A)$. We have $A \approx_l B$ if and only if*

$$\nu n.(\{^{n}/_{x}\} \mid A) \approx_l \nu n.(\{^{n}/_{x}\} \mid B).$$

Proof. The direct implication follows from context closure of $\approx_l$. Conversely, we show that the relation $\mathcal{R}$ defined by $A \mathcal{R} B$ if and only if A and B are closed extended processes and $\nu n.(\{^{n}/_{x}\} \mid A) \approx_l \nu n.(\{^{n}/_{x}\} \mid B)$ for some $x \notin dom(A)$ is a labeled bisimulation. This proof is detailed in Appendix D. □

In Lemma 4.9, the substitution $\{^{M}/_{x}\}$ can affect only the context, since A and B are closed. However, the lemma implies that the substitution does not give or mask any information about A and B to the context. In Lemma 4.10, the restriction on n and the substitution $\{^{n}/_{x}\}$ mean that the context can access n only indirectly, through the free variable x. Intuitively, the lemma says that indirect access is equivalent to direct access in this case.

Our labeled operational semantics contrasts with a more naive semantics carried over from the pure pi calculus, with output labels of the form $\nu\widetilde{u}.\overline{N}\langle M\rangle$ and rules that permit direct output of any term, such as

$$\textsc{Out-Term} \qquad \overline{N}\langle M\rangle.P \xrightarrow{\overline{N}\langle M\rangle} P$$

$$\textsc{Open} \qquad \frac{A \xrightarrow{\nu\widetilde{u}.\overline{N}\langle M\rangle} A' \qquad v \in fv(M) \cup fn(M) \setminus (fv(N) \cup fn(N) \cup \{\widetilde{u}\})}{\nu v.A \xrightarrow{\nu v,\widetilde{u}.\overline{N}\langle M\rangle} A'}.$$

These rules lead to a different, finer equivalence relation, which, for example, would distinguish $\nu k, s.\overline{a}\langle(k, s)\rangle$ and $\nu k.\overline{a}\langle(\mathsf{f}(k), \mathsf{g}(k))\rangle$. This equivalence relation is often inadequate in applications (as in Abadi and Gordon (1999, Section 5.2.1)), hence our definitions.

We have also studied intermediately liberal rules for output, which permit direct output of certain terms. In particular, the rules of the conference paper permit direct output of channel names. That feature implies that it is not necessary to export variables of channel types; as Section 4.5 explains, this property is needed for Theorem 4.8 for those rules. That feature makes little sense in the present calculus, in which arbitrary terms may be used as channels, so we abandon it in the rules above. Nevertheless, certain rules with more explicit labels can still be helpful. We explain those rules next.

4.4 Making the Output Labels More Explicit

In the labeled operational semantics of Section 4.3, the labels for outputs do not reveal anything about the terms being output: those terms are represented by fresh variables. Often, however, more explicit labels can be convenient in reasoning about protocols, and they do not cause harm as long as they only make explicit information that is immediately available to the environment. For instance, for the process $\nu k.\overline{a}\langle(\mathsf{Header}, \mathsf{enc}(M, k))\rangle$, the label $\nu y.\overline{a}\langle(\mathsf{Header}, y)\rangle$ is more informative than $\nu x.\overline{a}\langle x\rangle$. In this example, the environment could anyway observe that x is a pair such that $\mathsf{fst}(x) = \mathsf{Header}$ and use $\mathsf{snd}(x)$ for y. More generally, we rely on the following definition to characterize the information that the environment can derive.

Definition 4.11. Variables $\widetilde{x}$ *resolve to* $\widetilde{M}$ *in* A if and only if $A \equiv \{^{\widetilde{M}}/_{\widetilde{x}}\} \mid \nu\widetilde{x}.A$. They are *solvable* in A if and only if they resolve to some terms in A.

Hence, when variables $\widetilde{x}$ resolve to terms $\widetilde{M}$ in A, they are in $dom(A)$ and we can erase the restriction of $\nu\widetilde{x}.A$ by applying the context $\{^{\widetilde{M}}/_{\widetilde{x}}\} \mid \_$ and by structural equivalence. Intuitively, A does not reveal more information than $\nu\widetilde{x}.A$, because the environment can build the terms $\widetilde{M}$ and use them instead of $\widetilde{x}$.

In general, when variables $\widetilde{x}$ are in $dom(A)$, there exist $\widetilde{n}$, $\widetilde{M}$, and A' such that $A \equiv \nu\widetilde{n}.(\{^{\widetilde{M}}/_{\widetilde{x}}\} \mid A')$. If variables $\widetilde{x}$ resolve to $\widetilde{M}$ in A, then $\widetilde{n}$ can be chosen empty, so that the terms $\widetilde{M}$ are not under restrictions. The following lemma provides two reformulations of Definition 4.11, including a converse to this observation. Its proof appears in Appendix E.

Lemma 4.12. *The following three properties are equivalent:*

(1) *The variables* $\widetilde{x}$ *resolve to* $\widetilde{M}$ *in* A.
(2) *There exists* A' *such that* $A \equiv \{^{\widetilde{M}}/_{\widetilde{x}}\} \mid A'$.
(3) $(\widetilde{x} = \widetilde{M})\varphi(A)$ *and the substitution* $\{^{\widetilde{M}}/_{\widetilde{x}}\}$ *is cycle-free.*

For example, using pairs and symmetric encryption, we let

$$\varphi \stackrel{\text{def}}{=} \nu k.\{^{M}/_{x}, {}^{\mathsf{enc}(x,k)}/_{y}, {}^{(\mathsf{Header}, y)}/_{z}\}.$$

The variable y resolves to $\mathsf{snd}(z)$ in φ, since

$$\varphi \equiv \{^{\mathsf{snd}(z)}/_{y}\} \mid \nu k.\{^{M}/_{x}, {}^{(\mathsf{Header}, \mathsf{enc}(x,k))}/_{z}\},$$

and z resolves to (Header, y) in φ, since

$$\varphi \equiv \{^{(\mathsf{Header}, y)}/_{z}\} \mid \nu k.\{^{M}/_{x}, {}^{\mathsf{enc}(x,k)}/_{y}\}.$$

In contrast, x is not always solvable in φ (for instance, when M is k).

A second lemma shows that Definition 4.11 is robust in the sense that it is preserved by static equivalence, so a fortiori by labeled bisimilarity:

Lemma 4.13. *If* $A \approx_s B$ *and* $\widetilde{x}$ *resolve to* $\widetilde{M}$ *in* A, *then* $\widetilde{x}$ *resolve to* $\widetilde{M}$ *in* B.

Proof. Static equivalence preserves property 3 of Lemma 4.12, so we conclude by Lemma 4.12. □

We introduce an alternative semantics in which the rules permit composite terms in output labels but require that every restricted variable that is exported be solvable. In this semantics, the label α in the relation $A \xrightarrow{\alpha} A'$ ranges over the same input labels $N(M)$ as in Section 4.3, and over generalized output labels of the form $\nu\widetilde{x}.\overline{N}\langle M\rangle$, where $\{\widetilde{x}\} \subseteq fv(M) \setminus fv(N)$. The label $\nu\widetilde{x}.\overline{N}\langle M\rangle$ corresponds to an output of M on N that reveals the variables $\widetilde{x}$. We retain the rules for structural equivalence and reduction, and rules In, Par, and Struct of Section 4.3. We also keep rule Scope, but only for labels with no extrusion, that is, for labels $N(M)$ and $\overline{N}\langle M\rangle$. This restriction is necessary because variables may not remain solvable after the application of a context $\nu u.\_$. As a replacement for the rule Out-Var, we use the rule Out-Term discussed in Section 4.3 and

$$\textsc{Open-Var}\ \frac{A \xrightarrow{\overline{N}\langle M\rangle} A' \qquad \{\widetilde{x}\} \subseteq fv(M) \setminus fv(N) \qquad \widetilde{x} \text{ solvable in } \{^{M}/_{z}\} \mid A' \text{ for some } z \notin fv(A') \cup \{\widetilde{x}\}}{\nu\widetilde{x}.A \xrightarrow{\nu\widetilde{x}.\overline{N}\langle M\rangle} A'}.$$

These rules are more liberal than those of Section 4.3. For instance, consider $A_1 = \nu k.\overline{a}\langle(\mathsf{f}(k), \mathsf{g}(k))\rangle$ and $A_2 = \nu k.\overline{a}\langle(k, \mathsf{f}(k))\rangle$. With the rules of Section 4.3, we have

$$A_i \xrightarrow{\nu z.\overline{a}\langle z\rangle} \nu x, y.(\{^{(x,y)}/_{z}\} \mid \varphi_i),$$

where φ_i is as in Section 4.2. With the new rules, we also have

$$A_i \xrightarrow{\nu x, y.\overline{a}\langle(x,y)\rangle} \varphi_i. \tag{5}$$

Indeed, $A_i \equiv \nu x, y.(\overline{a}\langle(x, y)\rangle \mid \varphi_i)$ and the variables x, y are solvable in $\{^{(x,y)}/_{z}\} \mid \varphi_i$ because $\{^{(x,y}/_{z}\} \mid \varphi_i \equiv \{^{\mathsf{fst}(z)}/_{x}, {}^{\mathsf{snd}(z)}/_{y}\} \mid \nu x, y.(\{^{(x,y)}/_{z}\} \mid \varphi_i)$, so we derive

$$\overline{a}\langle(x, y)\rangle \xrightarrow{\overline{a}\langle(x,y)\rangle} \mathbf{0} \qquad \text{by Out-Term}$$

$$\overline{a}\langle(x, y)\rangle \mid \varphi_i \xrightarrow{\overline{a}\langle(x,y)\rangle} \varphi_i \qquad \text{by Par and Struct}$$

$$\nu x, y.(\overline{a}\langle(x, y)\rangle \mid \varphi_i) \xrightarrow{\nu x, y.\overline{a}\langle(x,y)\rangle} \varphi_i \qquad \text{by Open-Var}$$

$$A_i \xrightarrow{\nu x, y.\overline{a}\langle(x,y)\rangle} \varphi_i \qquad \text{by Struct.}$$

The transition in Equation (5) is the most informative for A_1 since x and y behave like fresh, independent values in φ_1. For A_2, we also have the more informative transition:

$$A_2 \xrightarrow{\nu x.\overline{a}\langle(x,\mathrm{f}(x))\rangle} \nu k.\{^{k}/_{x}\},$$

which reveals the link between x and y, but not that x is a name. As in this example, several output transitions are sometimes possible, each transition leading to an extended process with a different frame. In reasoning (e.g., in proving that a relation is included in labeled bisimilarity), it often suffices to consider any one of the transitions, so one may be chosen so as to limit the complexity of the resulting extended processes.

We name "simple semantics" the labeled semantics of Section 4.3 and "refined semantics" the semantics of this section, and "simple labels" and "refined labels" the corresponding labels. The next theorem states that the two labeled semantics yield the same notion of equivalence. Thus, making the output labels more explicit only makes apparent some of the information that is otherwise kept in the static, equational part of labeled bisimilarity.

Theorem 4.14. *Let $\approx_L$ be the relation of labeled bisimilarity obtained by applying Definition 4.7 to the refined semantics. We have $\approx_l = \approx_L$.*

The proof of Theorem 4.14 relies on the next two lemmas, which relate simple and refined output transitions.

Lemma 4.15. *$A \xrightarrow{\nu\widetilde{x}.\overline{N}\langle M\rangle} A'$ if and only if, for some z that does not occur in any of $A, A', \widetilde{x}, N$, and M, $A \xrightarrow{\nu z.\overline{N}\langle z\rangle} \nu\widetilde{x}.(\{^{M}/_{z}\} \mid A')$, $\{\widetilde{x}\} \subseteq fv(M) \setminus fv(N)$, and the variables $\widetilde{x}$ are solvable in $\{^{M}/_{z}\} \mid A'$.*

In Lemma 4.15, the transition $A \xrightarrow{\nu\widetilde{x}.\overline{N}\langle M\rangle} A'$ is performed in the refined semantics, while the transition $A \xrightarrow{\nu z.\overline{N}\langle z\rangle} \nu\widetilde{x}.(\{^{M}/_{z}\} \mid A')$ is performed in the simple semantics. However, Lemma 4.16 shows that the choice of the semantics does not matter. Lemma 4.16 is a consequence of Lemma 4.15.

Lemma 4.16. *$A \xrightarrow{\nu x.\overline{N}\langle x\rangle} A'$ in the refined semantics if and only if $A \xrightarrow{\nu x.\overline{N}\langle x\rangle} A'$ in the simple semantics.*

Theorem 4.14 is then proved as follows. By Lemma 4.16, $\approx_L$ is a simple-labeled bisimulation, and thus $\approx_L \subseteq \approx_l$. Conversely, to show that $\approx_l$ is a refined-labeled bisimulation, it suffices to prove its bisimulation property for any refined output label. This proof, which relies on Lemma 4.15, and the proofs of Lemmas 4.12, 4.15, and 4.16 are detailed in Appendix E.

