Cross-Tool Semantics for Protocol Security Goals

Joshua D. Guttman, John D. Ramsdell, and Paul D. Rowe
{guttman,ramsdell,prowe}@mitre.org

The MITRE Corporation

Abstract. Formal protocol analysis tools provide objective evidence that a protocol under standardization meets security goals, as well as counterexamples to goals it does not meet ("attacks"). Different tools are however based on different execution semantics and adversary models. If different tools are applied to alternative protocols under standardization, can formal evidence offer a yardstick to compare the results?

We propose a family of languages within first order predicate logic to formalize protocol safety goals (rather than indistinguishability). Although they were originally designed for the strand space formalism that supports the tool CPSA, we show how to translate them to goals for the applied π calculus that supports the tool ProVerif. We give a criterion for protocols expressed in the two formalisms to correspond, and prove that if a protocol in the strand space formalism satisfies a goal, then a corresponding applied π process satisfies the translation of that goal. We show that the converse also holds for a class of goal formulas, and conjecture a broader equivalence. We also describe a compiler that, from any protocol in the strand space formalism, constructs a corresponding applied π process and the relevant goal translation.

1 Introduction

Automated tools for analyzing cryptographic protocols have proven quite effective at finding flaws and verifying that proposed mitigations satisfy desirable properties. Recent efforts to apply these tools to protocols approved by standards bodies has led Basin et al. [5] to stress the importance of publishing the underlying threat models and desired security goals as part of the standard. This advice is in line with the ISO standard, ISO/IEC 29128 "Verification of Cryptographic Protocols," [22] which codifies a framework for certifying the design of cryptographic protocols. There are three key aspects to this framework (described in [23]). It calls for explicit (semi-)formal descriptions of the protocol, adversary model, and security properties to be achieved. One final aspect is the production of self-assessment evidence that the protocol achieves the stated goals with respect to the stated adversary model. This fourth aspect is critical. It increases transparency by allowing practitioners the ability to independently inspect and verify the evidence. So, for example, if the evidence is the input/output of some analysis tool, the results could be replicated by re-running the tool.

An interesting situation may arise when two different tools are used to evaluate the same protocol. For example, in 1999, Meadows [25] found weaknesses in the Internet Key Exchange (IKE) protocol using the NRL Protocol Analyzer [24], while in 2011

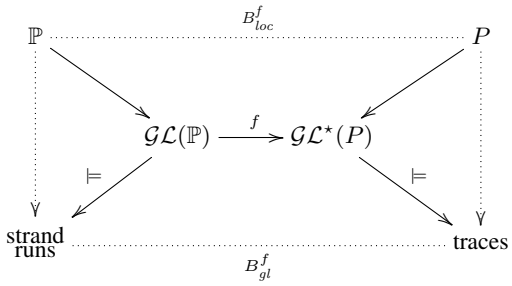

Fig. 1. Consistency of cross-tool semantics

Cremers [16] found additional flaws using Scyther [15]. In such situations it can be quite difficult to determine exactly what the cause of the difference is. Small differences in any of the first three aspects of the framework could result in important differences in the conclusions drawn. This has the potential to undermine some of the transparency gained by including the self-assessment evidence to begin with.

Ideally the first three aspects of the assessment framework (i.e. protocol description, adversary model, and protocol goals) could be described rigorously in a manner that is independent of the tool or underlying formalism used to verify them. For example, many tools assume a so-called Dolev-Yao adversary model. Although some details vary depending on which cryptographic primitives are being considered, there is generally a common understanding of what is involved in this adversary model. However, this is typically not the case for the other aspects. The particular syntax for describing a protocol is closely tied to the underlying semantics which is entirely tool-dependent. Similarly, security goals are frequently expressed in a stylized manner that is tightly coupled to the tool or underlying formalism. It is this last point that we begin to address in this work, by demonstrating how a particular language of security goals can have a consistent interpretation in two chosen tools: CPSA and ProVerif.

We adopt a security goal language $\mathcal{GL}$ for safety properties first introduced in the strand space context [21]. $\mathcal{GL}$ contains both protocol-specific and -independent vocabulary. Security goals take the so-called "geometric" form:

$$\forall \overline{x} \,.\, \Phi \Longrightarrow \Psi$$

where Φ, Ψ are built from atomic formulas using conjunction, disjunction, and existential quantification. Any protocol $\mathbb{P}$ induces the protocol-specific language $\mathcal{GL}(\mathbb{P})$. It was designed with limited expressivity in order to capture security goals that are preserved by a class of protocol transformations. The limited expressivity is advantageous for the current work because $\mathcal{GL}$ talks only about events, message parameters and the relevant relations among them. While some tools may represent and reason about more types of events than others, there is a common core set of events such as message transmission and reception that every tool must represent and reason about. As a consequence, all statements of security goals related to this core set of events and parameters are independent from the particular formalism that might be used to verify them.

In this paper we aim to demonstrate how to cross-validate results between the two tools CPSA [27] and ProVerif [7]. This requires us to demonstrate how to *consistently* interpret goal formulas relative to the underlying formalisms used by the two tools, in this case strand spaces and the applied π calculus respectively. Figure 1 diagrammatically depicts what is required for the consistency of such cross-tool semantics.

We draw the reader's attention to several aspects of this diagram. First, the two triangles represent standalone logical semantics for the goal language $\mathcal{GL}$ with respect to each of the execution semantics of the two tools. In Section 3 we describe the left triangle: the strand space semantics for CPSA of $\mathcal{GL}(\mathbb{P})$ relative to a notion of executions we call "strand runs." We cover the right triangle in Section 4 by giving the semantics of $\mathcal{GL}^\star(P)$ relative to a trace execution semantics of the applied π calculus for ProVerif.

We explicitly depict two different logical languages $\mathcal{GL}(\mathbb{P})$ and $\mathcal{GL}^\star(P)$ because applied π processes P can represent strictly more events than strand spaces. In particular, P may represent the *internal* events required to parse received messages. Thus we imagine an embedding $f : \mathcal{GL}(\mathbb{P}) \to \mathcal{GL}^\star(P)$ on the goal language. We can therefore only hope to get consistent answers from CPSA and ProVerif on goals expressible in $\mathcal{GL}(\mathbb{P})$, or equivalently, its image under f.

Of course, if the corresponding predicates of $\mathcal{GL}(\mathbb{P})$ and $\mathcal{GL}^\star(P)$ refer to essentially different things, we cannot expect consistent results. In Section 5 we present a relation $B^f_{loc}(\mathbb{P}, P)$ that characterizes when a protocol $\mathbb{P}$ and process term P represent the "same thing." The idea is to ensure that the two formulations of each of the roles are locally bisimilar. Here again the applied π calculus is generally more expressive than strand spaces, so we actually describe a *compiler* that transforms a strand space protocol $\mathbb{P}$ into a bisimilar process term P.

Finally, in Section 6 we demonstrate how the local bisimulation B^f_{loc} lifts to a global bisimulation B^f_{gl} on the operational semantics of the two sides. We then show that this bisimulation respects security goals in the sense that any goal satisfied on the left by a strand run is also satisfied on the right by a corresponding trace. The converse cannot be true for all goals because traces are totally ordered whereas strand runs may be only partially ordered. However, we conjecture that for any goal that is insensitive to the *inessential* orderings of a trace, if a trace satisfies the goal then so does a corresponding strand run.

1.1 Related Work

We have described above how this paper connects to the protocol verification framework described by Matsuo et al. [23] and standardized in the ISO in [22]. Although the use of formal logics to express protocol security goals is not new [9,17], our focus on using such a logic to connect distinct verification formalisms seems to be new. There was a lot of work in the early 2000s detailing the connections between the various protocol analysis formalisms being developed at the time [10,6,14,26]. This work tended to focus on connecting the underlying execution semantics of the various formalisms without explicit reference to formal security goals. Thus, in reference to Fig. 1, only the outside edges were described. By filling in the details of the internal connections, explicitly relating the execution semantics to a security goal language, it is easier for a practitioner to understand how the two sides relate.

There have been several related projects that unite a variety of protocol validation tools into a single tool suite. Most notably, the AVISPA [2] and AVANTSSAR [3] projects provide a unified interface to several backend tools. The available toolset seems to be limited to bounded verification, whereas in this paper we connect two formalisms capable of unbounded verification. Their protocol description language, ASLan++, however might be a good candidate for formalism-independent protocol descriptions.

Many tools have also embarked on establishing indistinguishability properties of protocols, also sometimes called privacy-type properties. In this area, logical languages to express goals are less developed. However, we consider this an important area to pursue the present cross-tool logical program also.

2 A Simple Example

In this section we introduce an example protocol, and mention the goals that it achieves. We then show how to formalize the goals it achieves in a first order language introduced for the strand space formalism [21,28].

A Simple Example Protocol. As a minimal example, consider the Simple Example Protocol (SEP) used by Blanchet [8] and many others [13]. In this protocol, an initiator A chooses a session key s, which it signs and then encrypts using the public encryption key of an intended peer B. It then waits to receive in exchange a sensitive payload d, delivered encrypted with s.

$$A \to B : \{|[\![s]\!]_{\mathsf{sk}(A)}|\}_{\mathsf{pk}(B)}$$
$$B \to A : \{|d|\}_s$$

One is traditionally interested in whether confidentiality is assured for d, and whether A authenticates B as the origin of d or B authenticates A as the origin of s. Actually, SEP already indicates why this way of expressing the goals is too crude. In Fig. 2 (a), we show the assumptions needed for a conclusion, from A's point of view, and the conclusion that B behaved according to expectations. That is, the protocol is successful

If A has a run of the protocol apparently with B;
and B's private decryption key $\mathsf{pk}(B)^{-1}$ is uncompromised;
and the session key s is freshly chosen,

then B transmitted d with matching parameters,
and d remains confidential.

(a)

If B has a run of the protocol apparently with A;
and B's private decryption key $\mathsf{pk}(B)^{-1}$ and A's signature key $\mathsf{sk}(A)$ are both uncompromised;
and the session key s and payload d are freshly chosen,
then A took the first step of an initiator session, originating the key s, with some intended peer C.

(b)

Fig. 2. Main goal achieved by SEP from the points of view of each role

from A's point of view. However, the story is different from B's point of view, as shown in Fig. 2 (b). Although B certainly can't know whether A receives the final message, the fact that A's intended peer is some C who may differ from B is troublesome. If C's private decryption key is uncompromised, then the adversary can recover s and A's signature, repackaging them for B, and using s to recover the intended secret d.