4.5 Proving Theorem 4.8 ($\approx = \approx_l$)

A claim of Theorem 4.8 appears, without proof, in the conference version of this article for the calculus as presented in that version. There, the channels in labels cannot be variables. The claim neglects to include a corresponding hypothesis that exported variables must not be of channel type. This hypothesis is implicitly assumed, as it holds trivially for plain processes and is maintained, as an invariant, by output transitions. Without it, the two extended processes $\nu a.(\{^{a}/_{x}\})$ and $\nu a.(\{^{a}/_{x}\} \mid \overline{a}\langle N\rangle)$ (where the exported variable x stands for the channel a) would constitute a counterexample:

they would not be observationally equivalent, but they would be bisimilar in the labeled semantics, since neither could make a labeled transition. Delaune et al. (2007, 2010) included the hypothesis in their study of symbolic bisimulation. Avik Chaudhuri (private communication, 2007) pointed out this gap in the statement of the theorem, and Bengtson et al. (2011) discussed it as motivation for their work on alternative calculi, the psi calculi, with a more abstract treatment of terms and a mechanized metatheory. On the other hand, Liu (2011) presented a proof of the theorem, making explicit the necessary hypothesis. Her proof demonstrated that the theorem was basically right—no radical changes or new languages were needed. More recently, Liu and others have also developed an extension of the proof for a stateful variant of the applied pi calculus (Arapinis et al. 2014).

Theorem 4.8, in its present form, does not require that hypothesis because of some of the details of the calculus as we define it in this article. Specifically, the labeled semantics allows variables that stand for channels in labels. Therefore, extended processes such as $\nu a.(\{^{a}/_{x}\} \mid \overline{a}\langle N\rangle)$ can make labeled transitions.

This section outlines the proof of Theorem 4.8. The appendix gives further details, including all proofs that this section omits. Those details are fairly long and technical. In particular, they rely on a definition of "partial normal forms" for extended processes, which are designed to simplify reasoning about reductions. (In an extended process $A \mid B$, the frame of A may affect B and vice versa, so A and B may not reduce independently of each other; partial normal forms are designed to simplify the analysis of reductions in such situations.) We believe that these partial normal forms may be useful in other proofs on the applied pi calculus. In this section, we omit further specifics on partial normal forms, since they are not essential to understanding our main arguments.

The proof of Theorem 4.8 starts with a fairly traditional definition of "labeled bisimulation up to ≡":

Definition 4.17. A relation $\mathcal{R}$ on closed extended processes is a *labeled bisimulation up to* ≡ if and only if $\mathcal{R}$ is symmetric and $A\ \mathcal{R}\ B$ implies:

(1) $A \approx_s B$;
(2) if $A \to A'$ and A' is closed, then $B \to^* B'$ and $A' \equiv\mathcal{R}\equiv B'$ for some closed B';
(3) if $A \xrightarrow{\alpha} A'$, A' is closed, and $fv(\alpha) \subseteq dom(A)$, then $B \to^* \xrightarrow{\alpha} \to^* B'$ and $A' \equiv\mathcal{R}\equiv B'$ for some closed B'.

This definition implies that, if $\mathcal{R}$ is a labeled bisimulation up to ≡, then $\equiv\mathcal{R}\equiv$ restricted to closed processes is a labeled bisimulation (since, by Lemma 4.4, static equivalence is invariant by structural equivalence).

We use the definition to establish the following lemma:

Lemma 4.18. *$\approx_l$ is closed by application of closing evaluation contexts.*

In the proof of this lemma (which is given in Appendix C.2), we show that we can restrict attention to contexts of the form $\nu\widetilde{u}.(\_ \mid C)$. To every relation $\mathcal{R}$ on closed extended processes, we associate a relation $\mathcal{R}' = \{(\nu\widetilde{u}.(A \mid C), \nu\widetilde{u}.(B \mid C)) \mid A\ \mathcal{R}\ B, \nu\widetilde{u}.(\_ \mid C) \text{ closing for } A \text{ and } B\}$. We prove that, if $\mathcal{R}$ is a labeled bisimulation, then $\mathcal{R}'$ is a labeled bisimulation up to ≡, and hence $\mathcal{R} \subseteq \equiv\mathcal{R}'\equiv \subseteq \approx_l$. For $\mathcal{R} = \approx_l$, this property entails that $\approx_l$ is closed by application of evaluation contexts $\nu\widetilde{u}.(\_ \mid C)$.

Another lemma characterizes barbs in terms of labeled transitions:

Lemma 4.19. *Let A be a closed extended process. We have $A \Downarrow a$ if and only if $A \to^* \xrightarrow{\nu x.\overline{a}\langle x\rangle} A'$ for some fresh variable x and some A'.*

We then obtain Lemma 4.20, which is one direction of Theorem 4.8:

Lemma 4.20. $\approx_l \subseteq \approx$.

Proof. We show that $\approx_l$ satisfies the three properties of Definition 4.1, as follows:

(1) To show that $\approx_l$ preserves barbs, we apply Lemma 4.19 and use Properties 2 and 3 of Definition 4.7.
(2) Suppose that $A \approx_l B$, $A \to^* A'$, and A' is closed. Given the trace $A = A_0 \to A_1 \to \cdots \to A_n = A'$, we instantiate all variables in $\bigcup_{i=0}^{n}(fv(A_i) \setminus dom(A_i))$ with fresh names. This instantiation yields a trace in which all intermediate processes are closed. We can then conclude that $B \to^* B'$ and $A' \approx_l B'$ for some B' by Property 2 of Definition 4.7.
(3) $\approx_l$ is closed by application of closing evaluation contexts by Lemma 4.18.

Moreover, $\approx_l$ is symmetric. Since $\approx$ is the largest relation that satisfies these properties, we obtain $\approx_l \subseteq \approx$. □

The other direction of Theorem 4.8 relies on two lemmas that characterize input and output transitions. The first lemma characterizes inputs $N(M)$ using processes of the form $T^{p}_{N(M)} \stackrel{\text{def}}{=} \overline{p}\langle p\rangle \mid \overline{N}\langle M\rangle.p(x)$. Here, the use of p as a message in $\overline{p}\langle p\rangle$ is arbitrary: we could equally use processes of the form $\overline{p}\langle M'\rangle$ for any term M'.

Lemma 4.21. *Let A be a closed extended process. Let N and M be terms such that $fv(\overline{N}\langle M\rangle) \subseteq dom(A)$. Let p be a name that does not occur in A, M, and N.*

(1) If $A \xrightarrow{N(M)} A'$ and p does not occur in A', then $A \mid T^{p}_{N(M)} \to\!\to A'$ and $A' \not\downarrow p$.

(2) If $A \mid T^{p}_{N(M)} \to^* A'$ and $A' \not\downarrow p$, then $A \to^* \xrightarrow{N(M)} \to^* A'$.

The second lemma characterizes outputs $\nu x.\overline{N}\langle x\rangle$ using processes of the form $T^{p,q}_{\nu x.\overline{N}\langle x\rangle} \stackrel{\text{def}}{=} \overline{p}\langle p\rangle \mid N(x).p(y).\overline{q}\langle x\rangle$.

Lemma 4.22. *Let A be a closed extended process. Let N be a term such that $fv(N) \subseteq dom(A)$. Let p and q be names that do not occur in A and N.*

(1) If $A \xrightarrow{\nu x.\overline{N}\langle x\rangle} A'$ and p and q do not occur in A', then $A \mid T^{p,q}_{\nu x.\overline{N}\langle x\rangle} \to\!\to \nu x.(A' \mid \overline{q}\langle x\rangle), \nu x.(A' \mid \overline{q}\langle x\rangle) \not\downarrow p$, and $x \notin dom(A)$.

(2) Let x be a variable such that $x \notin dom(A)$. If $A \mid T^{p,q}_{\nu x.\overline{N}\langle x\rangle} \to^* A''$ and $A'' \not\downarrow p$, then $A \to^* \xrightarrow{\nu x.\overline{N}\langle x\rangle} \to^* A'$ and $A'' \equiv \nu x.(A' \mid \overline{q}\langle x\rangle)$ for some A'.

A further lemma provides a way of proving the equivalence of two extended processes with the same domain by putting them in a context that binds the variables in their domain and extrudes them. Given a family of processes P_i for i in a finite set I, we write $\prod_i P_i$ for the parallel composition of the processes P_i if I is not empty, and $\mathbf{0}$ otherwise.

Lemma 4.23. *Let A and B be two closed extended processes with a same domain that contains $\widetilde{x}$. Let $E_{\widetilde{x}}[\_] \stackrel{\text{def}}{=} \nu\widetilde{x}.(\prod_{x\in\widetilde{x}} \overline{n_x}\langle x\rangle \mid \_)$ using names n_x that do not occur in A or B. If $E_{\widetilde{x}}[A] \approx E_{\widetilde{x}}[B]$, then $A \approx B$.*

The final lemma is the other direction of Theorem 4.8:

Lemma 4.24. *$\approx$ is a labeled bisimulation, and thus $\approx \subseteq \approx_l$.*

Proof. The relation $\approx$ is symmetric. We show that it satisfies the three properties of Definition 4.7.

(1) If $A \approx B$, then $A \approx_s B$, by Lemma 4.6.
(2) If $A \approx B$, $A \to A'$, and A' is closed, then $B \to^* B'$ and $A' \approx B'$ for some B', by Property 2 of the definition of $\approx$.
(3) If $A \approx B$, $A \xrightarrow{\alpha} A'$, A' is closed, and $fv(\alpha) \subseteq dom(A)$, then $B \to^* \xrightarrow{\alpha} \to^* B'$ and $A' \approx B'$ for some B'. To prove this property, we rely on characteristic contexts $\_ \mid T_\alpha$ that unambiguously test for a labeled transition $\xrightarrow{\alpha}$ using the disappearance of a barb $\Downarrow p$, and do not otherwise affect $\approx$.

Assume $A \approx B$, $A \xrightarrow{\alpha} A'$, A' is closed, and $fv(\alpha) \subseteq dom(A)$.

(a) For input $\alpha = N(M)$ (where N and M may contain variables exported by A and B) and some fresh name p, we have $A \mid T^p_{N(M)} \to\to A' \not\Downarrow p$ by Lemma 4.21(1), and hence $B \mid T^p_{N(M)} \to^* B' \not\Downarrow p$ with $A' \approx B'$, and hence $B \to^* \xrightarrow{N(M)} \to^* B'$ by Lemma 4.21(2).

(b) For output $\alpha = \nu x.\overline{N}\langle x\rangle$ and some fresh names p and q, we have $A \mid T^{p,q}_{\nu x.\overline{N}\langle x\rangle} \to\to \nu x.(A' \mid \overline{q}\langle x\rangle) \not\Downarrow p$ and $x \notin dom(A)$ by Lemma 4.22(1), and hence $B \mid T^{p,q}_{\nu x.\overline{N}\langle x\rangle} \to^* B'' \not\Downarrow p$ for some B'', and hence $B \to^* \xrightarrow{\nu x.\overline{N}\langle x\rangle} \to^* B'$ and $B'' \equiv \nu x.(B' \mid \overline{q}\langle x\rangle)$ for some B' by Lemma 4.22(2). We obtain a pair $\nu x.(A' \mid \overline{q}\langle x\rangle) \approx \nu x.(B' \mid \overline{q}\langle x\rangle)$ and conclude by applying Lemma 4.23.

Hence, $\approx$ is a labeled bisimulation, and $\approx \subseteq \approx_l$, since $\approx_l$ is the largest labeled bisimulation. □

Theorem 4.8 is an immediate consequence of Lemmas 4.20 and 4.24.

Considering this proof of Theorem 4.8, we can explain further some aspects of our definition of observational equivalence (Definition 4.1). That definition includes conditions related to barbs, reductions, and evaluation contexts (Conditions (1) to (3), respectively), as is done in work on the ν-calculus (Honda and Yoshida 1995) and on the join calculus (Abadi et al. 1998). In an alternative approach, used in CCS (Milner and Sangiorgi 1992) and in the pi calculus (Sangiorgi 1993), equivalence is defined in two stages:

(1) First, barbed bisimilarity is defined as the largest barbed bisimulation, that is, the largest symmetric relation $\mathcal{R}$ such that $A\ \mathcal{R}\ B$ implies Conditions (1) and (2) of Definition 4.1.
(2) Second, equivalence is defined as the largest congruence (i.e., the largest relation $\mathcal{R}$ such that $A\ \mathcal{R}\ B$ implies Condition (3) of Definition 4.1) contained in barbed bisimilarity.

The two approaches do not necessarily yield the same equivalence relation; see Fournet and Gonthier (1998) for positive and negative examples in variants of the pi calculus. The advantage of our approach is that, in reasoning about process equivalences, we can add a context at any point after reductions, as we do in the proof of Lemma 4.24. With the alternative approach, we can add a context only at the beginning, before any reduction, so we need to build contexts that test for all possible sequences of labeled transitions that the processes under consideration may make, and that manifest them as different combinations of barbs. This testing is not possible for all processes, so with the alternative approach, analogs of Theorem 4.8 would typically require a restriction to so-called *image finite* processes (Milner and Sangiorgi 1992). Our definition of observational equivalence avoids this restriction.