The goal language. We wish to express protocol security goals, such as those in Fig. 2, in a language that is independent of the underlying formalism used to verify the goals. We adopt a first order goal language developed in the context of the strand space formalism. It was originally designed by Guttman [21] to limit expressiveness in order to ensure goals in the language are preserved under a certain class of protocol transformations. The limited expressivity was leveraged by Rowe et al. [28] to measure and compare the strength of "related" protocols. We believe the limited expressivity makes it possible for the formal statement of security goals to be independent of any underlying verification methodology or formalism. Although the goal language was originally developed for the strand space formalism and incorporated into CPSA, the main purpose of this paper is to provide a semantics of the language for the applied π calculus that is consistent with the strand space semantics so that it might be used also by ProVerif.

As suggested by the informal goal statements of Fig. 2, the language needs predicates to express how far a principal progressed in a role, the value of parameters used in messages, the freshness of values, and the non-compromise of keys. We explain each of these in turn.

The progress made in a role is expressed with *role position predicates*. For example, predicates of the form $\texttt{InitDone}(n)$ or $\texttt{RespStart}(m)$ say that an initiator has completed its last step, or that a responder has completed its first step. Each role position predicate is a one-place predicate that says what kind of event its argument n, m refers to.

At each point in a role, the agent will have bound some of its local parameters to concrete values. The *parameter predicates* are two place predicates that express this binding. For example, if n refers to an initiator's event, we would use $\texttt{Self}(n, a)$ to express that the initiator's local value for their own identity is a. Similarly, $\texttt{SessKey}(m, s)$ would say that the value bound to the local session key parameter is referred to by s.

The role position predicates and the parameters predicates are *protocol-dependent* in that the length of roles and the parameter bindings at various points depend on the details of the protocol.

The goal language also contains *protocol-independent* predicates that apply to any protocol. These predicates appear in Table 1. They help to express the structural properties of protocol executions. $\texttt{Preceq}(m, n)$ asserts that either m and n represent the same event, or else m occurs before n; $\texttt{Coll}(m, n)$ says that they are both events of the same local session. $m = n$ is satisfied when m and n are equal.

The remaining predicates are used to express that values are fresh or uncompromised. This way of expressing freshness and non-compromise comes from the strand space formalism, but it is possible to make sense of them in any formalism. The idea is to characterize the effects of local choices as they manifest in executions. Randomly chosen values cannot be guessed by the adversary or other participants, so they may only "originate" from the local session in which it is chosen, if at all. We will make the

Functions:	$\texttt{pk}(a)$	$\texttt{sk}(a)$	$\texttt{inv}(k)$
	$\texttt{ltk}(a, b)$		
Relations:	$\texttt{Preceq}(m, n)$	$\texttt{Coll}(m, n)$	$=$
	$\texttt{Unq}(v)$	$\texttt{UnqAt}(n, v)$	$\texttt{Non}(v)$

Table 1. Protocol-independent vocabulary of languages $\mathcal{GL}(\Pi)$

meaning of "origination" more precise for each of the formalisms, but the intuition is that $\texttt{Unq}(v)$ says that v is a randomly chosen value, $\texttt{UnqAt}(n, v)$ specifies the node at which it originates, and $\texttt{Non}(v)$ says that v is never learned by the adversary. Within this language we can formally express the two goals of Fig. 2 as we have done in Fig. 3 for the second goal.

A formal semantics for this language has already been given with respect to the execution model of strand spaces [21]. Our main contribution is to provide a consistent semantics for this language with respect to the execution model of the applied π calculus. To simplify this task we assume that messages have the same representation in both formalisms. We now provide the necessary details of the underlying term algebra for modeling messages.

Term algebra. We will use an order-sorted term algebra to represent the values exchanged in protocols. There is a partial order of *sorts* S ordered by $<$. We assume the existence of a top sort $\top$ that is above all other sorts. We build terms from sorted names and variables. We call $<$-minimal sorts *basic sorts* and terms of those sorts are called *basic values*. The set of names is the disjoint union of names for each basic sort: $\mathcal{N} = \uplus_{\mathsf{s} \in \mathsf{S}} \mathcal{N}_\mathsf{s}$, where $\mathcal{N}_\mathsf{s} = \mathcal{N}_\mathsf{s}^0 \uplus \mathcal{N}_\mathsf{s}^\nu$ is the disjoint union of two sets. We also consider two disjoint sets of variables $\mathcal{X} = \uplus_{\mathsf{s} \in \mathsf{S}} \mathcal{X}_\mathsf{s}$ and $\mathcal{W} = \uplus_{\mathsf{s} \in \mathsf{S}} \mathcal{W}_\mathsf{s}$. Variables in $\mathcal{X}$ will be bound to parts of messages received by protocol participants, while variables in $\mathcal{W}$ will be used by the intruder.

We write $\mathcal{T}(\Sigma, A)$ to denote the set of terms built from set A using signature Σ in the usual way. A term is *ground* if it contains no variables. An *environment* is map from $\mathcal{N} \cup \mathcal{X} \cup \mathcal{W}$ that maps names to names and variables to terms. The result of applying an environment σ to a term u is denoted $\sigma(u)$. We only consider sort-respecting environments in that for every term $u : \mathsf{s}$, $\sigma(u) : \mathsf{s}'$ with $\mathsf{s}' \leq \mathsf{s}$. Environments also respect the difference between $\mathcal{N}_\mathsf{s}^0$ and $\mathcal{N}_\mathsf{s}^\nu$. Environments can be updated so that, for example, $\sigma[x \mapsto v]$ is the environment that maps x to v and otherwise acts like σ. We identify a subset of terms called *messages* by partitioning Σ into constructor sym-

$$\begin{array}{lcl} \forall n, b, a, s, d\,.\, \texttt{RespDone}(n) \wedge \texttt{Self}(n, b) \wedge & & \exists m, c\,.\, \texttt{InitStart}(m) \wedge \texttt{Self}(m, a) \wedge \\ \texttt{Peer}(n, a) \wedge \texttt{SessKey}(n, s) \wedge \texttt{Datum}(n, d) \wedge & \Rightarrow & \texttt{Peer}(m, c) \wedge \texttt{SessKey}(m, s) \wedge \\ \texttt{Non}(\texttt{sk}(b)) \wedge \texttt{Non}(\texttt{sk}(a)) \wedge \texttt{Unq}(s) & & \texttt{UnqAt}(m, s) \wedge \texttt{Preceq}(m, n) \end{array}$$

Fig. 3. Formalized goal achieved by SEP from the responder point of view

bols and destructor symbols, $\Sigma_c \uplus \Sigma_d$, and letting $\mathsf{MSG} = \mathcal{T}(\Sigma_c, \mathcal{N} \cup \mathcal{X})$. These are the terms that are sent and received by protocols. For concreteness assume $\Sigma_c = \{\{\!|\cdot|\!\}^{\mathsf{s}}_{\cdot}, \{\!|\cdot|\!\}^{\mathsf{a}}_{\cdot}, [\![\cdot]\!]_{\cdot}, \cdot\,\hat{}\,\cdot, \mathsf{pk}, \mathsf{sk}, \mathsf{ltk}, (\cdot)^{-1}\}$, and that $\Sigma_d = \{\mathsf{dec}^{\mathsf{s}}, \mathsf{dec}^{\mathsf{a}}, \mathsf{ver}, \mathsf{fst}, \mathsf{snd}\}$.

We say that t_0 is an *ingredient* of t, written $t_0 \sqsubseteq t$, iff either (i) $t_0 = t$; or (ii) $t = t_1\,\hat{}\,t_2$ and $t_0 \sqsubseteq t_1$ or $t_0 \sqsubseteq t_2$; or (iii) $t = \{\!|t_1|\!\}^{*}_{t_2}$ for $* \in \{\mathsf{s}, \mathsf{a}\}$ and $t_0 \sqsubseteq t_1$; or (iv) $t = [\![\, t_1 \,]\!]_{t_2}$ and $t_0 \sqsubseteq t_1$. The key of a cryptographic operation does not contribute to the ingredients of the result; only the plaintext does.

The adversary's ability to derive messages is represented in two equivalent ways. In the first method, we partition Σ into $\Sigma_{\mathsf{pub}} \uplus \Sigma_{\mathsf{priv}}$ and consider a convergent rewrite system with rules $\mathsf{g}(t_1, \ldots, t_n) \to t$ for $\mathsf{g} \in \Sigma_d$. Since the system is convergent, every term t has a normal form denoted $t{\downarrow}$. The set of messages derivable from some set X is thus $nf(\mathcal{T}(\Sigma_{\mathsf{pub}}, X)) \cap \mathsf{MSG}$, where $nf(T)$ produces the set of normal forms of the set of terms T. In the second method, the adversary uses derivability rules of the form $\{t_1, \ldots, t_n\} \vdash t$. The set of messages derivable from some set X is the smallest set containing X and closed under $\vdash$. When each rewrite rule $\mathsf{g}(t_1, \ldots, t_n) \to t$ corresponds to a derivation rule $\{t_1, \ldots, t_n\} \vdash t$ and vice versa, the two notions of derivability coincide.

3 Strand Spaces

In this section we present the syntax and execution semantics of strand spaces and we discuss how the executions furnish semantic models for the formulas of the goal language $\mathcal{GL}(\mathbb{P})$.

Strands. A *strand* is a sequence of transmission and reception events, each of which we will call a *node*. We use strands to represent the behavior of a single principal in a single local protocol session. By convention, we draw strands with double-arrows connecting the successive nodes $\bullet \Rightarrow \bullet$. We use single arrows $\bullet \to \bullet$ to denote the type of node (transmission vs. reception).

We write $+t$ for a node transmitting the term t and $-t$ for a node receiving t, and we write $\mathsf{msg}(n)$ for t if n is a node $\pm t$. We write $\mathsf{dmsg}(n)$ for the pair $\pm\mathsf{msg}(n)$, i.e. the message together with its direction, $+$ or $-$.

If s is a strand, we write $|s|$ for its length, i.e. the number of nodes on s. We use 1-based indexing for strands, writing $s@i$ for its i^{th} node. Thus, the sequence of nodes along s is $\langle s@1, \ldots, s@|s| \rangle$. A message t *originates* at a node $n = s@j$ iff (i) n is a transmission node; (ii) $t \sqsubseteq \mathsf{msg}(n)$; and (iii) t is not an ingredient of any earlier $\mathsf{msg}(m)$ where $m = s@k$ and $k < j$.

Protocols. A *protocol* $\mathbb{P}$ is a finite sequence of strands, called the *roles* of $\mathbb{P}$, together with possibly some auxiliary assumptions (detailed below) about fresh values. Regarding $\mathbb{P}$ as a sequence instead of a set will be convenient in Sec. 5.

The messages sent and received on these strands contain *parameters*, which are the names, nonces, keys, and other data occurring in the messages. The parameters account for the variability between different instances of the roles. More formally, a $\mathbb{P}$-*instance* is a triple consisting of a role $\rho \in \mathbb{P}$, a natural number $h \leq |\rho|$, and an environment σ that assigns messages to precisely those variables and names in ρ

Fig. 4. The SEP protocol.

that occur in its first h nodes. If $\iota = (\rho, h, \sigma)$ is an instance, then the *nodes* of ι are $\mathsf{nodes}(\iota) = \{(\iota, j) \colon 1 \leq j \leq h\}$. The transmission and reception nodes of ι are denoted $\mathsf{nodes}^+(\iota)$ and $\mathsf{nodes}^-(\iota)$ respectively. The message of a node is $\mathsf{msg}((\rho, h, \sigma), j) = \sigma(\mathsf{msg}(\rho@j))$. The idea is that the nodes are the part that has already happened. When $h = 0$, then $\mathsf{nodes}(\iota) = \emptyset$.

Each $\mathbb{P}$-instance $\iota = (\rho, h, \sigma)$ corresponds to a *regular strand* s of $\mathbb{P}$ by applying σ to ρ up to height h. That is $\mathsf{dmsg}(s@i) = \sigma(\mathsf{dmsg}(\rho@i))$ for each $i \leq h$ and $|s| = h$. An interesting subtlety arises when two roles have a common instance. That is (ρ, h, σ) and (ρ', h, σ') may satisfy $\sigma(\mathsf{dmsg}(\rho@i)) = \sigma'(\mathsf{dmsg}(\rho'@i))$ for each $1 \leq i \leq h$. This can represent a branching role that has a fixed trunk and alternate continuations. In the present paper we restrict our attention to non-branching protocols in the sense that no two roles share a common instance. This eases our connection to the applied π semantics later. We leave for future work the consideration of how to relate results for branching protocols.

$\mathbb{P}$ may make *role origination assumptions* $\mathsf{rlunique}$, stipulating that certain expressions involving the parameters originate at most once. These assumptions apply to *all* instances of the role. Formally, $\mathsf{rlunique}$ is a function of the roles of $\mathbb{P}$ and a height, returning a finite set of expressions: $\mathsf{rlunique} \colon \mathbb{P} \times \mathbb{N} \to \mathcal{P}(\mathsf{MSG})$. The set $\mathsf{rlunique}(\rho, i)$ gives ρ its unique origination assumptions for height i. We require that the image of $\mathsf{rlunique}$ consist only of terms in $\mathcal{N}_{\mathsf{s}}^{\nu}$ for the appropriate sort s, and that all other names in roles are chosen from the sets $\mathcal{N}_{\mathsf{s}}^{0}$.

We will assume that each protocol $\mathbb{P}$ contains the *listener role*, which consists of a single reception node $\overset{x}{\rightarrow}\bullet$. Each instance witnesses for the fact that the message instantiating x has been observed unprotected on the network. Thus, we use the listener role to express confidentiality failures. We also include a kind of dual to the listener role called a *blab role* that discloses a basic term to the adversary for it to use in deriving messages for reception. A blab strand witnesses for the fact that the adversary has managed to guess a value.

The two roles in Fig. 4 make up a strand-style definition of the SEP protocol (in which the listener and blab roles have been omitted).

Candidate strand runs. For the purposes of this paper, we slightly alter the notion of execution used for strand spaces. We argue in the appendix that this new notion preserves the semantics of $\mathcal{GL}(\mathbb{P})$. The notion of execution we consider, called a *candidate strand run*, or frequently, just a *candidate*, is a pair $\mathcal{I} = (I, \preceq)$ where $I = \langle \iota_1, \ldots, \iota_k \rangle$ is a finite sequence of $\mathbb{P}$-instances, and $\preceq$ is a partial order extending the strand succession orderings of $\mathsf{nodes}(\iota_i)$. We further require that $\mathcal{I}$ respect the $\mathsf{rlunique}$ assumptions of the roles. More formally, if $\iota_i = (\rho, h, \sigma)$ and $i \leq h$, then if $a \in \mathsf{rlunique}(\rho, i)$, then $\sigma(a)$ originates at most once in $\mathcal{I}$. The nodes of $\mathcal{I}$ are $\mathsf{nodes}(\mathcal{I}) = \{(i, n) \colon 1 \leq i \leq k \wedge n \in \mathsf{nodes}(\iota_i)\}$.

A reception node of $\mathcal{I}$ is *realized* if the adversary is in fact able to deliver $\mathsf{msg}(n)$ in time for each reception node n. This means that $\mathsf{msg}(n)$ should be derivable from previously transmitted messages. More formally, if $\mathcal{I} = (I, \preceq)$ is a candidate and $n \in \mathsf{nodes}^-(I)$, then n is *realized in* $\mathcal{I}$ iff

$$\{\mathsf{msg}(m) \in \mathsf{nodes}^+(I) \colon m \prec n\} \vdash \mathsf{msg}(n).$$

A candidate $\mathcal{I} = (I, \preceq)$ is a *strand run*, or just a *run*, iff, for every $n \in \mathsf{nodes}^-(I)$, n is realized in $\mathcal{I}$. We write $\mathsf{Runs}(\mathbb{P})$ for the set of strand runs of $\mathbb{P}$.

Operational semantics. The operational semantics of strand runs is obtained by defining an immediate successor relation on candidates and restricting it to runs. We first rely, however, on a localized notion of successor for instances.

If $\iota = (\rho, h, \sigma)$ and $\iota' = (\rho', h', \sigma')$ are instances, then ι' is *an immediate successor* of ι iff (i) $\rho = \rho'$; (ii) $h + 1 = h'$; and (iii) σ' restricted to the domain of σ agrees with σ. If ι' is an immediate successor of ι, then it extends σ to choose values for any new parameters that occur in $\mathsf{msg}(\rho @ h + 1)$, but not in $\mathsf{nodes}(\iota)$. This local successor relation lifts to a global successor relation on candidates.

One candidate $\mathcal{I}' = (I', \preceq')$ is an *immediate successor* of another candidate $\mathcal{I} = (I, \preceq)$ when there is one new node n in $\mathcal{I}'$, and the only change to the order is that some old nodes may precede n. More formally, $\mathcal{I}' = (I', \preceq')$ is an *immediate successor* of $\mathcal{I} = (I, \preceq)$ iff, letting $I = \langle \iota_1, \ldots, \iota_k \rangle$,

1. $\mathsf{nodes}(I') = \mathsf{nodes}(I) \cup \{n\}$, for a single $n \notin \mathsf{nodes}(I)$, i.e. either
   (a) $\mathsf{dom}(I') = \mathsf{dom}(I)$ and there is a $j \in \mathsf{dom}(I)$ s.t. $I'(j)$ is an immediate successor of ι_j, and for all $k \in \mathsf{dom}(I)$, if $k \neq j$ then $I'(k) = \iota_k$; or else
   (b) $I' = \langle \iota_1, \ldots, \iota_k, \iota'_{k+1} \rangle$, and ι'_{k+1} has height $h = 1$; and
2. There is a set of nodes $M \subseteq \mathsf{nodes}(I)$ such that $\preceq' = \preceq \cup \{(m, n) \colon m \in M\}$.

The empty candidate $\mathsf{NullRun} = (\langle\rangle, \emptyset)$ is a strand run, since it has no unrealized nodes. We regard it as the initial state in a transition relation, which is simply the "immediate successor" relation restricted to realized strand runs. We will write $S_{\mathbb{P}}$ for the immediate successor relation restricted to strand runs of $\mathbb{P}$, i.e. $S_{\mathbb{P}}(\mathcal{I}, \mathcal{I}')$ iff $\mathcal{I}, \mathcal{I}'$ are runs of $\mathbb{P}$ and $\mathcal{I}'$ is an immediate successor of $\mathcal{I}$.

Definition 1. *Let $\mathbb{P}$ be a protocol. The* operational semantics *of $\mathbb{P}$ is the state machine $M_{\mathbb{P}} = (\mathsf{Runs}(\mathbb{P}), \mathsf{NullRun}, S_{\mathbb{P}})$ where the set of states is $\mathsf{Runs}(\mathbb{P})$, the initial state is $\mathsf{NullRun}$, and the transition relation is $S_{\mathbb{P}}$.*

*A sequence of runs $\langle R_1, \ldots, R_i \rangle$ is an $M_{\mathbb{P}}$-*history *iff $R_1 = \mathsf{NullRun}$ and, for every j such that $1 \leq j < i$, $S_{\mathbb{P}}(R_j, R_{j+1})$.*

A run R is $\mathbb{P}$-accessible *iff for some $M_{\mathbb{P}}$-history $\langle R_1, \ldots, R_i \rangle$, $R = R_i$.* ///

By induction on the well-founded partial orders $\preceq_R$, we have:

Lemma 1. *Every $\mathbb{P}$ run is* $\mathbb{P}$-accessible. ///

Syntax and semantics of $\mathcal{GL}(\mathbb{P})$. $\mathcal{GL}(\mathbb{P})$'s protocol-dependent vocabulary contains one role position predicate $P_i^\rho(\cdot)$ for each role node $\rho@i$ of $\mathbb{P}$, and a collection of role parameter predicates $P_p^\rho(\cdot,\cdot)$, one for each parameter p in role ρ.

Candidates furnish models for the language $\mathcal{GL}(\mathbb{P})$ for security goals [21,28]. Candidates that are actually runs are the most important: They determine whether a protocol $\mathbb{P}$ *achieves* a formula $\Gamma \in \mathcal{GL}(\mathbb{P})$. In particular, $\mathbb{P}$ *achieves* Γ iff, for every *realized* run R and assignment η of objects in R to free variables in Γ, R satisfies Γ under η, typically written $R, \eta \models \Gamma$. The details of the semantics, using a slightly different notion of execution than the one used here, are in [21]. We show in the appendix that the semantics for runs is equivalent.

4 The Labeled Applied π Calculus

In this section we describe the triangle on the right side of Figure 1. We introduce a version of the applied π calculus [1]; we define a trace-based execution semantics; and we show how to extract $\mathcal{GL}^\star(P)$ from a protocol P, giving it a semantics with respect to the traces. Our process calculus is adapted from that of Cortier et al. [12] from which the following description borrows.