It would be interesting to formalize the proofs of this section (and also those of the rest of the article) with a theorem prover such as Coq. This formalization may perhaps benefit from past Coq developments on bisimulations for the pi calculus (Hirschkoff 1997; Honsell et al. 2001) and the spi calculus (Briais 2008). However, the applied pi calculus introduces additional difficulties (because

of the role of terms with equational theories), and proving our results with Coq would certainly require a major effort.

5 DIFFIE-HELLMAN KEY AGREEMENT

The fundamental Diffie-Hellman protocol allows two principals to establish a shared secret by exchanging messages over public channels (Diffie and Hellman 1976). The principals need not have any shared secrets in advance. The basic protocol, on which we focus here as an example, does not provide authentication; therefore, a "bad" principal may play the role of either principal in the protocol. On the other hand, the two principals that follow the protocol will communicate securely with one another afterward, even in the presence of active attackers. In extended protocols, such as the Station-to-Station protocol (Diffie et al. 1992) and SKEME (Krawczyk 1996), additional messages perform authentication.

We program the basic protocol in terms of the binary function symbol f and the unary function symbol g, with the equation

$$\mathrm{f}(x, \mathrm{g}(y)) = \mathrm{f}(y, \mathrm{g}(x)). \tag{6}$$

Concretely, the functions are $\mathrm{f}(x, y) = y^x \ mod\ p$ and $\mathrm{g}(x) = \alpha^x \ mod\ p$ for a prime p and a generator α of $\mathbb{Z}_p^*$, and we have the equation $\mathrm{f}(x, \mathrm{g}(y)) = (\alpha^y)^x = \alpha^{y \times x} = \alpha^{x \times y} = (\alpha^x)^y = \mathrm{f}(y, \mathrm{g}(x))$. However, we ignore the underlying number theory, working abstractly with f and g.

The protocol has two symmetric participants, which we represent by the processes A_0 and A_1. The protocol establishes a shared key, then the participants respectively run P_0 and P_1 using the key. We use the public channel c_{01} for messages from A_0 to A_1 and the public channel c_{10} for communication in the opposite direction. (Although the use of two distinct public channels is of no value for security, it avoids some trivial confusions and so makes for a cleaner presentation.) We assume that none of the values introduced in the protocol appear in P_0 and P_1, except for the key.

In order to establish the key, A_0 invents a name n_0 and sends $\mathrm{g}(n_0)$ to A_1, and A_1 proceeds symmetrically. Then A_0 computes the key as $\mathrm{f}(n_0, \mathrm{g}(n_1))$ and A_1 computes it as $\mathrm{f}(n_1, \mathrm{g}(n_0))$, with the same result. We find it convenient to use the following substitutions for A_0's message and key:

$$\sigma_0 \stackrel{\mathrm{def}}{=} \{^{\mathrm{g}(n_0)}/_{x_0}\}$$
$$\phi_0 \stackrel{\mathrm{def}}{=} \{^{\mathrm{f}(n_0, x_1)}/_{y}\}$$

and the corresponding substitutions σ_1 and ϕ_1, as well as the frame

$$\varphi \stackrel{\mathrm{def}}{=} (\nu n_0.\ (\phi_0 \mid \sigma_0)) \mid (\nu n_1.\ \sigma_1).$$

With these notations, A_0 is

$$A_0 \stackrel{\mathrm{def}}{=} \nu n_0.(\overline{c_{01}}\langle x_0\sigma_0\rangle \mid c_{10}(x_1).P_0\phi_0),$$

and A_1 is analogous.

Two reductions represent a normal run of the protocol:

$$A_0 \mid A_1 \longrightarrow\!\!\!\!\!\rightarrow \nu x_0, x_1, n_0, n_1.\ (P_0\phi_0 \mid P_1\phi_1 \mid \sigma_0 \mid \sigma_1) \tag{7}$$
$$\equiv \quad \nu x_0, x_1, n_0, n_1, y.\ (P_0 \mid P_1 \mid \phi_0 \mid \sigma_0 \mid \sigma_1) \tag{8}$$
$$\equiv \quad \nu y.(P_0 \mid P_1 \mid \nu x_0, x_1.\ \varphi). \tag{9}$$

The two communication steps (Equation (7)) use structural equivalence to activate the substitutions σ_0 and σ_1 and extend the scope of the secret values n_0 and n_1. The structural equivalence (Equation (8)) crucially relies on Equation (6) in order to reuse the active substitution ϕ_0 instead

of ϕ_1 after the reception of x_0 in A_1. The next structural equivalence (Equation (9)) tightens the scope for restricted names and variables, then uses the definition of φ.

We model an eavesdropper as a process $c_{01}(x_0).\overline{c_{01}}\langle x_0\rangle.c_{10}(x_1).\overline{c_{10}}\langle x_1\rangle.P$ that intercepts messages on c_{01} and c_{10}, remembers them, but forwards them unmodified. Using the labeled semantics to represent the interaction of $A_0 \mid A_1$ with such a passive attacker, we obtain

$$\begin{aligned}
A_0 \mid A_1 &\xrightarrow{\overline{c_{01}}\langle x_0\rangle} \nu n_0.(\sigma_0 \mid c_{10}(x_1).P_0\phi_0) \mid A_1 \\
&\xrightarrow{c_{01}(x_0)} \nu n_0.(\sigma_0 \mid c_{10}(x_1).P_0\phi_0) \mid \nu n_1.(\overline{c_{10}}\langle \sigma_1 x_1\rangle \mid P_1\phi_1) \\
&\xrightarrow{\overline{c_{10}}\langle x_1\rangle} \nu n_0.(\sigma_0 \mid c_{10}(x_1).P_0\phi_0) \mid \nu n_1.(\sigma_1 \mid P_1\phi_1) \\
&\xrightarrow{c_{10}(x_1)} \nu n_0.(\sigma_0 \mid P_0\phi_0) \mid \nu n_1.(\sigma_1 \mid P_1\phi_1) \\
&\equiv \quad \nu n_0, n_1, y.\ (P_0 \mid P_1 \mid \phi_0 \mid \sigma_0 \mid \sigma_1) \\
&\equiv \quad \nu y.(P_0 \mid P_1 \mid \varphi).
\end{aligned}$$

The labeled transitions $\xrightarrow{\overline{c_{01}}\langle x_0\rangle}\xrightarrow{c_{01}(x_0)}$ show that the eavesdropper obtains the message sent on c_{01} by A_0, stores it in x_0, and forwards it to A_1. The transitions $\xrightarrow{\overline{c_{10}}\langle x_1\rangle}\xrightarrow{c_{10}(x_1)}$ deal with the message on c_{10} in a similar way. The absence of the restrictions on x_0 and x_1 corresponds to the fact that the eavesdropper has obtained the values of these variables.

The following theorem relates this process to

$$\nu k.(P_0 \mid P_1)\{^k/_y\},$$

which represents the bodies P_0 and P_1 of A_0 and A_1 sharing a key k. This key appears as a simple shared name, rather than as the result of communication and computation. Intuitively, we may read $\nu k.(P_0 \mid P_1)\{^k/_y\}$ as the ideal outcome of the protocol: P_0 and P_1 execute using a shared key, without concern for how the key was established, and without any side effects from weaknesses in the establishment of the key. The theorem says that this ideal outcome is essentially achieved, up to some "noise." This "noise" is a substitution that maps x_0 and x_1 to unrelated, fresh names. It accounts for the fact that an attacker may have the key-exchange messages, and that they look just like unrelated values to the attacker. In particular, the key in use between P_0 and P_1 has no observable relation to those messages, or to any other leftover secrets. We view this independence of the shared key as an important forward-secrecy property.

Theorem 5.1. *Let P_0 and P_1 be processes with free variable y where the name k does not appear. We have*

$$\nu y.(P_0 \mid P_1 \mid \varphi) \ \approx\ \nu k.(P_0 \mid P_1)\{^k/_y\} \mid \nu s_0.\{^{s_0}/_{x_0}\} \mid \nu s_1.\{^{s_1}/_{x_1}\}.$$

Proof. The theorem follows from Lemma 4.5 and the static equivalence $\varphi \approx_s \nu s_0, s_1, k.\{^{s_0}/_{x_0}, ^{s_1}/_{x_1}, ^k/_y\}$, which says that the frame φ generated by the protocol execution is equivalent to one that maps variables to fresh names. This static equivalence is proved automatically by ProVerif, using the technique presented in Blanchet et al. (2008). We conclude by applying the context $\nu y.(P_0 \mid P_1 \mid \_)$. □

Extensions of the basic protocol add rounds of communication that confirm the key and authenticate the principals. We have studied one such extension with key confirmation. There, the shared secret $\mathrm{f}(n_0, \mathrm{g}(n_1))$ is used in confirmation messages. Because of these messages, the shared secret can no longer be equated with a virgin key for P_0 and P_1. Instead, the final key is computed by hashing the shared secret. This hashing guarantees the independence of the final key.

$$
\begin{aligned}
vk.(A \mid B) &\xrightarrow{a(M)} vk.(A \mid B \mid \overline{b}\langle(M, \mathsf{mac}(k, M))\rangle) \\
&\xrightarrow{\nu x.\overline{b}\langle x\rangle} vk.(A \mid B \mid \{^{(M,\mathsf{mac}(k,M))}/_{x}\}) \\
&\xrightarrow{b(x)} vk.(A \mid \overline{c}\langle M\rangle \mid \{^{(M,\mathsf{mac}(k,M))}/_{x}\}) \\
&\xrightarrow{\nu y.\overline{c}\langle y\rangle} vk.(A \mid \{^{(M,\mathsf{mac}(k,M))}/_{x}, {}^{M}/_{y}\})
\end{aligned}
$$

Fig. 4. A correct trace.

We have also studied more advanced protocols that rely on a Diffie-Hellman key exchange, such as the JFK protocol (Aiello et al. 2004). The analysis of JFK in the applied pi calculus (Abadi et al. 2007) illustrates the composition of manual reasoning with invocations of ProVerif.

6 HASH FUNCTIONS AND MESSAGE AUTHENTICATION CODES

Section 3 briefly discusses cryptographic hash functions. In this section, we continue their study, and also treat message authentication codes (MACs). We consider constructions of both hash functions and MACs. These examples provide a further illustration of the usefulness of equations in the applied pi calculus. On the other hand, some aspects of the constructions are rather low level, and we would not expect to account for all their combinatorial details (e.g., the "birthday attacks" (Menezes et al. 1996)). A higher-level task is to express and reason about protocols treating hash functions and MACs as primitive; this is squarely within the scope of our approach.

6.1 Using MACs

MACs serve to authenticate messages using shared keys. When k is a key and M is a message, and k is known only to a certain principal A and to the recipient B of the message, B may take $\mathsf{mac}(k, M)$ as proof that M comes from A. More precisely, B can check $\mathsf{mac}(k, M)$ by recomputing it upon receipt of M and $\mathsf{mac}(k, M)$, and reason that A must be the sender of M. This property should hold even if A generates MACs for other messages as well; those MACs should not permit forging a MAC for M. In the worst case, it should hold even if A generates MACs for other messages on demand.

Using a new binary function symbol mac, we may describe this scenario by the following processes:

$$
\begin{aligned}
A &\stackrel{\text{def}}{=} !a(x).\overline{b}\langle(x, \mathsf{mac}(k, x))\rangle \\
B &\stackrel{\text{def}}{=} b(y).\mathit{if}\ \mathsf{mac}(k, \mathsf{fst}(y)) = \mathsf{snd}(y)\ \mathit{then}\ \overline{c}\langle\mathsf{fst}(y)\rangle \\
S &\stackrel{\text{def}}{=} vk.(A \mid B)
\end{aligned}
$$

The process S represents the complete system, composed of A and B; the restriction on k means that k is private to A and B. The process A receives messages on a public channel a and returns them MACed on the public channel b. When B receives a message on b, it checks its MAC and acts upon it, here simply by forwarding on a channel c. Intuitively, we would expect that B forwards on c only a message that A has MACed. In other words, although an attacker may intercept, modify, and inject messages on b, it should not be able to forge a MAC and trick B into forwarding some other message. Hence, every message output on c equals a preceding input on a, as illustrated in Figure 4.

This property can be expressed precisely in terms of the labeled semantics and it can be checked without too much difficulty when mac is a primitive function symbol with no equations. The property remains true even if there is a function extract that maps a MAC $\mathsf{mac}(x, y)$ to the underlying

$$
\begin{array}{lll}
\nu k.(A \mid B) & \xrightarrow{a(M)} & \nu k.(A \mid B \mid \overline{b}\langle (M, \mathsf{mac}(k, M))\rangle) \\
& \xrightarrow{\nu x.\overline{b}\langle x\rangle} & \nu k.(A \mid B \mid \{^{(M,\mathsf{mac}(k,M))}/_{x}\}) \\
& \xrightarrow{b((M+N,\mathsf{f}(\mathsf{snd}(x),N)))} & \nu k.(A \mid \overline{c}\langle M + N\rangle \mid \{^{(M,\mathsf{mac}(k,M))}/_{x}\}) \\
& \xrightarrow{\nu y.\overline{c}\langle y\rangle} & \nu k.(A \mid \{^{(M,\mathsf{mac}(k,M))}/_{x}, {}^{M+N}/_{y}\})
\end{array}
$$

Fig. 5. An attack scenario.

cleartext y, with the equation $\mathsf{extract}(\mathsf{mac}(x, y)) = y$. Since MACs are not supposed to guarantee secrecy, such a function may well exist, so it is safer to assume that it is available to the attacker.