Applied π calculus syntax. Protocols are modeled as processes built on an infinite set of channel names $\mathcal{C}h$, using the following grammar.

$$\begin{array}{llll} P,Q = 0 & \mid \mathsf{in}(c,x)\,.\,P & \mid \mathsf{out}(c,u)\,.\,P & \mid \mathsf{let}\ x:\mathsf{s} = v\ \mathsf{in}\ P\ \mathsf{else}\ Q \\ \quad \mid (P \mid Q) & \mid \mathsf{new}\ n:\mathsf{s}\,.\,P & \mid \mathsf{sum}\ n':\mathsf{s}\,.\,P & \mid \mathsf{!new}\ \ tid\ .\ \mathsf{out}(c,tid)\,.\,P \\ \quad \mid \ell\,.\,P & & & \end{array}$$

Here $c, c' \in \mathcal{C}h$, $x \in \mathcal{X}$, $n \in \mathcal{N}_\mathsf{s}^\nu$, and $n' \in \mathcal{N}_\mathsf{s}^0$. We assume $u \in \mathsf{MSG}$ is a constructor term; $v \in \mathcal{T}(\Sigma, \mathcal{N} \cup \mathcal{X})$ can be any term. In this grammar, replication and channel restriction $\mathsf{new}\ \ tid$ occur only together.

We include the operator $\mathsf{sum}\ n' : \mathsf{s}$, acting as an infinite summation $P(n_1)+P(n_2)+\ldots$ for $n_i \in \mathcal{N}_\mathsf{s}^0$ (sse e.g. [20]). It binds parameters that might not be freshly chosen, e.g. agent names. Other processes may make sum choices that collide with this, but these choices never collide with the fresh choice $\mathsf{new}\ n : \mathsf{s}$. Although verification could be difficult with infinite choices, under reasonable restrictions it suffices to consider a bounded number of agents [11,12]. Similar results may apply equally to other types of values.

The *labels* ℓ are also prefixes. They implement the *begin-end events* in ProVerif and many other approaches, e.g. [29,19]. They signal the occurrence of steps mentioned in security goals such as authentication properties.

The free variables, free names, and free channels of P are denoted $fv(P)$, $fn(P)$, and $fc(P)$ respectively. P is a *basic process* iff P contains no parallel or replication operators, and all else branches in P are 0.

Modeling protocols. The roles of protocols are formalized as replicated processes $\mathsf{!new}\ \ tid\ .\ \mathsf{out}(c,tid)\,.\,P$ where P is a basic process. It is no restriction to assume that every role uses the *same* channel tid since each replicated session will instantiate tid

$$
\begin{aligned}
A := {} & \mathsf{sum}\ a : \mathsf{agt}.\mathsf{sum}\ b : \mathsf{agt}.\mathsf{new}\ s : \mathsf{skey}.\mathtt{InitStart}.\mathsf{out}(tid, \{\!|[\![s]\!]_{\mathsf{sk}(a)}|\}^{\mathsf{a}}_{\mathsf{pk}(b)}).\\
& \quad \mathsf{in}(tid, z).\mathsf{let}\ d : \mathsf{data} = \mathsf{dec}^{\mathsf{s}}(z, s)\ \mathsf{in}\ \mathtt{InitDone}.0\\
B := {} & \mathsf{sum}\ a : \mathsf{agt}.\mathsf{sum}\ b : \mathsf{agt}.\mathsf{in}(tid, z).\mathsf{let}\ x : \top = \mathsf{dec}^{\mathsf{a}}(z, \mathsf{sk}(b))\ \mathsf{in}\\
& \quad \mathsf{let}\ s : \mathsf{skey} = \mathsf{ver}(x, \mathsf{pk}(a))\ \mathsf{in}\ \mathtt{RespStart}.\mathsf{new}\ d : \mathsf{data}.\\
& \quad \mathtt{RespDone}.\mathsf{out}(tid, \{|d|\}^{\mathsf{s}}_{s}).0\\
P := {} & !\mathsf{new}\ \ tid\ .\mathsf{out}(c, tid).A \mid !\mathsf{new}\ \ tid\ .\mathsf{out}(c, tid).B \mid\\
& !\mathsf{new}\ \ tid\ .\mathsf{out}(c, tid).\mathsf{sum}\ v : \mathsf{s}.\mathtt{Blab}_{\mathsf{s}}.\mathsf{out}(tid, v).0 \mid \ldots
\end{aligned}
$$

Fig. 5. Applied π representation of SEP.

with a distinct fresh channel. Any parameters $p : \mathsf{s}$ assumed to be freshly chosen during *every* local session of a role will be bound $\mathsf{new}\ p : \mathsf{s}$ in P. Other parameters of the local session will be bound $\mathsf{sum}\ p : \mathsf{s}$ in P.

Each protocol includes *blab* roles, namely replicated processes $!\mathsf{new}\ \ tid\ .\ \mathsf{sum}\ p : \mathsf{s}.\mathsf{out}(tid, p)$. We must also include versions that send $\mathsf{f}(p)$ for each $\mathsf{f} \in \Sigma_{\mathsf{priv}}$. The representation of SEP is shown in Fig. 5, where the remaining blab processes are elided.

Operational semantics. Our operational semantics includes traces, namely sequences of *events*, i.e. pairs $(p, \mathcal{E})$ consisting of the prefix p of the process being reduced, together with the environment $\mathcal{E}$ that results from the reduction. A prefix p is either $\mathsf{in}(c, x : \top)$, $\mathsf{out}(c, u)$, or ℓ for some label ℓ. The trace joins together the begin-end events that the ProVerif semantics use with the message events in other semantics such Cortier et al.'s [12]. The labels help to provide semantics for role position predicates and parameter predicates, much as ProVerif etc. express authentication properties. Transmission and reception events reconstruct the semantics of origination.

The operational semantics acts on *configurations* $\mathcal{C}$, which are triples:

- $\mathcal{S}$ is a *trace*, namely a sequence of pairs of a prefix and an environment. It records the successive prefixes that have undergone reduction, and the environment in force when each reduction had occurred.
- $\mathcal{PE}$ is a multiset of pairs $(P, \mathcal{E})$ of a process and an environment. Each process is a subexpression of the original process expression, and represents possible future behavior. The environment records the bindings in force for its names and variables. We use it to remember the association of these values with the names and variables occurring in the original expression.
  The multiset operator is essentially the parallel operator, which obeys the usual associative-commutative structural rules, with unit 0.
- ϕ is a *frame*. It associates variables $w \in \mathcal{W}$ to transmitted messages. It indicates which messages from the regular participants the adversary is acting on.

The operational semantics (see Fig. 6 and Fig. 9 in the appendix), is a transition relation $\longrightarrow$ on configurations. In the IN rule, we do not substitute the new binding into the

$$\begin{array}{lrcl}
\mathsf{IN}: & \mathcal{S};\ (\mathsf{in}(c, x:\top).P, \mathcal{E}) \uplus \mathcal{PE};\ \phi & \longrightarrow & \mathcal{S}.(\mathsf{in}(c, x:\top), \mathcal{E}');\\
 & & & (P, \mathcal{E}') \uplus \mathcal{PE};\ \phi\\
\mathsf{OUT}: & \mathcal{S};\ (\mathsf{out}(c, u).P, \mathcal{E}) \uplus \mathcal{PE};\ \phi & \longrightarrow & \mathcal{S}.(\mathsf{out}(c, u), \mathcal{E});\\
 & & & (P, \mathcal{E}) \uplus \mathcal{PE};\ \phi[w \mapsto \mathcal{E}(u)]\\
\mathsf{LB}: & \mathcal{S};\ (\ell.P, \mathcal{E}) \uplus \mathcal{PE};\ \phi & \longrightarrow & \mathcal{S}.(\ell, \mathcal{E});\ (P, \mathcal{E}) \uplus \mathcal{PE};\ \phi\\
\mathsf{SESS}: & \mathcal{S};\ (!\mathsf{new}\ c'.\mathsf{out}(c, c').P, \mathcal{E}) \uplus \mathcal{PE};\ \phi & \longrightarrow & \mathcal{S};\ (P, \mathcal{E}') \uplus \mathcal{PE};\ \phi
\end{array}$$

where, in IN: $\mathcal{E}' = \mathcal{E}[x \mapsto \phi(R)\!\downarrow]$ for some $R \in \mathcal{T}(\Sigma_{\mathsf{pub}}, \mathcal{W})$;
OUT: $w \in \mathcal{W}$ is fresh;
SESS: $\mathcal{E}' = \mathcal{E}[c' \mapsto ch]$ where $ch \in \mathcal{C}h$ is fresh

Fig. 6. Some reduction rules

process expression, but simply accumulate it in the environment. In the OUT rule, the environment is consulted, producing the same effect the substitution would have had.

Reducing a prefix p to produce an environment $\mathcal{E}$ simply appends $(p, \mathcal{E})$ to the end of the trace. This retains the particular value to which a variable or name is bound when each prefix is executed. The role parameter predicates get their semantics from the bindings in $\mathcal{E}$ when a label is reduced, and the role position predicates get theirs from which label ℓ it is.

Goal language syntax. The goal language $\mathcal{GL}^\star(P)$ for a process P contains the same *protocol-independent* vocabulary as shown in Tab. 1. Its *protocol-dependent* vocabulary consists of *event predicates*, which are like role-position predicates of $\mathcal{GL}(\mathbb{P})$, and *environment predicates*, which are akin to parameter predicates.

Event predicates are one-place predicates. For each ℓ occurring in P, $\ell(\cdot)$ will be a (one-place) event predicate; it holds true of index i in trace $\mathcal{S}$ if the event $\mathcal{S}(i) = e$ is of the form $(\ell, \mathcal{E})$.

The environment predicates are two-place predicates. For each name or variable u occurring in P, $u(\cdot, \cdot)$ will be a (two-place) environment predicate. It will be true of pairs i, v when e is an event $(\ell, \mathcal{E})$ at index i in the trace, v is a message in normal form, and $\mathcal{E}(u) = t$ maps the name or variable u to t. When u is not bound in $\mathcal{E}$, $u(\cdot, \cdot)$ is false for i and every t.

Goal language semantics. Suppose that $\langle\rangle; P; \emptyset \longrightarrow^* \mathcal{S}; Q; \phi$, so that $\mathcal{S}$ is a trace of P. The semantics of $\mathcal{GL}^\star(P)$ is presented in Figure 7. The clauses are particularly simple, because we arranged for $\mathcal{S}$ to hold just the information needed to express them. In particular, retaining the environments $\mathcal{E}$ in $\mathcal{S}$ makes the semantics of the environment predicates very easy.

The predicate $\mathtt{Coll}(\cdot, \cdot)$ says that two events belong to the same instance ("session") of a role. By tying process replication to channel restriction, we ensure that $\mathcal{E}(tid)$ identifies the session that an event belongs to.

The final three predicates $\mathtt{Unq}(\cdot)$, $\mathtt{UnqAt}(\cdot, \cdot)$, and $\mathtt{Non}(\cdot)$ rely on *origination*, which thus must be determined by $\mathcal{S}$. This is why we included inputs $(\mathsf{in}(tid, x), \mathcal{E})$ and outputs $(\mathsf{out}(tid, u), \mathcal{E})$ in traces.

$\mathcal{S}, \eta \models \ell(m)$	iff	$\mathcal{S}(\eta(m)) = (\ell, \mathcal{E})$
$\mathcal{S}, \eta \models u(m, v)$	iff	$\mathcal{S}(\eta(m)) = (\ell, \mathcal{E})$ and $\mathcal{E}(u) = \eta(v)$
$\mathcal{S}, \eta \models v = v'$	iff	$\eta(v) = \eta(v')$
$\mathcal{S}, \eta \models \texttt{Preceq}(m, n)$	iff	$\eta(m) \leq \eta(n)$
$\mathcal{S}, \eta \models \texttt{Coll}(m, n)$	iff	$\mathcal{S}(\eta(m)) = (\ell_m, \mathcal{E}_m), \mathcal{S}(\eta(n)) = (\ell_n, \mathcal{E}_n)$ and $\mathcal{E}_m(tid) = \mathcal{E}_n(tid)$
$\mathcal{S}, \eta \models \texttt{Unq}(v)$	iff	$\eta(v)$ uniquely originates in $\mathcal{S}$
$\mathcal{S}, \eta \models \texttt{UnqAt}(m, v)$	iff	$\eta(v)$ uniquely originates at $\mathcal{S}(\eta(m))$
$\mathcal{S}, \eta \models \texttt{Non}(v)$	iff	$\eta(v)$ does not originate in $\mathcal{S}$

Fig. 7. Formal semantics of $\mathcal{GL}^\star(P)$, when $\langle\rangle; P; \emptyset \longrightarrow^* \mathcal{S}; Q; \phi$

Message t *originates* at $\mathcal{S}(i) = (p, \mathcal{E})$ if $p = \mathsf{out}(tid, u)$, $t \sqsubseteq \mathcal{E}(u)$ and for all $j < i$, if $\mathcal{S}(j) = (p', \mathcal{E}')$ with $\mathcal{E}'(tid) = \mathcal{E}(tid)$ and $p' = \mathsf{out}(tid, u')$ or $p' = \mathsf{in}(tid, u')$ then $t \not\sqsubseteq \mathcal{E}'(u')$. A message t *uniquely originates at* $\mathcal{S}(i)$ if t originates at $\mathcal{S}(i)$ and for all $j \neq i$, t does not originate at $\mathcal{S}(j)$. Similarly, we say t originates uniquely in $\mathcal{S}$ if it originates at $\mathcal{S}(i)$ for some unique i.

5 Compiling Strand Protocols to the Applied π Calculus

The previous two sections described the left and right triangles of Fig. 1. Each triangle makes sense in isolation: given a security goal and a protocol description, we can choose to use either formalism to validate that the protocol achieves the goal. However, we want goals verified in one formalism to hold in the other also. We thus expect to receive the same answer when evaluating the same protocol in either formalism. This of course requires a useful notion of *sameness* for descriptions of protocols in the two formalisms. However, a syntactic criterion for this would be difficult.

Instead, in this section we will briefly summarize a compiler (written in Prolog) that translates strand protocols $\mathbb{P} = \langle \rho_1, \ldots, \rho_k \rangle$ to processes P in the applied π calculus. Details of the compiler can be found in Appendix A. We designed it to correlate the goal languages $\mathcal{GL}(\mathbb{P})$ and $\mathcal{GL}^\star(P)$ smoothly, when P is an output from input $\mathbb{P}$.

We implement labels ℓ by expressions $\mathsf{pos}(i, j)$, and also use each label ℓ as a one place role position predicate $\ell(x)$ in $\mathcal{GL}^\star(P)$. The compiler associates each label used in the output with a node by constructing an injective function $\Lambda\colon \mathsf{Labs}(P) \to \mathsf{nodes}(\mathbb{P})$ where $\Lambda(\mathsf{pos}(i, j)) = \rho_i @ j$. The action of the function $f\colon \mathcal{GL}(\mathbb{P}) \to \mathcal{GL}^\star(P)$ on role position predicates (see Fig. 1) is inverse to Λ. More precisely,

$$\Lambda(f(\tau_r(\rho_i @ j))) = \rho_i @ j, \tag{1}$$

for all roles ρ_i and nodes $\rho_i @ j$ on it. We have written τ_r here for the map from role nodes to role position predicates, which partly determines the function $\mathbb{P} \to \mathcal{GL}(\mathbb{P})$. Thus, Λ is essentially inverse to $f \circ \tau_r$.

$$
\begin{aligned}
&!\mathsf{new}\ tid\ .\ \mathsf{out}(c, tid).\\
&\qquad \mathsf{pos}(2,1).\ \mathsf{out}(tid, \{\!|[\![\, s \,]\!]_{\mathsf{sk}(a)}|\!\}^{\mathsf{a}}_{\mathsf{pk}(b)}).\\
&\qquad \mathsf{in}(tid, x_1).\ \mathsf{let}\, x_2 : \top = \mathsf{dec}^{\mathsf{s}}(x_1, s)\, \mathsf{in}\, \mathsf{let}\, d : \mathsf{D} = x_2\, \mathsf{in}\, \mathsf{pos}(2,2).0
\end{aligned}
$$

Fig. 8. Translation of SEP initiator

The compiler also translates each parameter u of ρ_i, which may be either a name or a variable, to the same name or variable u in its target output. We use u as a two-place parameter predicate in $\mathcal{GL}^\star(P)$, where in this case f must satisfy:

$$f(\tau_p(\rho_i, u)) = u, \tag{2}$$

where we write τ_p for the map from roles ρ_i and parameters u to parameter predicates in $\mathcal{GL}(\mathbb{P})$. The function f is the identity function on protocol-independent vocabulary, so equations 1–2 characterize the translation f. There will also be other names and variables used in the process output by the compiler, which is why the map $f\colon \mathcal{GL}(\mathbb{P}) \to \mathcal{GL}^\star(P)$ is an embedding in this direction.

For simplicity, our compiler makes an assumption: It is designed to compile protocols whose roles are disjoint, in the sense that there are no strands that are common instances of distinct roles. Roles with overlapping instances are used to represent branching protocols, in which choices are made by principals or determined by the messages they receive. We have not refined our compiler to emit corresponding if-then-else expressions in the target π calculus. Throughout the remainder of this paper, we will assume that each strand-based protocol $\mathbb{P}$ has disjoint roles.

Compiler sketch. If the compiler translates the tail $n_{j+1} \Rightarrow \ldots \Rightarrow n_k$ of role ρ_i to a process P, then it prepends some code to P to translate $n_j \Rightarrow n_{j+1} \Rightarrow \ldots \Rightarrow n_k$. In particular, if $\mathsf{dmsg}(n_j) = +t$, then it emits a label followed by an output:

$$\mathsf{pos}(i,j)\ .\ \mathsf{out}\,(c,t)\ .\ P.$$

If t has a parameter that appears as an ingredient in t for the first time, the compiler should wrap this parameter in a new-binding.

If $\mathsf{dmsg}(n_j) = -t$, then the situation is more complicated. It must emit an input $\mathsf{in}\,(c,x)$ with a fresh variable x followed by a sequence of let bindings that destructure the received message. We insert the label $\mathsf{pos}(i,j)$ after this destructuring sequence. This is because its presence in a trace should imply that the expected message structure was present in the message bound to x. Message components that must equal known values will be checked, and previously unknown message components will be bound to fresh variables. When one of these components is represented by a parameter d in $-t$, the compiler re-uses d. Thus, parameters in ρ_i will also appear in its translation.

Having translated the content of a role ρ to a process P_0, the compiler wraps this and emits $!\mathsf{new}\ tid\ .\ \mathsf{out}(c, tid)\ .\ P_0$. The compiler does not rebind tid inside P_0, although it is convenient to use it as the public channel for input and output. As an example, the initiator role of SEP (left side of Fig. 4) yields the process expression shown in Fig. 8.

The first line shows the wrapping; the second line, the label and output for the first node of the role; the third line, the input, destructuring, and label for the second node, and the null termination. The tricky part of the compiler is computing the sequence of destructurings and checks. For this we use a simple flow analysis to determine choices for the participant's initial knowledge, followed by a backtracking analysis to explore the feasible combinations of destructuring input components vs. building known terms and checking equality with input components. This backtracking made the compiler convenient to implement in Prolog.

In both the input and the output case, the compiler emits code to add two entries to the trace for each single node of the source role ρ. Thus, we will correlate a single transmission node of ρ to a label followed by an output in the target process P. We will correlate a single reception node of ρ to an input followed by a label (confirming that destructuring has succeeded) in the target process P.

We codify this in a transition relation on configurations $\mathcal{C}$. We say that a configuration $\mathcal{C}_2 = (\mathcal{S}_2; \mathcal{PE}_2; \phi_2)$ is an *immediate successor* of a configuration $\mathcal{C}_1 = (\mathcal{S}_1; \mathcal{PE}_1; \phi_1)$ iff $\mathcal{C}_1 \longrightarrow^+ \mathcal{C}_2$ and for some values of the remaining variables, either

$$\mathcal{S}_2 = \mathcal{S}_1.(\ell, \mathcal{E}).(\mathsf{out}\,(c, u), \mathcal{E}) \quad \text{or else} \quad \mathcal{S}_2 = \mathcal{S}_1.(\mathsf{in}\,(c, x), \mathcal{E}).(\ell, \mathcal{E}').$$

Semantic correctness criterion. Intuitively, a role ρ and a (replicated) process term P represent the "same" activity if they can produce corresponding sequences of observable events. This suggests a kind of local bisimulation between role instances ι and basic processes P. However, we will correlate nodes on the strand side with pairs on the process side, whether a label-out pair or a in-label pair. We use the map Λ from labels back to nodes to define the correspondence.

Now, because basic processes retain only their future events, whereas instances contain both their past and their potential future, we actually correlate ι with a basic process and its environment, together with the trace $\mathcal{S}$ which retains information about the past.

We begin by defining an auxiliary predicate B_0^Λ, parameterized by the function Λ above, which captures the notion that the instance and the process have the same past. This predicate uses only the labeled entries in the trace, and ignores the inputs and outputs that it also contains.