The property is more delicate if mac is defined from other operations, as it invariably is in practice. In that case, the property may even be taken as *the* specification of MACs (Goldwasser and Bellare 1999). Thus, a MAC implementation may be deemed correct if and only if the process S works as expected when mac is instantiated with that implementation. More specifically, the next section deals with the question of whether the property remains true when mac is defined from hash functions.

6.2 Constructing Hash Functions and MACs

In Section 3, we give no equations for hash functions. In practice, following Merkle and Damgård, hash functions are commonly defined by iterating a basic binary compression function, which maps two input blocks to one output block (Menezes et al. 1996). Furthermore, keyed hash functions include a key as an additional argument. Thus, we may have

$$\mathsf{h}(x, y_0 :: y_1 :: z) = \mathsf{h}(\mathsf{f}(x, y_0), y_1 :: z) \tag{10}$$

$$\mathsf{h}(x, y :: \mathsf{nil}) = \mathsf{f}(x, y). \tag{11}$$

Here, we use the sorts Block for blocks and BlockList for sequences of blocks, defined as lists as in Section 3, with sorts Block and BlockList instead of Data and List, respectively. The function $\mathsf{h} : \mathsf{Block} \times \mathsf{BlockList} \to \mathsf{Block}$ is the keyed hash function; $\mathsf{f} : \mathsf{Block} \times \mathsf{Block} \to \mathsf{Block}$ is the compression function.

In these equations, we are rather abstract in our treatment of blocks, their sizes, and therefore padding and other related issues. We also ignore two common twists: some functions use initialization vectors to start the iteration, and some append a length block to the input. Nevertheless, we can explain various MAC constructions, describing flaws in some and reasoning about the properties of others.

A first, classical definition of a MAC from a keyed hash function h is

$$\mathsf{mac}(x, y) \stackrel{\mathrm{def}}{=} \mathsf{h}(x, y).$$

For instance, the MAC of a three-block message $M = M_1 :: M_2 :: M_3 :: \mathsf{nil}$ with key k is $\mathsf{mac}(k, M) = \mathsf{f}(\mathsf{f}(\mathsf{f}(k, M_1), M_2), M_3)$. More generally, the MAC of an n-block message $M = M_1 :: \ldots :: M_n :: \mathsf{nil}$ is $\mathsf{mac}(k, M) = \mathsf{f}(\ldots(\mathsf{f}(k, M_1), \ldots), M_n)$. This implementation is subject to a well-known extension attack. Given the MAC of $M = M_1 :: \ldots :: M_n :: \mathsf{nil}$, an attacker can compute the MAC of any extension $M + N = M_1 :: \ldots :: M_n :: N :: \mathsf{nil}$ without knowing the MAC key, since $\mathsf{mac}(k, M + N) = \mathsf{f}(\mathsf{mac}(k, M), N)$.

We describe the extension attack formally, through the operational semantics of the process S of Section 6.1, in Figures 5 and 6. These figures use the semantics of Sections 4.3 and 4.4, respectively. In both cases, we assume $k \notin fn(M) \cup fn(N)$. Additionally, we adopt the sorts

$$
\begin{array}{lcl}
\nu k.(A \mid B) & \xrightarrow{a(M)} & \nu k.(A \mid B \mid \overline{b}\langle (M, \mathsf{mac}(k, M))\rangle) \\
& \xrightarrow{\nu x.\overline{b}\langle (M,x)\rangle} & \nu k.(A \mid B \mid \{^{\mathsf{mac}(k,M)}/_{x}\}) \\
& \xrightarrow{b((M+N,\mathsf{f}(x,N)))} & \nu k.(A \mid \overline{c}\langle M +\!\!+ N\rangle \mid \{^{\mathsf{mac}(k,M)}/_{x}\}) \\
& \xrightarrow{\overline{c}\langle M+N\rangle} & \nu k.(A \mid \{^{\mathsf{mac}(k,M)}/_{x}\})
\end{array}
$$

Fig. 6. An attack scenario (with refined labels).

pair : BlockList × Block → Data, fst : Data → BlockList, and snd : Data → Block; the abbreviation (M, N) for $\mathsf{pair}(M, N)$; and Equations (1) and (2) of Section 3. In Figures 5 and 6, we see that the message M that the system MACs differs from the message $M +\!\!+ N$ that it forwards on c. These transitions are not enabled with the primitive MAC of Section 6.1; hence, S with the proposed MAC implementation is not labeled bisimilar to S with the primitive MAC.

There are several ways to address extension attacks, and indeed the literature contains many MAC constructions that are not subject to these attacks. We have considered some of them. Here we describe a construction that uses the MAC key twice:

$$\mathsf{mac}(x, y) \stackrel{\mathrm{def}}{=} \mathsf{f}(x, \mathsf{h}(x, y)).$$

Under this definition, the MAC of $M = M_1 :: M_2 :: M_3 :: \mathsf{nil}$ with key k is $\mathsf{mac}(k, M) = \mathsf{f}(k, \mathsf{f}(\mathsf{f}(\mathsf{f}(k, M_1), M_2), M_3))$, and the process S forwards on c only a message that it has previously MACed, as desired.

Looking beyond the case of S, we can prove a more general result by comparing the situation where mac is primitive (and has no special equations) and one with the definition of $\mathsf{mac}(x, y)$ as $\mathsf{f}(x, \mathsf{h}(x, y))$. Given a name k and an extended process C that uses the symbol mac, we write $[\![C]\!]$ for the translation of C in which the definition of mac is expanded wherever the key k is used, with $\mathsf{f}(k, \mathsf{h}(k, M))$ replaced for $\mathsf{mac}(k, M)$. The theorem says that this translation yields an equivalent process (so, intuitively, the constructed MACs work as well as the primitive ones). It applies to a class of equational theories generated by rewrite rules.

Theorem 6.1. *Suppose that the signature* Σ *is equipped with an equational theory generated by a convergent rewrite system such that* mac *and* f *do not occur in the left-hand sides of rewrite rules; the only rewrite rules with* h *at the root of the left-hand side are those of Equations (10) and (11) oriented from left to right; there are no rewrite rules with* :: *nor* nil *at the root of the left-hand side; and names do not occur in rewrite rules. Suppose that* C *is closed and the name* k *appears only as first argument of* mac *in* C. *Then* $\nu k.C \approx \nu k.[\![C]\!]$.

In the proof of this theorem (which is given in Appendix F), we use the same notion of partial normal form as in the proof of Theorem 4.8. We define a relation $\mathcal{R}$ by $A\ \mathcal{R}\ B$ if and only if A and B are closed, $A \equiv \nu k.C$, $B \equiv \nu k.[\![C]\!]$, C is a closed extended process in partial normal form, and the name k appears only as MAC key in C. We show that the relation $\mathcal{R} \cup \mathcal{R}^{-1}$ (i.e., the union of $\mathcal{R}$ with its inverse relation) is a labeled bisimulation. Static equivalence follows from the preservation of equality by the translation $[\![\ \cdot\ ]\!]$ for terms in which k occurs only as MAC key; reductions commute with the translation $[\![\ \cdot\ ]\!]$ and preserve the restriction on the occurrences of the key k. We conclude by Theorem 4.8. An alternative proof of similar complexity would show that $\mathcal{R} \cup \mathcal{R}^{-1}$ is an observational bisimulation.

Theorem 6.1 considers a single MAC key at a time. For an extended process with several MAC keys $k_1, \ldots, k_n$, we can apply Theorem 6.1 once for each key k_i, using structural equivalence to move each restriction vk_i to the root of the extended process.

Theorem 6.1 allows cryptographic primitives other than hash functions and MACs, provided the assumptions on the equational theory are satisfied. The following corollary states a simple special case for the primitives mentioned in this section. It suffices for treating the system S.

Corollary 6.2. *Suppose that the signature Σ is equipped with the equational theory defined by Equations (1), (2), (3), (4), (10), and (11). Suppose that C is closed and the name k appears only as a first argument of* mac *in C. Then $vk.C \approx vk.[\![C]\!]$.*

6.3 Constructing Robust Hash Functions

Constructions of hash functions, of the kind described in Section 6.2, typically impose constraints on the use of these functions. For example, some care is needed in order to thwart extension attacks in the definition of MACs. The possibility of such attacks stems from structural flaws in the constructions; details such as the iteration of a compression function are not completely hidden, lead to unwanted additional properties, and can be exploited.

A line of work in cryptography studies safer hash functions with stronger guarantees (Coron et al. 2005). Although these functions are generally built much as in Section 6.2 by iterating a compression function, their design conceals their inner structure. The functions thus aim to behave like abstract "random oracles" on inputs of arbitrary length. A notion of indifferentiability captures this goal.

In this section, as a final, more advanced example, we describe one design that strengthens the Merkle-Damgård approach, following Coron et al. (2005, Section 3.4). In this example, the attacker is given only indirect access to functions, such as the hash function h. We model this restriction by inserting a private name k as the first argument of h. (Cryptographically, the name k may reflect the initial random sampling of h.) We refer to this argument as a key, of sort Key. We use sorts Block for blocks and BlockList for sequences of blocks, defined as lists as in Section 3, with sorts Block and BlockList instead of Data and List, respectively. We use sort Block2 for pairs of blocks, with pair : Block $\times$ Block $\rightarrow$ Block2, fst : Block2 $\rightarrow$ Block, and snd : Block2 $\rightarrow$ Block; the abbreviation (x, y) for $\mathsf{pair}(x, y)$; and the equations

$$\mathsf{fst}((x, y)) = x \qquad \mathsf{snd}((x, y)) = x \qquad (\mathsf{fst}(x), \mathsf{snd}(x)) = x. \tag{12}$$

The third equation of (12) is not present in Section 3; it models that all elements of sort Block2 are pairs. We use sort Block3 for pairs of a Block2 and a Block defined in the same way with overloaded function symbols pair, fst, and snd, and sort Bool for Booleans.

We define the hash function h : Key $\times$ BlockList $\rightarrow$ Block by

$$\mathsf{h}(k, z) = \mathsf{h}_2(k, (0, 0), z), \tag{13}$$

where

$$\mathsf{h}_2(k, x, \mathsf{nil}) = \mathsf{fst}(x) \tag{14}$$

$$\mathsf{h}_2(k, x, y :: z) = \mathsf{h}_2(k, \mathsf{f}(k, (x, y)), z). \tag{15}$$

The function h_2 : Key $\times$ Block2 $\times$ BlockList $\rightarrow$ Block uses a compression function f : Key $\times$ Block3 $\rightarrow$ Block2. In $\mathsf{h}_2(k, x, z)$, the variable x represents the fixed-size internal state of the hash function and z is the remainder of the input. The internal state starts at $(0, 0)$ and is updated by applications of the compression function f as input blocks are processed. Finally, only the first half of the internal state is returned.

For instance, the hash of a two-block message $M = M_1 :: M_2 :: \mathsf{nil}$ with key k is $\mathsf{h}(k, M) = \mathsf{fst}(\mathsf{f}(k, (\mathsf{f}(k, ((0,0), M_1)), M_2)))$. More generally, we have

$$\mathsf{h}(k, M_1 :: \ldots :: M_n :: \mathsf{nil}) = \mathsf{fst}(\mathsf{f}(k, (\ldots \mathsf{f}(k, ((0,0), M_1)) \ldots, M_n))).$$

Indifferentiability requires that the hash function behave like a black box (like a "random oracle"), even in interaction with an adversary that also has access to the underlying compression function. The compression function and the hash function are related, of course. However, as far as the adversary can tell, it is the compression function that may be defined from the hash function (in fact, from an ideal hash function without equations as in Section 3) rather than the other way around. Thus, we express indifferentiability as the equivalence of two systems, each of which provides access to the hash function and the compression function. In the applied pi calculus, one of the systems is

$$\nu k.(A_h^0 \mid A_f^0),$$

where the processes

$$A_h^0 = !c_h(y).\mathit{if}\ \mathsf{ne\_list}(y) = \mathsf{true}\ \mathit{then}\ \overline{c'_h}\langle \mathsf{h}(k, y)\rangle$$
$$A_f^0 = !c_f(x).\overline{c'_f}\langle \mathsf{f}(k, x)\rangle$$

answer requests to evaluate h and f with key k. We restrict ourselves to hashes of nonempty sequences of blocks. In practice, one never hashes the empty string, because the input of the hash function is padded to a nonzero multiple of the block length. This restriction is important in this example, because the definition of h yields $\mathsf{h}(k, \mathsf{nil}) = 0$, and this special hash value would break indifferentiability. In order to enforce this restriction, we use symbols $\mathsf{true} : \mathsf{Bool}$ and $\mathsf{ne\_list} : \mathsf{BlockList} \to \mathsf{Bool}$, with equations

$$\mathsf{ne\_list}(x :: \mathsf{nil}) = \mathsf{true} \qquad \mathsf{ne\_list}(x :: y :: z) = \mathsf{ne\_list}(y :: z). \tag{16}$$

The term $\mathsf{ne\_list}(M)$ is equal to true when M is a nonempty list.