Definition 2. *1. If $\mathcal{S}$ is the sequence $\mathcal{S} = \langle (p_1, \mathcal{E}_1), \ldots, (p_k, \mathcal{E}_k) \rangle$, then let $\mathcal{S}\uparrow_{tid} t$ be the subsequence of $\mathcal{S}$ which contains $(p_i, \mathcal{E}_i)$ iff p_i is a label ℓ_i and $\mathcal{E}_i(tid) = t$.*

2. Let ι be an instance, and let $\mathcal{S}$ be a trace and $\mathcal{E}$ an environment. $B_0^\Lambda(\iota; \mathcal{S}, \mathcal{E})$ holds iff, letting $\iota = (\rho, h, \sigma)$ and $\mathcal{T} = \mathcal{S}\uparrow_{tid}\mathcal{E}(tid)$,

(a) $\mathcal{E}$ restricts to σ, i.e. $\mathsf{dom}(\sigma) \subseteq \mathsf{dom}(\mathcal{E})$, and $\mathcal{E}(x) = \sigma(x)$ for all $x \in \mathsf{dom}(\sigma)$; and

(b) for all j such that $1 \leq j \leq h$, letting $\mathcal{T}(j) = (\ell_j, \mathcal{E}_j)$,

$$\mathsf{dmsg}(\iota, j) = \mathcal{E}_j(\mathsf{dmsg}(\Lambda(\ell_j))). \qquad ///$$

Condition (a) of Item 3 ensures that the parameters common to both formalizations have been bound in the same way by the two environments. Condition (b) ensures that if we apply Λ to the label of the j^{th} event and then apply the environment in effect at that event, we get the same directed message as the j^{th} node of the instance ι. Thus

the successive messages sent and received in $\mathcal{S}$ that are attributable to tid match the messages that ι has sent and received so far.

Thus, given Λ, the input and output events in the trace are effectively redundant. We include them so that the $\mathcal{GL}^\star$ semantics of Section 4 may be defined using only the intrinsic content of P and its reduction sequences. We would not want the semantics to be well-defined only for processes in the range of the compiler.

Lemma 2. *Let $B_0^\Lambda(\iota; \mathcal{S}, \mathcal{E})$, $\mathcal{T} = \mathcal{S}\restriction_{tid}\mathcal{E}(tid)$, and θ be an order-preserving bijection between $\mathsf{nodes}(\iota)$ and events of $\mathcal{T}$. Then for any atomic formula ϕ with a role position predicate, or parameter predicate, $\iota, \eta \models \phi$ iff $\mathcal{S}, \theta \circ \eta \models \phi$.* ///

We next need to describe what it means for an instance and a process to have the same possible futures. We thus define B_1^Λ to be the largest bisimulation that respects B_0^Λ. We use the immediate successor relation on configurations by injecting $P, \mathcal{E}$ to the singleton multiset $\mathcal{PE} = \{P, \mathcal{E}\}$.

Definition 3. *Let B_1^Λ be the most inclusive relation such that $B_1^\Lambda(\iota; \mathcal{S}, P, \mathcal{E})$ implies $B_0^\Lambda(\iota; \mathcal{S}, \mathcal{E})$, and moreover:*

1. *for all ι' such that ι' is an immediate successor of ι with new node n, there exist $\mathcal{S}', P', \mathcal{E}', \phi, \phi'$ such that $\mathcal{S}'; \{P', \mathcal{E}'\}; \phi'$ is an immediate successor of $\mathcal{S}; \{P, \mathcal{E}\}; \phi$, and $B_1^\Lambda(\iota'; \mathcal{S}', P', \mathcal{E}')$.*
2. *for all $\mathcal{S}', P', \mathcal{E}', \phi, \phi'$, if $\mathcal{S}'; \{P', \mathcal{E}'\}; \phi'$ is an immediate successor of $\mathcal{S}; \{P, \mathcal{E}\}; \phi$, then there is an immediate successor ι' of ι and $B_1^\Lambda(\iota'; \mathcal{S}', P', \mathcal{E}')$.* ///

We can also lift the B_1^Λ relation from an individual instance and basic process to a relation between a protocol $\mathbb{P}$ and a fully replicated process expression. In particular, we will assume that the roles in $\mathbb{P}$ are ordered, so that we can correlate them with parts of a process expression.

Definition 4. *Let σ_0 be the empty environment; let $[tid \mapsto v]$ be the environment with domain $\{tid\}$ and range v; and let $\mathcal{S}_0 = \langle\rangle$ be the empty trace.*

Suppose that $\mathbb{P} = \langle \rho_1, \ldots, \rho_k \rangle$, and let P be of the form:

$$|_{j \in \{1,\ldots,k\}} !\mathsf{new}\ tid\ .\ \mathsf{out}(c, tid)\ .\ P_j.$$

Then P represents *$\mathbb{P}$ via Λ iff, for each j such that $1 \leq j \leq k$, $B_1^\Lambda(\iota_j,\ \mathcal{S}_0, P_j, \mathcal{E}_j)$, where $\iota_j = (\rho_j, 0, \sigma_0)$, and $\mathcal{E}_j$ is of the form $[tid \mapsto v]$ for some v.* ///

The above definition serves as a semantic correctness criterion for a compiler that takes a strand space protocol $\mathbb{P}$ and produces an applied π process P. The next section shows why this is the case by lifting the local bisimulations to a global bisimulation and demonstrating that goal satisfaction is preserved when Definition 4 is met.

6 Bisimulation and preserving goals

Correctness: The idea. Λ, as generated by the compiler, and $f : \mathcal{GL}(\mathbb{P}) \to \mathcal{GL}^\star(P)$ are closely related, as shown in Section 5 (Eqs. 1–2). Hence, the behavioral match

between compiler input $\mathbb{P}$ and output P carries over to ensure that the goal formulas of $\mathcal{GL}(\mathbb{P})$ are preserved in $\mathcal{GL}^\star(P)$. We will not in fact prove that the compiler is correct—in the semantic sense of Def. 4—that its output P represents its input $\mathbb{P}$ via Λ, although we believe it. What we do prove is that *if* P represents $\mathbb{P}$, and the runs of $\mathbb{P}$ all achieve a security goal Γ, then the traces generated by P achieve $f(\Gamma)$.

The bisimulation. To do so, we demonstrate a weak bisimulation between the strand space operational semantics and the applied π reduction semantics. The bisimulation is between run-protocol pairs $(R, \mathbb{P})$ on the one hand and trace-configuration pairs $(\mathcal{S}, \mathcal{PE})$ on the other.

The initial configuration of $|_{1\leq j\leq k}\ P_j$ is $\langle\rangle, \{(P_1, \mathcal{E}_0), \ldots, (P_k, \mathcal{E}_0)\}, \emptyset$, and it evolves only to configurations $\mathcal{S}, \mathcal{PE}, \phi$ where $\mathcal{PE}$ splits into two parts:

$$\{(P_1, \mathcal{E}_0), \ldots, (P_k, \mathcal{E}_0)\} \uplus \{(BP_1, \mathcal{E}_1), \ldots, (BP_j, \mathcal{E}_j)\};$$

The latter is a multiset of pairs where each BP_i is a basic process. That is, the initially given replicated processes always remain unchanged, and all the additional processes can correspond to individual strand instances ι. We now formalize this correspondence via a bijection θ between labeled events and the nodes of these instances.

Definition 5. *$B^\Lambda_\theta(R, \mathbb{P}; \mathcal{S}, \mathcal{PE})$ iff θ is an bijection between $\mathsf{nodes}(R)$ and label events $(\ell, \mathcal{E})$ of $\mathcal{S}$ that preserves the orderings of R such that the following both hold:*

1. *θ induces a bijection between $\mathbb{P}$-instances ι of R and basic processes $P, \mathcal{E}$ of $\mathcal{PE}$ such that $B^\Lambda_1(\iota; \mathcal{S}, P, \mathcal{E})$.*
2. *There is a bijection ζ between roles ρ of $\mathbb{P}$ and replicated members of $\mathcal{PE}$ such that, for some fresh channel v, letting $\iota = (\rho, 0, \sigma_0)$, $B^\Lambda_1(\iota;\ \mathcal{S}_0, \zeta(\rho), [tid \mapsto v])$.*

We write $B^\Lambda(R, \mathbb{P}; \mathcal{S}, \mathcal{PE})$ iff, for some θ, $B^\Lambda_\theta(R, \mathbb{P}; \mathcal{S}, \mathcal{PE})$. ///

Lemma 3. *Suppose $B^\Lambda(R, \mathbb{P}; \mathcal{S}, \mathcal{PE})$, and let $R_{out} = \{\mathsf{msg}(m) : m \in \mathsf{nodes}^+(R)\}$ and $\mathcal{S}_{out} = \{\mathcal{E}(u) : \mathcal{S}(i) = (\mathsf{out}(tid, u), \mathcal{E})$ for some $i.\}$. Then $R_{out} \vdash t$ iff $\mathcal{S}_{out} \vdash t$.*

Proof (Sketch.). Being in the B^Λ relation ensures that $R_{out} = \mathcal{S}_{out}$. □

Lemma 4. *$B^\Lambda(R, \mathbb{P}; \mathcal{S}, \mathcal{PE})$ is a bisimulation.*

Proof. We begin by showing that $\mathcal{S}, \mathcal{PE}$ simulates $R, \mathbb{P}$. By assumption, there is some θ that matches the instances ι of R to the unreplicated process environment pairs $P, \mathcal{E}$ of $\mathcal{PE}$ so that $B^\Lambda_1(\iota; \mathcal{S}, P, \mathcal{E})$. Let $\phi = \phi(\mathcal{S})$ be the environment associated with trace $\mathcal{S}$. The run R can advance in one of two ways, (a) some current instance is extended to a successor instance, or (b) some new instance is created from a role of $\mathbb{P}$. In the first case, since $B^\Lambda_1(\iota; \mathcal{S}, P, \mathcal{E})$, the configuration $\mathcal{S}; \mathcal{PE}; \phi$ can evolve similarly if either the new node in the extended instance is a transmission, or, in case it is a reception $-m$, if $\mathcal{S}_{out} \vdash m$. But since the run R could only have advanced with a reception if $R_{out} \vdash m$, Lemma 3 ensures that $\mathcal{S}_{out} \vdash m$, as required.

In the second case, we note that we can first silently create a new unreplicated basic process BP_{j+1} with environment $\mathcal{E}_{j+1} = [tid \mapsto v]$ for some fresh channel v by performing a $SESS$ reduction. Condition 2 of Def. 5 ensures that $B^\Lambda_1(\iota'; \mathcal{S}, BP_{j+1}, \mathcal{E}_{j+1})$ where ι' is the 0-height prefix of the new instance ι. We can thus proceed to argue as in the first case above. The proof of the reverse simulation is similar. □

Theorem 1. *Suppose that* $|_{1\leq j\leq k}\ P_j$ *represents* $\mathbb{P}$ *via* Λ*, and let* θ *be the bijection with empty domain. Then* $B^{\Lambda}_{\theta}(\emptyset, \mathbb{P}; \langle\rangle, \{(P_1, \mathcal{E}_0), \ldots, (P_k, \mathcal{E}_0)\})$.

Proof. Condition 1 of Def. 5 is vacuously satisfied. Since $|_{1\leq j\leq k}\ P_j$ represents $\mathbb{P}$ via Λ, Def. 4 applies which ensures Condition 2 holds. □

Lemma 5. *Suppose that* $B^{\Lambda}_{\theta}(R, \mathbb{P}; \mathcal{S}, \mathcal{PE})$*, where* $\theta\colon \mathsf{nodes}(R) \to \mathsf{Labs}(\mathcal{S})$*. Let* $\hat{\theta}$ *extend* θ *to* MSG *also by acting as the identity. Let* ϕ *be an atomic formula of* $\mathcal{GL}(\mathbb{P})$.

1. *If* $R, \eta \models \phi$*, then* $\mathcal{S}, \hat{\theta} \circ \eta \models f(\phi)$.
2. *If* ϕ *does not contain* `Preceq`*, then* $\mathcal{S}, \hat{\theta} \circ \eta \models f(\phi)$ *implies* $R, \eta \models \phi$.

Proof (Sketch). Lemma 2 takes care of the cases for role position predicates and parameter predicates. The bisimulation relation ensures that origination, message equality, and local session orderings are preserved. Since θ only preserves orders from R to $\mathcal{S}$, we must exclude `Preceq` for Condition 2. □

Theorem 2. *If* P *represents* $\mathbb{P}$ *via* Λ *and* $\mathbb{P}$ *achieves* $\forall \bar{x}\ .\ \Phi \Longrightarrow \Psi$*, where only* $\vee, \wedge, \exists$ *appear in* Φ *and* Ψ*, then* P *achieves* $f(\forall \bar{x}\ .\ \Phi \Longrightarrow \Psi)$. ///

The converse is false, since the execution model of P is linear, while the runs of $\mathbb{P}$ are partially ordered. In particular, the formula $\forall n, m\ .\ n \preceq m \vee m \preceq n$ holds of P, but need not hold of $\mathbb{P}$. However, we conjecture that $\mathbb{P}$ achieves a security goal $\forall \bar{x}\ .\ \Phi \Longrightarrow \Psi$, where Φ, Ψ use only $\vee, \wedge, \exists$, if P satisfies $f(\forall \bar{x}\ .\ \Phi \Longrightarrow \Psi)$ and either

1. $\preceq$ does not appear in Ψ; or else
2. $\vee$ does not appear in Ψ.

In the first case, we transport satisfying instances from traces of P back to corresponding runs of $\mathbb{P}$, as in Clause 2. The second appears to be true because if Ψ is $\vee$-free, its `Preceq`-containing atomic formulas are satisfied in all traces of P. Thus, they hold in all interleavings, whence they must be true in the corresponding partially ordered $\mathbb{P}$ run.

7 Conclusion

In this paper, we studied a particular case of the cross-tool security goal problem for protocol standardization. We showed how to correlate statements in a goal language for a strand space tool with statements in a related language for applied π. We proved that if a strand-based protocol achieves a security goal, then related protocols in applied π achieve the corresponding goal. We conjecture that the converse is true for a large set of security goals also. We provided a compiler to produce a related applied π protocol.

These technical contributions support the protocol verification framework codified in ISO/IEC 29128. A goal language that does not depend on the underlying verification tool allows for greater transparency for published standards: it allows practitioners to independently verify the same results using the tool of their choice.

We view this paper as a start on a program to which many hands may contribute, adapting the semantics of different tools to this or a comparable security goal language. Although the languages $\mathcal{GL}(\mathbb{P})$ express only safety properties, rather than indistinguishability properties also, it seems likely that a similar program could equally apply to indistinguishability properties.

References

1. Martín Abadi and Cédric Fournet. Mobile values, new names, and secure communication. In *28th ACM Symposium on Principles of Programming Languages (POPL '01)*, pages 104–115, January 2001.
2. A. Armando, D. A. Basin, Y. Boichut, Y. Chevalier, L. Compagna, J. Cuéllar, P. Hankes Drielsma, P.-C. Héam, O. Kouchnarenko, J. Mantovani, S. Mödersheim, D. von Oheimb, M. Rusinowitch, J. Santiago, M. Turuani, L. Viganò, and L. Vigneron. The AVISPA tool for the automated validation of internet security protocols and applications. In Kousha Etessami and Sriram K. Rajamani, editors, *CAV*, volume 3576 of *Lecture Notes in Computer Science*, pages 281–285. Springer, 2005.
3. Alessandro Armando, Wihem Arsac, Tigran Avanesov, Michele Barletta, Alberto Calvi, Alessandro Cappai, Roberto Carbone, Yannick Chevalier, Luca Compagna, Jorge Cuéllar, Gabriel Erzse, Simone Frau, Marius Minea, Sebastian Mödersheim, David von Oheimb, Giancarlo Pellegrino, Serena Elisa Ponta, Marco Rocchetto, Michaël Rusinowitch, Mohammad Torabi Dashti, Mathieu Turuani, and Luca Viganò. The AVANTSSAR platform for the automated validation of trust and security of service-oriented architectures. In *Tools and Algorithms for the Construction and Analysis of Systems - 18th International Conference, TACAS 2012, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2012, Tallinn, Estonia, March 24 - April 1, 2012. Proceedings*, pages 267–282, 2012.
4. David Baelde, Stéphanie Delaune, and Lucca Hirschi. Partial order reduction for security protocols. In *26th International Conference on Concurrency Theory, CONCUR 2015, Madrid, Spain, September 1.4, 2015*, pages 497–510, 2015.
5. David A. Basin, Cas J. F. Cremers, Kunihiko Miyazaki, Sasa Radomirovic, and Dai Watanabe. Improving the security of cryptographic protocol standards. *IEEE Security & Privacy*, 13(3):24–31, 2015.
6. Stefano Bistarelli, Iliano Cervesato, Gabriele Lenzini, and Fabio Martinelli. Relating multiset rewriting and process algebras for security protocol analysis. *Journal of Computer Security*, 13(1):3–47, 2005.
7. Bruno Blanchet. An efficient protocol verifier based on Prolog rules. In *14th Computer Security Foundations Workshop*, pages 82–96. IEEE CS Press, June 2001.
8. Bruno Blanchet. *Vérification automatique de protocoles cryptographiques: modèle formel et modèle calculatoire. Automatic verification of security protocols: formal model and computational model*. Mémoire d'habilitation à diriger des recherches, Université Paris-Dauphine, November 2008.
9. Michael Burrows, Martín Abadi, and Roger Needham. A logic of authentication. *Proceedings of the Royal Society*, Series A, 426(1871):233–271, December 1989.
10. Iliano Cervesato, Nancy A. Durgin, and Patrick Lincoln. A comparison between strand spaces and multiset rewriting for security protocol analysis. *Journal of Computer Security*, 13(2):265–316, 2005.
11. Hubert Comon and Véronique Cortier. Security properties: two agents are sufficient. *Science of Computer Programming*, 50(1-3):51–71, March 2004.
12. Véronique Cortier, Antoine Dallon, and Stéphanie Delaune. Bounding the number of agents, for equivalence too. In *Principles of Security and Trust - 5th International Conference, POST 2016, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2016, Eindhoven, The Netherlands, April 2-8, 2016, Proceedings*, pages 211–232, 2016.
13. Veronique Cortier and Steve Kremer, editors. *Formal Models and Techniques for Analyzing Security Protocols*. Cryptology and Information Security Series. IOS Press, 2011.

14. Federico Crazzolara and Glynn Winskel. Events in security protocols. In *CCS 2001, Proceedings of the 8th ACM Conference on Computer and Communications Security, Philadelphia, Pennsylvania, USA, November 6-8, 2001.*, pages 96–105, 2001.
15. Cas Cremers and Sjouke Mauw. *Operational semantics and verification of security protocols*. Springer, 2012.
16. Cas J. F. Cremers. Key exchange in ipsec revisited: Formal analysis of ikev1 and ikev2. In *Computer Security - ESORICS 2011 - 16th European Symposium on Research in Computer Security, Leuven, Belgium, September 12-14, 2011. Proceedings*, pages 315–334, 2011.
17. Anupam Datta, Ante Derek, John C. Mitchell, and Arnab Roy. Protocol composition logic (PCL). *Electr. Notes Theor. Comput. Sci.*, 172:311–358, 2007.
18. Joseph A. Goguen and José Meseguer. Order-sorted algebra I: equational deduction for multiple inheritance, overloading, exceptions and partial operations. *Theoretical Computer Science*, 105(2):217–273, 1992.
19. Andrew D. Gordon and Alan Jeffrey. Types and effects for asymmetric cryptographic protocols. *Journal of Computer Security*, 12(3/4):435âĂŞ–484, 2004.
20. J.F. Groote and M.R. Mousavi. *Modeling and analysis of communicating systems*. MIT Press, Cambridge, MA, 2014.
21. Joshua D. Guttman. Establishing and preserving protocol security goals. *Journal of Computer Security*, 22(2):201–267, 2014.
22. ISO/IEC. *Information Technology - Security techniques — Verification of Cryptographic Protocols*, 2011.
23. Shin'ichiro Matsuo, Kunihiko Miyazaki, Akira Otsuka, and David A. Basin. How to evaluate the security of real-life cryptographic protocols? - the cases of ISO/IEC 29128 and CRYPTREC. In *Financial Cryptography and Data Security, FC 2010 Workshops, RLCPS, WECSR, and WLC 2010, Tenerife, Canary Islands, Spain, January 25-28, 2010, Revised Selected Papers*, pages 182–194, 2010.
24. C. Meadows. The NRL protocol analyzer: An overview. *The Journal of Logic Programming*, 26(2):113–131, 1996.
25. Catherine Meadows. Analysis of the Internet Key Exchange protocol using the NRL protocol analyzer. In *Proceedings, 1999 IEEE Symposium on Security and Privacy*. IEEE CS Press, May 1999.
26. Dale Miller. Encryption as an abstract data type. *Electr. Notes Theor. Comput. Sci.*, 84:18–29, 2003.
27. John D. Ramsdell and Joshua D. Guttman. CPSA: A cryptographic protocol shapes analyzer, 2009. `http://hackage.haskell.org/package/cpsa`.
28. Paul D. Rowe, Joshua D. Guttman, and Moses D. Liskov. Measuring protocol strength with security goals. *International Journal of Information Security*, Accepted: Forthcoming.
29. Thomas Y. C. Woo and Simon S. Lam. A lesson on authentication protocol design. *Operating Systems Review*, pages 24–37, 1994.

A Appendix

A.1 Equivalence of two strand space semantics

The new operational semantics presented in Section 3 is only inessentially different from the usual strand space semantics in terms of realized skeletons. In order to demonstrate this, we now present the usual notion of execution for strand spaces, and demonstrate the equivalence of the two semantics.