The other system offers an analogous interface for an ideal hash function $\mathsf{h}' : \mathsf{Key} \times \mathsf{BlockList} \to \mathsf{Block}$ and for a stateful compression function built from h':

$$\nu k.(A_h^1 \mid A_f^1).$$

The process A_h^1 answers requests to evaluate an ideal hash function h':

$$A_h^1 = !c_h(y).\mathit{if}\ \mathsf{ne\_list}(y) = \mathsf{true}\ \mathit{then}\ \overline{c'_h}\langle \mathsf{h}'(k, y)\rangle$$

and A_f^1 simulates the compression function using h'. The code for A_f^1, which is considerably more intricate, captures the core of the security argument as it might appear in the cryptography literature. (The paper by Coron et al. (2005) omits this argument and, as far as we know, this argument does not appear elsewhere.)

$$A_f^1 = \nu \ell, c_s.(!c_s(s).c_f(x).\overline{\ell}\langle x, s, s\rangle \mid !Q \mid \overline{c_s}\langle((0,0), \mathsf{nil}) :: \mathsf{nil}\rangle)$$
$$Q = \ell(x, t, s).\mathit{if}\ t = \mathsf{nil}\ \mathit{then}\ P_0\ \mathit{else}$$
$$\quad \mathit{if}\ \mathsf{fst}(\mathsf{hd}(t)) = \mathsf{fst}(x)\ \mathit{then}\ P_1\ \mathit{else}\ \overline{\ell}\langle x, \mathsf{tl}(t), s\rangle$$
$$P_0 = \overline{c'_f}\langle \mathsf{f}'(k, x)\rangle \mid \overline{c_s}\langle s\rangle$$
$$P_1 = \mathit{let}\ z = \mathsf{snd}(\mathsf{hd}(t))\ \mathit{in}$$
$$\quad \mathit{let}\ z' = z +\!\!+ \mathsf{snd}(x)\ \mathit{in}$$
$$\quad \mathit{let}\ r = (\mathsf{h}'(k, z'), \mathsf{f}_c(k, z'))\ \mathit{in}$$
$$\quad \overline{c'_f}\langle r\rangle \mid \overline{c_s}\langle(r, z') :: s\rangle$$

In this definition, *let* $x = M$ *in* P is syntactic sugar for $P\{^{M}/_{x}\}$; $\ell(x,t,s).P$ is syntactic sugar for $\ell(y).let\ x = \mathsf{fst}(y)\ in\ let\ t = \mathsf{fst}(\mathsf{snd}(y))\ in\ let\ s = \mathsf{snd}(\mathsf{snd}(y))\ in\ P$, where y is a fresh variable; and $\overline{\ell}\langle x,t,s\rangle$ is syntactic sugar for $\overline{\ell}\langle(x,(t,s))\rangle$, with the appropriate sorts and overloading of the function symbols for pairs. The function symbol $\mathsf{f}' : \mathsf{Key} \times \mathsf{Block3} \to \mathsf{Block2}$ represents the compression function outside the domain used for implementing the hash function, and the function symbol $\mathsf{f}_c : \mathsf{Key} \times \mathsf{BlockList} \to \mathsf{Block}$ represents the second projection of the compression function inside that domain. The channel c_s maintains the global private state, a lookup table that maps each term $(\mathsf{h}'(k,M), \mathsf{f}_c(k,M))$ with $M = M_1 :: \ldots :: M_n :: \mathsf{nil}$ built as a result of previous compression requests to the term M, and initially maps $(0,0)$ to nil. This lookup table is represented as a list of pairs. Each table element, of sort Block2Blocks, is a pair of a Block2 and a BlockList; the table, of sort Block2Blocks_List, is a list of Block2Blocks; we overload the function symbols for pairs and lists. Upon a compression request with input x, the process Q looks up $\mathsf{fst}(x)$ in the table: Q receives as input x, the initial state of the table s, and the tail t of the lookup table. It uses a local channel l for encoding the recursive call. The auxiliary processes P_0 and P_1 complete compression requests in the cases where lookups fail and succeed, respectively. When a lookup fails, the compression request is outside the domain used for implementing the hash function, so P_0 answers it using f' and leaves the table unchanged. When a lookup succeeds, we have either $\mathsf{fst}(x) = (\mathsf{h}'(k,M), \mathsf{f}_c(k,M))$ with $M = M_1 :: \ldots :: M_{n-1} :: \mathsf{nil}$ and $n > 0$, or $\mathsf{fst}(x) = (0,0)$ and we let $M = \mathsf{nil}$. The lookup yields $z = M$, and P_1 computes $z' = z +\!\!+ \mathsf{snd}(x)$ and returns $r = (\mathsf{h}'(k,z'), \mathsf{f}_c(k,z'))$ as result of the compression request. The table is extended by adding the mapping from r to z'.

Let us now explain, informally, why this code ensures that the results of the compression function are consistent with those of hash computations. The result of a compression request with argument x needs to be made consistent with the hash function when

$$x = (\mathsf{f}(k, (\ldots \mathsf{f}(k, ((0,0), M_1)) \ldots, M_{n-1})), M_n) \tag{17}$$

for some $M_1, \ldots, M_n$ $(n > 0)$, because in that case

$$\mathsf{h}(k, M_1 :: \ldots :: M_n :: \mathsf{nil}) = \mathsf{fst}(\mathsf{f}(k,x)); \tag{18}$$

that is, in the system $\nu k.(A^0_h \mid A^0_f)$, the result of the hash request with argument $M_1 :: \ldots :: M_n :: \mathsf{nil}$ computed by A^0_h is equal to the first block of the result of the compression request with argument x computed by A^0_f. We need to have an analogous equality in the system $\nu k.(A^1_h \mid A^1_f)$. In the system $\nu k.(A^0_h \mid A^0_f)$, the equality in Equation (17) holds exactly when $\mathsf{fst}(x)$ is the result of previous compression requests $\mathsf{f}(k, (\ldots \mathsf{f}(k, ((0,0), M_1)) \ldots, M_{n-1}))$ for some $M_1, \ldots, M_{n-1}$. In the system $\nu k.(A^1_h \mid A^1_f)$, the table lookup tests a corresponding condition, and when it succeeds, P_1 retrieves $z = M_1 :: \ldots :: M_{n-1} :: \mathsf{nil}$, computes $z' = M_1 :: \ldots :: M_{n-1} :: M_n :: \mathsf{nil}$ since $\mathsf{snd}(x) = M_n$, and returns $r = (\mathsf{h}'(k,z'), \mathsf{f}_c(k,z'))$. Hence, $\mathsf{fst}(r) = \mathsf{h}'(k,z')$ and the result of the hash request with argument $z' = M_1 :: \ldots :: M_n :: \mathsf{nil}$ computed by A^1_h is equal to the first block of the result r of the compression request with argument x computed by A^1_f.

Formally, we obtain the following observational equivalence:

THEOREM 6.3. $\nu k.(A^0_h \mid A^0_f) \approx \nu k.(A^1_h \mid A^1_f)$.

In the proof of this theorem (which is given in Appendix G), we define a relation $\mathcal{R}$ between configurations of the two systems and show that $\mathcal{R} \cup \mathcal{R}^{-1}$ is a labeled bisimulation. A key step of this proof consists in proving static equivalence between related configurations; this step formalizes the informal explanation of the process A^f_1 given above. We conclude by Theorem 4.8.

7 RELATED WORK

This section aims to position the applied pi calculus with respect to research on process calculi and on the analysis of security protocols. As discussed in Section 1, the applied pi calculus has been the basis for much further work since its initial publication; this section does not discuss many papers that build on the applied pi calculus. (Some of those papers, and others, are mentioned elsewhere in this article.)

7.1 Process Calculi

The applied pi calculus has many commonalities with the original pi calculus (Milner 1999) and its relatives, such as the spi calculus (Abadi and Gordon 1999) (discussed in Sections 3 and 4). In particular, the model of communication adopted in the applied pi calculus is deliberately classical: communication is through named channels, and value computation is rather separate from communication.

Furthermore, active substitutions are reminiscent of the constraints of the fusion calculus (Victor 1998). They are especially close to the substitution environments that Boreale et al. employ in their proof techniques for a variant of the spi calculus with a symmetric cryptosystem (Boreale et al. 1999). We incorporate substitutions into processes, systematize them, and generalize from symmetric cryptosystems to arbitrary operations and equations.

Extensions of the pi calculus are not limited to modeling cryptography: many extensions and variants of the pi calculus have been designed for diverse applications. Examples include calculi for mobility, such as the ambient calculus (Cardelli and Gordon 2000); calculi for modeling biological processes, such as the enhanced pi calculus (Curti et al. 2004); and calculi for service-oriented computing, which model the contracts that services implement, the composition of services, and their protocols (Buscemi and Montanari 2007; Cruz-Filipe et al. 2014; Lapadula et al. 2007; Lucchi and Mazzara 2007). The psi calculi (Bengtson et al. 2011) provide a general framework parameterized by nominal data types for terms, conditions (generalizing comparison between terms), and assertions (generalizing our notion of frames) and their operational semantics. They also give sufficient conditions on these parameters to ensure that the resulting observational equivalence coincides with labeled bisimilarity. The framework accommodates encodings of the pi calculus and several of its variants, for example, ones with fusion (Wischik and Gardner 2004) and concurrent constraints (Buscemi and Montanari 2007). In particular, Bengtson et al. give an encoding of the applied pi calculus in their framework. However, as they explain, the result of this encoding differs from the applied pi calculus in the way processes interact with contexts. In particular, an important difference is that, when an encoded process sends a ciphertext, the ciphertext appears on the label of the transition, and an agent that receives this message will immediately learn the cleartext and the key. In psi calculi, one can avoid such counterintuitive disclosures by explicitly creating and using aliases. A recent extension of psi calculi (Borgström et al. 2014, 2016) addresses these difficulties with a new form of pattern matching. In contrast, the management of aliases is built into the applied pi calculus, facilitating the modeling of security protocol attackers as contexts.

7.2 Analysis of Security Protocols

The analysis of a security protocol generally requires reasoning about its possible executions. However, the ways of talking about the executions and their properties vary greatly. We use a process calculus whose semantics provides a detailed specification for interactions with a context. Because the process calculus has a proper "new" construct (like the pi calculus but unlike CSP), it provides a direct account of the generation of new keys and other fresh quantities. It also enables reasoning with equivalence and implementation relations.

Reasoning with those relations is often more challenging than reasoning about trace properties, but it can be worthwhile. Equivalences are useful, in particular, for modeling privacy properties (Pfitzmann and Köhntopp 2001), for instance, in electronic voting (Delaune et al. 2009). While proofs of equivalences are difficult to automate in general—and observational equivalence is undecidable as noted in Section 4.1—several tools support certain automatic proofs of equivalences in the applied pi calculus and similar languages: tools have focused on establishing particular kinds of equivalences such as trace equivalence for bounded processes (i.e., processes without replication) (Chadha et al. 2012; Cheval et al. 2013; Tiu and Dawson 2010) or for restricted classes of unbounded processes (Chrétien et al. 2015a, 2015b). Although ProVerif initially focused on proofs of trace properties (Blanchet 2009), it also supports automatic proofs of diff-equivalences, which are equivalences between processes that share the same structure and differ only in the choice of terms (Blanchet et al. 2008). A diff-equivalence between two processes requires that the two processes reduce in the same way, in the presence of any adversary. In particular, the two processes must have the same branching behaviour. Hence, diff-equivalence is much stronger than observational equivalence. Maude-NPA (Santiago et al. 2014) and Tamarin (Basin et al. 2015) also employ that notion. Baudet (2005, 2007) showed that diff-equivalence is decidable for bounded processes: he treated a class of equivalences that model security against offline guessing attacks in Baudet (2005) and proved the full result in Baudet (2007).

Furthermore, the use of a process calculus permits treating security protocols as programs written in a programming notation—subject to typing (Abadi 1999; Cardelli 2000; Gordon and Jeffrey 2004), to other static analyses (Bodei et al. 1998), and to translations (Abadi 1998; Abadi et al. 1998, 2000). Thus, language-based approaches have led to tools such as ProVerif where protocols can be described by programs and analyzed using automated techniques that leverage type systems and Horn clauses (Abadi and Blanchet 2005a).

The applied pi calculus is also convenient as an intermediate language. Translations to ProVerif have been implemented from TulaFale (a language for standardized web services protocols) (Bhargavan et al. 2003), from F# (Bhargavan et al. 2008b), and from JavaScript in order to verify protocols, including TLS (Bhargavan et al. 2008a).

As in many other works (e.g., Abadi and Gordon (1999), Amadio and Lugiez (2000), Armando et al. (2005), Cremers (2008), Dam (1998), DeMillo et al. (1982), Dolev and Yao (1983), Escobar et al. (2006), Kemmerer et al. (1994), Lowe (1996), Merritt (1983), Mitchell et al. (1997), Paulson (1998), Schmidt et al. (2012), Schneider (1996), and Thayer Fábrega et al. (1998)), our use of the applied pi calculus conveniently avoids matters of computational complexity and probability. In contrast, other techniques for the analysis of security protocols employ more concrete computational models, where principals are basically Turing machines that manipulate bitstrings, and security depends on the computational limitations of attackers (e.g., Bellare and Rogaway (1993), Goldwasser and Bellare (1999), Goldwasser and Micali (1984), Goldwasser et al. (1988), and Yao (1982)).

Although these two approaches remained rather distinct during the 1980s and 1990s, fruitful connections have now been established (e.g., Abadi et al. (2009), Abadi and Rogaway (2002), Backes et al. (2009), Barthe et al. (2010), Blanchet (2006), Blanchet and Pointcheval (2006), Comon-Lundh and Cortier (2008), Datta et al. (2005), Lincoln et al. (1998), and Pfitzmann et al. (2000)). In particular, some work interprets symbolic proofs in terms of concrete, bitstring-based models (Abadi and Rogaway 2002), in some cases specifically studying the "computational soundness" of the applied pi calculus (Backes et al. 2009; Baudet et al. 2009b; Comon-Lundh and Cortier 2008). Other work focuses directly on those concrete models but benefits from notations and ideas from process calculi and programming languages. For example, the tool CryptoVerif (Blanchet 2006; Blanchet and Pointcheval 2006) provides guarantees in terms of bitstrings, running times, and probabilities, but

its input language is strongly reminiscent of the applied pi calculus, which influenced it—rather than of Turing machines.

8 CONCLUSION

In this article, we describe a uniform extension of the pi calculus, the applied pi calculus, in which messages may be compound values, not just channel names. We study its theory, developing its semantics and proof techniques. Although the calculus has no special, built-in features to deal with security, it has proved useful in the analysis of security protocols.

Famously, the pi calculus is the language of those lively social occasions where all conversations are exchanges of names. The applied pi calculus opens the possibility of more substantial, structured conversations; the cryptic character of some of these conversations can only add to their charm and to their tedium.

The previous paragraph closed the conference paper that introduced the applied pi calculus in 2001. We are now in a better position to assess the possibility to which it refers. As we hoped in 2001, the applied pi calculus has been extensively used for modeling and for reasoning about security protocols, particularly ones that rely heavily on cryptography (and less for ones that rely on simple capabilities). The flexibility of the applied pi calculus is a key enabler for those applications. This flexibility did not render unnecessary or uninteresting the exploration of variants and extensions. However, it did allow the applied pi calculus to remain a relevant core system—it was not displaced by an extended language with many more constructs.

We are also in a better position to comment on matters of charm and tedium, alas. It is debatable whether security protocols have become more charming or more tedious since 2001. It is clear, however, that they play an ever-growing role, and that their security remains problematic. The evolution of TLS exemplifies these points. The literature now contains many attacks on TLS (e.g., Adrian et al. (2015), Aviram et al. (2016), Bhargavan et al. (2014a), and Vanhoef and Piessens (2015)), but also several partial specifications and proofs (Cremers et al. 2016; Jager et al. 2012; Krawczyk et al. 2013; Paterson et al. 2011), sometimes relying on the applied pi calculus (Bhargavan et al. 2017a, 2012), and sometimes with language-based methods of the kind that the applied pi calculus started to explore (Bhargavan et al. 2014b, 2013). The state-of-the-art approaches (Bhargavan et al. 2017a, 2017b, 2014b, 2013; Jager et al. 2012; Krawczyk et al. 2013; Paterson et al. 2011) rely on refined frameworks that consider matters of computational complexity and probability, which are beyond the (explicit) scope of the applied pi calculus, as explained above. In such applications, tools play a helpful role, often an essential one. Although they sometimes lead to important insights, manual proofs—in particular, manual proofs of equivalences—can be rather painful and tedious. (We may have suspected this fact in 2001; on the basis of our experience since then, we now know it with certainty.) On the other hand, the relative ease of use of ProVerif has contributed greatly to the spread of the applied pi calculus. The applied pi calculus has evolved through its implementation in ProVerif, and as a result of its use in this context. That evolution is, in our opinion, an improvement, so the present article, aims to reflect it.

Finally, independently of the merits and the future of the applied pi calculus, we believe that languages, and in particular the formalization of attackers as contexts, should continue to play a role in the analysis of security protocols. The alternatives (defining protocols as interacting Turing machines?) are not easier. Describing protocols in a programming notation not only makes them precise but also brings them into the realm where ideas and tools from programming can support analysis.

ACKNOWLEDGMENTS

We thank Rocco De Nicola, Andy Gordon, Tony Hoare, and Phil Rogaway for discussions that contributed to this work. Georges Gonthier and Jan Jürjens suggested improvements to the

presentation of the conference version of this paper. Steve Kremer and Ben Smyth provided helpful comments on a draft of this paper.

REFERENCES

Martín Abadi. 1998. Protection in programming-language translations. In *Proceedings of the 25th International Colloquium on Automata, Languages and Programming (Lecture Notes in Computer Science)*, Kim G. Larsen, Sven Skyum, and Glynn Winskel (Eds.), Vol. 1443. Springer, Heidelberg, 868–883. Also Digital Equipment Corporation Systems Research Center report No. 154, April 1998.

Martín Abadi. 1999. Secrecy by typing in security protocols. *Journal of the ACM* 46, 5 (Sept. 1999), 749–786.

Martín Abadi. 2007. Security protocols: Principles and calculi. In *Foundations of Security Analysis and Design IV (FOSAD'07) Tutorial Lectures (Lecture Notes in Computer Science)*, Alessandro Aldiniand Roberto Gorrieri (Eds.), Vol. 4677. Springer, Heidelberg, 1–23.

Martín Abadi and Bruno Blanchet. 2005a. Analyzing security protocols with secrecy types and logic programs. *Journal of the ACM* 52, 1 (Jan. 2005), 102–146.

Martín Abadi and Bruno Blanchet. 2005b. Computer-assisted verification of a protocol for certified email. *Science of Computer Programming* 58, 1–2 (Oct. 2005), 3–27. Special issue SAS'03.

Martín Abadi, Bruno Blanchet, and Hubert Comon-Lundh. 2009. Models and proofs of protocol security: A progress report. In *Computer Aided Verification, 21st International Conference (Lecture Notes in Computer Science)*, Ahmed Bouajjani and Oded Maler (Eds.), Vol. 5643. Springer, Heidelberg, 35–49.

Martín Abadi, Bruno Blanchet, and Cédric Fournet. 2007. Just Fast Keying in the pi calculus. *ACM Transactions on Information and System Security* 10, 2 (2007), 1–59.

Martín Abadi and Véronique Cortier. 2006. Deciding knowledge in security protocols under equational theories. *Theoretical Computer Science* 367, 1–2 (Nov. 2006), 2–32.

Martín Abadi, Cédric Fournet, and Georges Gonthier. 1998. Secure implementation of channel abstractions. In *Proceedings of the 13th Annual IEEE Symposium on Logic in Computer Science*. IEEE Computer Society, Los Alamitos, CA, 105–116.

Martín Abadi, Cédric Fournet, and Georges Gonthier. 2000. Authentication primitives and their compilation. In *Proceedings of the 27th ACM Symposium on Principles of Programming Languages*. ACM Press, New York, 302–315.

Martín Abadi and Andrew D. Gordon. 1999. A calculus for cryptographic protocols: The spi calculus. *Information and Computation* 148, 1 (Jan. 1999), 1–70. An extended version appeared as Digital Equipment Corporation Systems Research Center report No. 149, January 1998.

Martín Abadi and Phillip Rogaway. 2002. Reconciling two views of cryptography (the computational soundness of formal encryption). *Journal of Cryptology* 15, 2 (2002), 103–127.

David Adrian, Karthikeyan Bhargavan, Zakir Durumeric, Pierrick Gaudry, Matthew Green, J. Alex Halderman, Nadia Heninger, Drew Springall, Emmanuel Thomé, Luke Valenta, Benjamin VanderSloot, Eric Wustrow, Santiago Zanella-Béguelin, and Paul Zimmermann. 2015. Imperfect forward secrecy: How Diffie-Hellman fails in practice. In *ACM SIGSAC Conference on Computer and Communications Security (CCS'15)*. ACM Press, New York, 5–17.

W. Aiello, S. M. Bellovin, M. Blaze, R. Canetti, J. Ionnidis, A. D. Keromytis, and O. Reingold. 2004. Just Fast Keying: Key agreement in a hostile Internet. *ACM Transactions on Information and System Security* 7, 2 (May 2004), 1–30.

Xavier Allamigeon and Bruno Blanchet. 2005. Reconstruction of attacks against cryptographic protocols. In *18th IEEE Computer Security Foundations Workshop (CSFW'05)*. IEEE Computer Society, Los Alamitos, CA, 140–154.

Roberto M. Amadio and Denis Lugiez. 2000. On the reachability problem in cryptographic protocols. In *CONCUR 2000: Concurrency Theory (11th International Conference) (Lecture Notes in Computer Science)*, Catuscia Palamidessi (Ed.), Vol. 1877. Springer, Heidelberg, 380–394.

Myrto Arapinis, Jia Liu, Eike Ritter, and Mark Ryan. 2014. Stateful applied pi calculus. In *Principles of Security and Trust—Third International Conference (Lecture Notes in Computer Science)*, Martín Abadi and Steve Kremer (Eds.), Vol. 8414. Springer, Heidelberg, 22–41.

Myrto Arapinis, Eike Ritter, and Mark Dermot Ryan. 2011. StatVerif: Verification of stateful processes. In *24th IEEE Computer Security Foundations Symposium*. IEEE Computer Society, Los Alamitos, CA, 33–47.

Alessandro Armando, David Basin, Yohan Boichut, Yannick Chevalier, Luca Compagna, Jorge Cuellar, Paul Hankes Drielsma, Pierre-Cyrille Héam, Olga Kouchnarenko, Jacopo Mantovani, Sebastian Mödersheim, David von Oheimb, Michaël Rusinowitch, Judson Santiago, Mathieu Turuani, Luca Viganó, and Laurent Vigneron. 2005. The AVISPA tool for automated validation of Internet security protocols and applications. In *Computer Aided Verification, 17th International Conference (CAV'05) (Lecture Notes in Computer Science)*, Kousha Etessami and Sriram K. Rajamani (Eds.), Vol. 3576. Springer, Heidelberg, 281–285.

Nimrod Aviram, Sebastian Schinzel, Juraj Somorovsky, Nadia Heninger, Maik Dankel, Jens Steube, Luke Valenta, David Adrian, J. Alex Halderman, Viktor Dukhovni, Emilia Käsper, Shaanan Cohney, Susanne Engels, Christof Paar, and Yuval Shavitt. 2016. DROWN: Breaking TLS using SSLv2. In *USENIX Security Symposium*. USENIX, Berkeley, CA, 689–706.

Michael Backes, Dennis Hofheinz, and Dominique Unruh. 2009. CoSP: A general framework for computational soundness proofs. In *16th ACM Conference on Computer and Communications Security*. ACM Press, New York, 66–78.

Michael Backes, Matteo Maffei, and Dominique Unruh. 2008. Zero-knowledge in the applied pi-calculus and automated verification of the direct anonymous attestation protocol. In *IEEE Symposium on Security and Privacy (S&P'08)*. IEEE Computer Society, Los Alamitos, CA, 202–215.

Michael Baldamus, Joachim Parrow, and Björn Victor. 2004. Spi calculus translated to pi-calculus preserving may-tests. In *19th Annual IEEE Symposium on Logic in Computer Science*. IEEE Computer Society, Los Alamitos, CA, 22–31.

Chetan Bansal, Karthikeyan Bhargavan, and Sergio Maffeis. 2012. Discovering concrete attacks on website authorization by formal analysis. In *25th IEEE Computer Security Foundations Symposium*. IEEE Computer Society, Los Alamitos, CA, 247–262.

Gilles Barthe, Benjamin Grégoire, and Santiago Zanella Béguelin. 2010. Programming language techniques for cryptographic proofs. In *Interactive Theorem Proving, 1st International Conference (Lecture Notes in Computer Science)*, Matt Kaufmann and Lawrence C. Paulson (Eds.), Vol. 6172. Springer, Heidelberg, 115–130.

David Basin, Jannik Dreier, and Ralf Casse. 2015. Automated symbolic proofs of observational equivalence. In *22nd ACM Conference on Computer and Communications Security (CCS'15)*. ACM, New York, 1144–1155.

Mathieu Baudet. 2005. Deciding security of protocols against off-line guessing attacks. In *Proceedings of the 12th ACM Conference on Computer and Communications Security (CCS'05)*. ACM Press, New York, 16–25.

Mathieu Baudet. 2007. *Sécurité Des Protocoles Cryptographiques: Aspects Logiques et Calculatoires*. Ph.D. Dissertation. Ecole Normale Supérieure de Cachan.

Mathieu Baudet, Véronique Cortier, and Stéphanie Delaune. 2009a. YAPA: A generic tool for computing intruder knowledge. In *Rewriting Techniques and Applications (RTA'09) (Lecture Notes in Computer Science)*, Ralf Treinen (Ed.), Vol. 5595. Springer, Heidelberg, 148–163. http://dx.doi.org/10.1007/978-3-642-02348-4_11.

Mathieu Baudet, Véronique Cortier, and Steve Kremer. 2009b. Computationally sound implementations of equational theories against passive adversaries. *Information and Computation* 207, 4 (2009), 496–520.

Mihir Bellare and Phillip Rogaway. 1993. Entity authentication and key distribution. In *Advances in Cryptology (CRYPTO'94) (Lecture Notes in Computer Science)*, Vol. 773. Springer, Heidelberg, 232–249.

Jesper Bengtson, Magnus Johansson, Joachim Parrow, and Björn Victor. 2011. Psi-calculi: A framework for mobile processes with nominal data and logic. *Logical Methods in Computer Science* 7, 1, Article 11 (2011), 44 pages.

Gérard Berry and Gérard Boudol. 1992. The chemical abstract machine. *Theoretical Computer Science* 96, 1 (April 1992), 217–248.

Karthikeyan Bhargavan, Bruno Blanchet, and Nadim Kobeissi. 2017a. Verified models and reference implementations for the TLS 1.3 standard candidate. In *IEEE Symposium on Security and Privacy (S&P'17)*. IEEE, Los Alamitos, CA, 483–503.

Karthikeyan Bhargavan, Ricardo Corin, Cédric Fournet, and Eugen Zălinescu. 2008a. Cryptographically verified implementations for TLS. In *15th ACM Conference on Computer and Communications Security (CCS'08)*. ACM, New York, 459–468.

Karthikeyan Bhargavan, Antoine Delignat-Lavaud, Cédric Fournet, Markulf Kohlweiss, Jianyang Pan, Jonathan Protzenko, Aseem Rastogi, Nikhil Swamy, Santiago Zanella-Béguelin, and Jean Karim Zinzindohoué. 2017b. Implementing and proving the TLS 1.3 record layer. In *IEEE Symposium on Security and Privacy (S&P'17)*. IEEE, Los Alamitos, CA, 463–482.

Karthikeyan Bhargavan, Antoine Delignat-Lavaud, Cédric Fournet, Alfredo Pironti, and Pierre-Yves Strub. 2014a. Triple handshakes and cookie cutters: Breaking and fixing authentication over TLS. In *IEEE Symposium on Security and Privacy (S&P'14)*. IEEE Computer Society, Los Alamitos, CA, 98–113.

Karthikeyan Bhargavan, Cédric Fournet, Ricardo Corin, and Eugen Zălinescu. 2012. Verified cryptographic implementations for TLS. *ACM TOPLAS* 15, 1, Article 3 (2012), 32 pages.

Karthikeyan Bhargavan, Cédric Fournet, Andrew D. Gordon, and Riccardo Pucella. 2003. TulaFale: A security tool for web services. In *Formal Methods for Components and Objects (FMCO'03) (Lecture Notes in Computer Science)*, Frank S. de Boer, Marcello M. Bonsangue, Susanne Graf, and Willem-Paul de Roever (Eds.), Vol. 3188. Springer, Heidelberg, 197–222.

Karthikeyan Bhargavan, Cédric Fournet, Andrew D. Gordon, and Stephen Tse. 2008b. Verified interoperable implementations of security protocols. *ACM Transactions on Programming Languages and Systems* 31, 1, Article 5 (Dec. 2008), 61 pages.

Karthikeyan Bhargavan, Cédric Fournet, Markulf Kohlweiss, Alfredo Pironti, Pierre-Yves Strub, and Santiago Zanella-Béguelin. 2014b. Proving the TLS handshake secure (as it is). In *Advances in Cryptology (CRYPTO'14) (Lecture Notes in Computer Science)*, Vol. 8617. Springer, Heidelberg, 235–255.

Karthikeyan Bhargavan, Cédric Fournet, Markulf Kohlweiss, Alfredo Pironti, and Pierre-Yves Strub. 2013. Implementing TLS with verified cryptographic security. In *IEEE Symposium on Security and Privacy (S&P'13)*. IEEE Computer Society, Los Alamitos, CA, 445–459.

Bruno Blanchet. 2001. An efficient cryptographic protocol verifier based on Prolog rules. In *14th IEEE Computer Security Foundations Workshop*. IEEE Computer Society, Los Alamitos, CA, 82–96.

Bruno Blanchet. 2004. Automatic proof of strong secrecy for security protocols. In *2004 IEEE Symposium on Security and Privacy*. IEEE Computer Society, Los Alamitos, CA, 86–100.

Bruno Blanchet. 2006. A computationally sound mechanized prover for security protocols. In *IEEE Symposium on Security and Privacy (S&P'06)*. IEEE Computer Society, Los Alamitos, CA, 140–154.

Bruno Blanchet. 2009. Automatic verification of correspondences for security protocols. *Journal of Computer Security* 17, 4 (July 2009), 363–434.

Bruno Blanchet. 2016. Modeling and verifying security protocols with the applied pi calculus and ProVerif. *Foundations and Trends in Privacy and Security* 1, 1–2 (Oct. 2016), 1–135.

Bruno Blanchet, Martín Abadi, and Cédric Fournet. 2008. Automated verification of selected equivalences for security protocols. *Journal of Logic and Algebraic Programming* 75, 1 (Feb.–March 2008), 3–51.

Bruno Blanchet and Benjamin Aziz. 2003. A calculus for secure mobility. In *8th Asian Computing Science Conference (ASIAN'03) (Lecture Notes in Computer Science)*, Vijay Saraswat (Ed.), Vol. 2896. Springer, Heidelberg, 188–204.

Bruno Blanchet and David Pointcheval. 2006. Automated security proofs with sequences of games. In *Advances in Cryptology (CRYPTO'06) (Lecture Notes in Computer Science)*, Vol. 4117. Springer, Heidelberg, 537–554.

Chiara Bodei, Pierpaolo Degano, Flemming Nielson, and Hanne Riis Nielson. 1998. Control flow analysis for the pi-calculus. In *CONCUR'98: Concurrency Theory (9th International Conference) (Lecture Notes in Computer Science)*, Davide Sangiorgi and Robert de Simone (Eds.), Vol. 1466. Springer, Heidelberg, 84–98.

Michele Boreale, Rocco De Nicola, and Rosario Pugliese. 1999. Proof techniques for cryptographic processes. In *Proceedings of the 14th Annual IEEE Symposium on Logic in Computer Science*. IEEE Computer Society, Los Alamitos, CA, 157–166.

Johannes Borgström, Ramunas Gutkovas, Joachim Parrow, Björn Victor, and Johannes Åman Pohjola. 2014. A sorted semantic framework for applied process calculi (extended abstract). In *Trustworthy Global Computing (TGC'13) (Lecture Notes in Computer Science)*, Martín Abadi and Alberto Lluch Lafuente (Eds.), Vol. 8358. Springer, Cham, 103–118.

Johannes Borgström, Ramunas Gutkovas, Joachim Parrow, Björn Victor, and Johannes Aman Pohjola. 2016. A sorted semantic framework for applied process calculi. *Logical Methods in Computer Science* 12, 1, Article 8 (March 2016), 49 pages.

Sébastien Briais. 2008. *Theory and Tool Support for the Formal Verification of Cryptographic Protocols*. Ph.D. Dissertation. École Polytechnique Fédérale de Lausanne.

Maria Grazia Buscemi and Ugo Montanari. 2007. CC-Pi: A constraint-based language for specifying service level agreements. In *Programming Languages and Systems, 16th European Symposium on Programming (ESOP'07) (Lecture Notes in Computer Science)*, Rocco De Nicola (Ed.), Vol. 4421. Springer, Berlin, 18–32.

Marco Carbone and Sergio Maffeis. 2003. On the expressive power of polyadic synchronisation in pi-calculus. *Nordic Journal of Computing* 10, 2 (2003), 70–98.

Luca Cardelli. 2000. Mobility and security. In *Foundations of Secure Computation (NATO Science Series)*, F. L. Bauer and R. Steinbrueggen (Eds.). IOS Press, Amsterdam, 3–37.

Luca Cardelli and Andrew D. Gordon. 2000. Mobile ambients. *Theoretical Computer Science* 240, 1 (June 2000), 177–213.

Rohit Chadha, Stefan Ciobaca, and Steve Kremer. 2012. Automated verification of equivalence properties of cryptographic protocols. In *Programming Languages and Systems - 21st European Symposium on Programming (ESOP'12) (Lecture Notes in Computer Science)*, Helmut Seidl (Ed.), Vol. 7211. Springer, Heidelberg, 108–127.

Vincent Cheval, Véronique Cortier, and Stéphanie Delaune. 2013. Deciding equivalence-based properties using constraint solving. *Theoretical Computer Science* 492 (June 2013), 1–39.

Rémy Chrétien, Véronique Cortier, and Stéphanie Delaune. 2015a. Decidability of trace equivalence for protocols with nonces. In *28th IEEE Computer Security Foundations Symposium (CSF'15)*. IEEE Computer Society, Los Alamitos, CA, 170–184. DOI: http://dx.doi.org/10.1109/CSF.2015.19

Rémy Chrétien, Véronique Cortier, and Stéphanie Delaune. 2015b. From security protocols to pushdown automata. *ACM Transactions on Computational Logic* 17, 1, Article 3 (Sept. 2015), 45 pages. DOI: http://dx.doi.org/10.1145/2811262

Ştefan Ciobâcă, Stéphanie Delaune, and Steve Kremer. 2012. Computing knowledge in security protocols under convergent equational theories. *Journal of Automated Reasoning* 48, 2 (Feb. 2012), 219–262. DOI: http://dx.doi.org/10.1007/s10817-010-9197-7

Hubert Comon-Lundh and Véronique Cortier. 2008. Computational soundness of observational equivalence. In *Proceedings of the 15th ACM Conference on Computer and Communications Security*. ACM Press, New York, 109–118.

Sylvain Conchon and Fabrice Le Fessant. 1999. Jocaml: Mobile Agents for Objective-Caml. In *1st International Symposium on Agent Systems and Applications (ASA'99)/3rd International Symposium on Mobile Agents (MA'99)*. IEEE Computer Society, 22–29.

Core SDI S.A.1998. SSH insertion attack. *Bugtraq mailing list*. (June 1998). Retrieved from http://seclists.org/bugtraq/1998/Jun/65.

Jean-Sébastien Coron, Yevgeniy Dodis, Cécile Malinaud, and Prashant Puniya. 2005. Merkle-Damgård revisited: How to construct a hash function. In *Advances in Cryptology (CRYPTO'05) (Lecture Notes in Computer Science)*, Vol. 3621. Springer, Heidelberg, 430–448.

Véronique Cortier and Steve Kremer. 2014. Formal models and techniques for analyzing security protocols: A tutorial. *Foundations and Trends in Programming Languages* 1, 3 (2014), 151–267.

Cas Cremers, Marko Horvat, Sam Scott, and Thyla van der Merwe. 2016. Automated analysis and verification of TLS 1.3: 0-RTT, resumption and delayed authentication. In *IEEE Symposium on Security and Privacy (S&P'16)*. IEEE Computer Society, Los Alamitos, CA, 470–485.

Cas J. F. Cremers. 2008. Unbounded verification, falsification, and characterization of security protocols by pattern refinement. In *15th ACM Conference on Computer and Communications Security (CCS'08)*. ACM Press, New York, 119–128.

Luís Cruz-Filipe, Ivan Lanese, Francisco Martins, António Ravara, and Vasco Thudichum Vasconcelos. 2014. The stream-based service-centered calculus: A foundation for service-oriented programming. *Formal Aspects of Computing* 26, 5 (2014), 865–918.

M. Curti, P. Degano, C. Priami, and C. T. Baldari. 2004. Modelling biochemical pathways through enhanced π-calculus. *Theoretical Computer Science* 325 (2004), 111–140.

Mads Dam. 1998. Proving trust in systems of second-order processes. In *Proceedings of the 31th Hawaii International Conference on System Sciences*, Vol. VII. IEEE Computer Society, Los Alamitos, CA, 255–264.

Anupam Datta, Ante Derek, John C. Mitchell, and Dusko Pavlovic. 2005. A derivation system and compositional logic for security protocols. *Journal of Computer Security* 13, 3 (2005), 423–482.

Stéphanie Delaune, Steve Kremer, and Mark D. Ryan. 2007. *Symbolic Bisimulation for the Applied Pi Calculus*. Research Report LSV-07-14. LSV, ENS Cachan.

Stéphanie Delaune, Steve Kremer, and Mark D. Ryan. 2009. Verifying privacy-type properties of electronic voting protocols. *Journal of Computer Security* 17, 4 (July 2009), 435–487. DOI: http://dx.doi.org/10.3233/JCS-2009-0340

Stéphanie Delaune, Steve Kremer, and Mark D. Ryan. 2010. Symbolic bisimulation for the applied pi calculus. *Journal of Computer Security* 18, 2 (2010), 317–377.

Richard A. DeMillo, Nancy A. Lynch, and Michael Merritt. 1982. Cryptographic protocols. In *Proceedings of the 14th Annual ACM Symposium on Theory of Computing*. ACM Press, New York, 383–400.

T. Dierks and E. Rescorla. 2008. The Transport Layer Security (TLS) Protocol Version 1.2. *IETF RFC 5246*. (2008).

W. Diffie and M. Hellman. 1976. New directions in cryptography. *IEEE Transactions on Information Theory* IT-22, 6 (Nov. 1976), 644–654.

Whitfield Diffie, Paul C. van Oorschot, and Michael J. Wiener. 1992. Authentication and authenticated key exchanges. *Designs, Codes and Cryptography* 2 (1992), 107–125.

Danny Dolev and Andrew C. Yao. 1983. On the security of public key protocols. *IEEE Transactions on Information Theory* IT-29, 12 (March 1983), 198–208.

Santiago Escobar, Catherine Meadows, and José Meseguer. 2006. A rewriting-based inference system for the NRL protocol analyzer and its meta-logical properties. *Theoretical Computer Science* 367, 1–2 (2006), 162–202.

Cédric Fournet and Georges Gonthier. 1998. A hierarchy of equivalences for asynchronous calculi. In *Proceedings of the 25th International Colloquium on Automata, Languages and Programming (Lecture Notes in Computer Science)*, Kim G. Larsen, Sven Skyum, and Glynn Winskel (Eds.), Vol. 1443. Springer, Heidelberg, 844–855.

Alan O. Freier, Philip Karlton, and Paul C. Kocher. November 1996. The SSL Protocol: Version 3.0. (November 1996). Internet Draft retrieved from http://tools.ietf.org/html/draft-ietf-tls-ssl-version3-00.

Shafi Goldwasser and Mihir Bellare. 1999. Lecture Notes on Cryptography. Summer Course "Cryptography and Computer Security" at MIT, 1996–1999. (Aug. 1999).

Shafi Goldwasser and Silvio Micali. 1984. Probabilistic encryption. *Journal on Computer System Science* 28 (April 1984), 270–299.

Shafi Goldwasser, Silvio Micali, and Ronald Rivest. 1988. A digital signature scheme secure against adaptive chosen-message attack. *SIAM Journal on Computing* 17 (1988), 281–308.

Andrew Gordon and Alan Jeffrey. 2004. Types and effects for asymmetric cryptographic protocols. *Journal of Computer Security* 12, 3/4 (2004), 435–484.

Daniel Hirschkoff. 1997. A full formalisation of π-calculus theory in the calculus of constructions. In *Theorem Proving in Higher Order Logics (Lecture Notes in Computer Science)*, Elsa L. Gunter and Amy Felty (Eds.), Vol. 1275. Springer, New York, 153–169.

Kohei Honda and Nobuko Yoshida. 1995. On reduction-based process semantics. *Theoretical Computer Science* 151 (1995), 437–486.

Furio Honsell, Marino Miculan, and Ivan Scagnetto. 2001. π-calculus in (co) inductive type theory. *Theoretical Computer Science* 253, 2 (2001), 239–285.

Tibor Jager, Florian Kohlar, Sven Schäge, and Jörg Schwenk. 2012. On the security of TLS-DHE in the standard model. In *CRYPTO'12*. Springer, New York, 273–293.

R. Kemmerer, C. Meadows, and J. Millen. 1994. Three systems for cryptographic protocol analysis. *Journal of Cryptology* 7, 2 (Spring 1994), 79–130.

Hugo Krawczyk. 1996. SKEME: A versatile secure key exchange mechanism for Internet. In *Proceedings of the Internet Society Symposium on Network and Distributed Systems Security*. IEEE Computer Society, Los Alamitos, CA, 114–127.

Hugo Krawczyk, Kenneth G. Paterson, and Hoeteck Wee. 2013. On the security of the TLS protocol: A systematic analysis. In *CRYPTO'13*. Springer, New York, 429–448.

Steve Kremer and Robert Künnemann. 2014. Automated analysis of security protocols with global state. In *IEEE Symposium on Security and Privacy (S&P'14)*. IEEE Computer Society, Los Alamitos, CA, 163–178.

Steve Kremer and Mark D. Ryan. 2005. Analysis of an electronic voting protocol in the applied pi calculus. In *Programming Languages and Systems: 14th European Symposium on Programming (ESOP'05) (Lecture Notes in Computer Science)*, Mooly Sagiv (Ed.), Vol. 3444. Springer, Heidelberg, 186–200.

Alessandro Lapadula, Rosario Pugliese, and Francesco Tiezzi. 2007. A calculus for orchestration of web services. In *Programming Languages and Systems, 16th European Symposium on Programming (ESOP'07) (Lecture Notes in Computer Science)*, Rocco De Nicola (Ed.), Vol. 4421. Springer, Berlin, 33–47.

Ben Liblit and Alexander Aiken. 2000. Type systems for distributed data structures. In *Proceedings of the 27th ACM Symposium on Principles of Programming Languages*. ACM Press, New York, 199–213.

P. Lincoln, J. Mitchell, M. Mitchell, and A. Scedrov. 1998. A probabilistic poly-time framework for protocol analysis. In *Proceedings of the 5th ACM Conference on Computer and Communications Security*. ACM Press, New York, 112–121.

Jia Liu. 2011. A Proof of Coincidence of Labeled Bisimilarity and Observational Equivalence in Applied Pi Calculus. http://lcs.ios.ac.cn/~jliu/papers/LiuJia0608.pdf. (2011).

Jia Liu and Humin Lin. 2012. A complete symbolic bisimulation for full applied pi calculus. *Theoretical Computer Science* 458 (Nov. 2012), 76–112.

Gavin Lowe. 1996. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In *Tools and Algorithms for the Construction and Analysis of Systems (Lecture Notes in Computer Science)*, Vol. 1055. Springer, Heidelberg, 147–166.

Roberto Lucchi and Manuel Mazzara. 2007. A pi-calculus based semantics for WS-BPEL. *Journal of Logic and Algebraic Programming* 70 (2007), 96–118.

Joana Martinho and António Ravara. 2011. Encoding cryptographic primitives in a calculus with polyadic synchronisation. *Journal of Automated Reasoning* 46, 3–4 (2011), 293–323.

Simon Meier, Benedikt Schmidt, Cas Cremers, and David A. Basin. 2013. The Tamarin prover for the symbolic analysis of security protocols. In *Computer Aided Verification, 25th International Conference (CAV'13) (Lecture Notes in Computer Science)*, Natasha Sharygina and Helmut Veith (Eds.), Vol. 8044. Springer, Heidelberg, 696–701.

Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. 1996. *Handbook of Applied Cryptography*. CRC Press, Boca Raton, FL.

Michael J. Merritt. 1983. *Cryptographic Protocols*. Ph.D. Dissertation. Georgia Institute of Technology.

Robin Milner. 1989. *Communication and Concurrency*. Prentice Hall, Upper Saddle River, NJ.

Robin Milner. 1992. Functions as processes. *Mathematical Structures in Computer Science* 2 (1992), 119–141.

Robin Milner. 1999. *Communicating and Mobile Systems: The π-Calculus*. Cambridge University Press, Cambridge.

Robin Milner and Davide Sangiorgi. 1992. Barbed bisimulation. In *Automata, Languages and Programming: 19th International Colloquium Wien, Austria, July 13–17, 1992 Proceedings*, W. Kuich (Ed.). Springer, Berlin, 685–695. DOI: http://dx.doi.org/10.1007/3-540-55719-9_114

John C. Mitchell. 1996. *Foundations for Programming Languages*. MIT Press, Cambridge, MA.

John C. Mitchell, Mark Mitchell, and Ulrich Stern. 1997. Automated analysis of cryptographic protocols using Murϕ. In *Proceedings of the 1997 IEEE Symposium on Security and Privacy*. IEEE Computer Society, Los Alamitos, CA, 141–151.

Kenneth G. Paterson, Thomas Ristenpart, and Thomas Shrimpton. 2011. Tag size does matter: Attacks and proofs for the TLS record protocol. In *ASIACRYPT*. Springer, Berlin, 372–389.

Lawrence C. Paulson. 1998. The inductive approach to verifying cryptographic protocols. *Journal of Computer Security* 6, 1–2 (1998), 85–128.

Andreas Pfitzmann and Marit Köhntopp. 2001. Anonymity, unobservability, and pseudonymity – A proposal for terminology. In *International Workshop on Design Issues in Anonymity and Unobservability (Lecture Notes in Computer Science)*, Vol. 2009. Springer, New York, 1–9. Extended versions available at http://dud.inf.tu-dresden.de/Anon_Terminology.shtml.

Birgit Pfitzmann, Matthias Schunter, and Michael Waidner. 2000. Cryptographic security of reactive systems (extended abstract). *Electronic Notes in Theoretical Computer Science* 32 (April 2000), 59–77.

Benjamin C. Pierce and David N. Turner. 2000. Pict: A programming language based on the pi-calculus. In *Proof, Language and Interaction: Essays in Honour of Robin Milner (Foundations of Computing)*, Gordon Plotkin, Colin Stirling, and Mads Tofte (Eds.). MIT Press, Cambridge, MA, 455–494.

Mark D. Ryan and Ben Smyth. 2011. Applied pi calculus. In *Formal Models and Techniques for Analyzing Security Protocols*, Véronique Cortier and Steve Kremer (Eds.). IOS Press, Amsterdam, 112–142. http://www.bensmyth.com/files/Smyth10-applied-pi-calculus.pdf.

Peter Y. A. Ryan and Steve A. Schneider. 1998. An attack on a recursive authentication protocol. a cautionary tale. *Information Processing Letters* 65, 1 (Jan. 1998), 7–10.

Davide Sangiorgi. 1993. *Expressing Mobility in Process Algebras: First-Order and Higher-Order Paradigms*. Ph.D. Dissertation. University of Edinburgh.

D. Sangiorgi. 1998. On the bisimulation proof method. *Journal of Mathematical Structures in Computer Science* 8 (1998), 447–479.

Sonia Santiago, Santiago Escobar, Catherine Meadows, and José Meseguer. 2014. A formal definition of protocol indistinguishability and its verification using Maude-NPA. In *Security and Trust Management (STM'14)* (Lecture Notes in Computer Science), Sjouke Mauw and Christian Damsgaard Jensen (Eds.), Vol. 8743. Springer, Heidelberg, 162–177.

Benedikt Schmidt, Simon Meier, Cas Cremers, and David Basin. 2012. Automated analysis of Diffie-Hellman protocols and advanced security properties. In *25th IEEE Computer Security Foundations Symposium (CSF'12)*. IEEE Computer Society, Los Alamitos, CA, 78–94.

Steve Schneider. 1996. Security properties and CSP. In *Proceedings of the 1996 IEEE Symposium on Security and Privacy*. IEEE Computer Society, Los Alamitos, CA, 174–187.

Bruce Schneier. 1996. *Applied Cryptography: Protocols, Algorithms, and Source Code in C (2nd ed.)*. John Wiley & Sons, Hoboken, NJ.

Stuart G. Stubblebine and Virgil D. Gligor. 1992. On message integrity in cryptographic protocols. In *Proceedings of the 1992 IEEE Symposium on Research in Security and Privacy*. IEEE Computer Society, Los Alamitos, CA, 85–104.

F. Javier Thayer Fábrega, Jonathan C. Herzog, and Joshua D. Guttman. 1998. Strand spaces: Why is a security protocol correct? In *Proceedings of the 1998 IEEE Symposium on Security and Privacy*. IEEE Computer Society, Los Alamitos, CA, 160–171.

Alwen Tiu and Jeremy Dawson. 2010. Automating open bisimulation checking for the spi-calculus. In *23rd IEEE Computer Security Foundations Symposium (CSF'10)*. IEEE Computer Society, Los Alamitos, CA, 307–321.

Mathy Vanhoef and Frank Piessens. 2015. All your biases belong to us: Breaking RC4 in WPA-TKIP and TLS. In *USENIX Security Symposium*. USENIX, Berkeley, CA, 97–112.

Björn Victor. 1998. *The Fusion Calculus: Expressiveness and Symmetry in Mobile Processes*. Ph.D. Dissertation. Dept. of Computer Systems, Uppsala University, Sweden.

Peter H. Welch and Frederick R. M. Barnes. 2005. Communicating mobile processes: Introducing occam-pi. In *Communicating Sequential Processes. The First 25 Years (Lecture Notes in Computer Science)*, Ali E. Abdallah, Cliff B. Jones, and Jeff W. Sanders (Eds.), Vol. 3525. Springer, Berlin, 175–210.

Lucian Wischik and Philippa Gardner. 2004. Strong bisimulation for the explicit fusion calculus. In *Foundations of Software Science and Computation Structures, 7th International Conference (FOSSACS'04) (Lecture Notes in Computer Science)*, Igor Walukiewicz (Ed.), Vol. 2987. Springer, Berlin, 484–498.

Andrew C. Yao. 1982. Theory and applications of trapdoor functions. In *Proceedings of the 23rd Annual Symposium on Foundations of Computer Science (FOCS'82)*. IEEE Computer Society, Los Angeles, CA, 80–91.

Received October 2015; revised July 2017; accepted July 2017