Skeletons. A *skeleton* $\mathbb{A}$ for $\mathbb{P}$ is a structure that provides partial information about a set of executions of $\mathbb{P}$. It consists of (i) a finite sequence of regular strands (or equivalently, instances) of $\mathbb{P}$; (ii) a partial ordering $\preceq_{\mathbb{A}}$ on the nodes of $\mathbb{A}$ extending the strand succession orderings; and (iii) two sets of terms $\mathsf{unique}_{\mathbb{A}}$ and $\mathsf{non}_{\mathbb{A}}$ representing terms that may originate on at most one node and terms that must not originate respectively. We assume that $\mathbb{A}$ inherits the origination assumptions from the roles of the protocol in that the set $\mathsf{unique}_{\mathbb{A}} \supseteq \sigma(\mathsf{rlunique}(\rho, i))$ for every instance $\iota = (\rho, h, \sigma)$ of $\mathbb{A}$ and every $i \leq h$.

A skeleton $\mathbb{A}$ is *realized* iff, for every reception node $n \in \mathsf{nodes}^{-}(\mathbb{A})$, $\mathsf{msg}(n)$ is derivable from previously transmitted messages and guessable values. More formally, $T \cup (B \setminus X) \vdash \mathsf{msg}(n)$ where $T = \{\mathsf{msg}(m) \mid m \in \mathsf{nodes}^{+}(\mathbb{A}) \wedge m \prec_{\mathbb{A}} n\}$, B is the set of basic values, and $X = \mathsf{unique}_{\mathbb{A}} \cup \mathsf{non}_{\mathbb{A}}$ is the set of all non-guessable basic values.

We can correlate the realized skeletons of any protocol $\mathbb{P}$ (that excluded blab roles) with the $\mathbb{P}'$-accessible runs, where $\mathbb{P}' = \mathbb{P} \cup \{blabs\}$. The idea is to add blab nodes for all the basic values the adversary is allowed to guess. More formally, let

$$B_{\mathbb{A}} = \{b \mid \exists n \in \mathsf{nodes}(\mathbb{A}) \,.\, b \text{ is a subterm of } \mathsf{msg}(n) \wedge b \text{ is a basic value}\}$$

and let B' be a set of blab nodes, one for each element of $B_{\mathbb{A}} \setminus (\mathsf{unique}_{\mathbb{A}} \cup \mathsf{non}_{\mathbb{A}})$. We say that a realized skeleton $\mathbb{A}$ and a run R are *related* iff $\mathsf{nodes}(R) = \mathsf{nodes}(\mathbb{A}) \cup B'$, and $\preceq_{\mathbb{A}} = \preceq_R \cap (\mathsf{nodes}(\mathbb{A}) \times \mathsf{nodes}(\mathbb{A}))$.

Lemma 6. *Let $\mathbb{P}' = \mathbb{P} \cup \{blabs\}$. Every realized $\mathbb{P}$-skeleton $\mathbb{A}$ has a related $\mathbb{P}'$ run R. Every $\mathbb{P}'$ run R has a related realized $\mathbb{P}$-skeleton $\mathbb{A}$.* ///

Lemma 7. *Let $\mathbb{P}' = \mathbb{P} \cup \{blabs\}$, and let $\mathbb{A}$ be a realized $\mathbb{P}$-skeleton, and R a related $\mathbb{P}'$ run. Then for any atomic formula ϕ and any variable assignment η of variables to nodes and terms in $\mathbb{A}$, $\mathbb{A}, \eta \models \phi$ iff $R, \eta \models \phi$.* ///

Lemma 7 in fact lifts to goal formulas Γ as a natural corollary. The set of goals achieved by $\mathbb{P}'$ is essentially the same as that achieved by $\mathbb{P}$. In particular, any goal Γ true of a skeleton $\mathbb{A}$ is also true of some related run R. Similarly, as long as the Γ does not express anything explicitly about the blab nodes, if the formula is true of R it is also true of $\mathbb{A}$. It is therefore no danger to use the operational semantics of runs instead of the skeleton semantics when forming a connection to the applied π semantics.

A.2 Remaining Reduction Rules

The reduction rules omitted in Section 4 are gathered in Fig. 9. As usual, we assume that $!P$ is structurally equivalent to $P \mid !P$.

A.3 Compilation

In this section we describe our translation of a strand space role into a labeled applied π-calculus process term.

$$\frac{\mathcal{E}' = \mathcal{E}[n \mapsto n'] \text{ with } n' \text{ fresh from } \mathcal{N}_{\mathsf{s}}^{\nu}}{\mathcal{S}; (\mathsf{new}\ n : \mathsf{s}.P, \mathcal{E}) \uplus \mathcal{PE}; \phi \to \mathcal{S}; (P, \mathcal{E}') \uplus \mathcal{PE}; \phi}\ \text{NEW}$$

$$\frac{\mathcal{E}' = \mathcal{E}[n \mapsto n'] \text{ with } n' \in \mathcal{N}_{\mathsf{s}}^{0}}{\mathcal{S}; (\mathsf{sum}\ n : \mathsf{s}.P, \mathcal{E}) \uplus \mathcal{PE}; \phi \to \mathcal{S}; (P, \mathcal{E}') \uplus \mathcal{PE}; \phi}\ \text{SUM}$$

$$\frac{\mathcal{E}' = \mathcal{E}[x \mapsto v\!\downarrow] \text{ with } v\!\downarrow\!: \mathsf{s} \in \mathcal{M}_{\Sigma}}{\mathcal{S}; (\mathsf{let}\ x : \mathsf{s} = v\ \mathsf{in}\ P\ \mathsf{else}\ Q, \mathcal{E}) \uplus \mathcal{PE}; \phi \to \mathcal{S}; (P, \mathcal{E}') \uplus \mathcal{PE}; \phi}\ \text{LET}$$

$$\frac{v\!\downarrow\!\notin \mathcal{M}_{\Sigma} \text{ or } \neg v\!\downarrow\!: \mathsf{s}}{\mathcal{S}; (\mathsf{let}\ x : \mathsf{s} = v\ \mathsf{in}\ P\ \mathsf{else}\ Q, \mathcal{E}) \uplus \mathcal{PE}; \phi \to \mathcal{S}; (Q, \mathcal{E}) \uplus \mathcal{PE}; \phi}\ \text{LET-FAIL}$$

$$\frac{}{\mathcal{S}; (0, \mathcal{E}) \uplus \mathcal{PE}; \phi \to \mathcal{S}; \mathcal{PE}; \phi}\ \text{NULL}$$

$$\frac{}{\mathcal{S}; (P \mid Q, \mathcal{E}) \uplus \mathcal{PE}; \phi \to \mathcal{S}; (P, \mathcal{E}) \uplus (Q, \mathcal{E}) \uplus \mathcal{PE}; \phi}\ \text{PAR}$$

Fig. 9. Remaining reduction rules

At a high level, the translation takes a transmission event $+m$ to $\mathsf{out}(tid, m)$, and it takes a reception event $-m$ to $\mathsf{in}(tid, z).P$ where P is a sequence of let bindings that attempt to parse the received term according to the structure of the expected term. The complexity of the latter translation is due to the use of pattern matching for receptions in strand spaces that is absent in processes. If we are to preserve the semantics of the goal language under this translation to the process calculus, we must ensure that receptions based on pattern matching succeed on a given message m if and only if the corresponding sequence of let bindings succeeds on the same message. This requires some care.

One issue is that there may be several sequences that can be used to verify the structure of a message. Since the parsing process binds some values and requires others already to be bound, some sequences are sensible with respect to some initial input and others are not.

We start with a strand space trace (a sequence of events) constructed from message terms derived from the order-sort signature in Fig. 10. We compute the relation between a strand space trace and a process calculus term two steps.

1. Perform a flow analysis to find a set of input basic values (See Fig. 11).
2. Translate the trace into a process calculus expression relative to a given set of inputs (See Fig. 14).

The algorithm has been simplified by ignoring role unique origination assumptions, but their processing is sketched near the end of this appendix. Most of the algorithm described here has been implemented in Prolog. The Prolog implementation operates on a many-sorted algebra isomorphic to the order-sorted algebra as described in [18, Sec. 4]. We leave that translation implicit in this document.

Sorts:	$\top$, D, S, A, N		
Subsorts:	$\mathsf{D} < \top$, $\mathsf{S} < \top$, $\mathsf{A} < \top$, $\mathsf{N} < \top$		
Operations:	$(\cdot,\cdot)$	$: \top \times \top \to \top$	Pairing
	$\{\!\|\cdot\|\!\}_{(\cdot)}$	$: \top \times \mathsf{S} \to \top$	Symmetric encryption
	$\{\!\|\cdot\|\!\}_{(\cdot)}$	$: \top \times \mathsf{A} \to \top$	Asymmetric encryption
	$(\cdot)^{-1}$	$: \mathsf{A} \to \mathsf{A}$	Asymmetric key inverse
	pk	$: \mathsf{N} \to \mathsf{A}$	Public key for name
Equation:	$(x^{-1})^{-1} = x$ for $x : \mathsf{A}$		

Fig. 10. Simple Crypto Algebra Signature

The signature in Fig. 10 is a simplification of the one used by CPSA. The Simple Example Protocol initiator role using this signature is:

$$init(a, b\colon N, s\colon S, d\colon D) = [+\{\!|\{\!|s|\!\}_{pk(a)^{-1}}|\!\}_{pk(b)}, -\{\!|d|\!\}_s]. \tag{3}$$

A.4 Flow Analysis

$$\frac{\emptyset, \emptyset, C \rhd I, A}{C \rhd I}$$

$$I, A, [\,] \rhd I, A$$

$$\frac{I_1, A_1, M \rhd^+ I_2, A_2 \qquad I_2, A_2, C \rhd I_3, A_3}{I_1, A_1, +M :: C \rhd I_3, A_3}$$

$$\frac{I_1, A_1, M \rhd^- I_2, A_2 \qquad I_2, A_2, C \rhd I_3, A_3}{I_1, A_1, -M :: C \rhd I_3, A_3}$$

Fig. 11. Flow Analysis

The aim of the flow analysis $C \rhd I$ (see Fig. 12) is to find a set of basic values that allow a procedural interpretation of a trace, in particular, a procedural interpretation of the implied pattern matching that is part of a strand space reception event.

There are two ways to interpret the reception of a pair, either the left part is matched first or the right part. A decryption key might or might not become available based on this choice.

There are two ways to interpret the reception of an encryption. If its decryption key in known at the point of the match, the contents of the encryption can be extracted. Alternatively, if the encryption has been seen previously or can be constructed, then an equality check implements the match.

Fig. 13 explores the various possibilities. The flow analysis for the initiator trace is:

$$I = \{\{pk(b), pk(a)^{-1}, s\}, \{d, pk(b), pk(a)^{-1}, s\}\}, \tag{4}$$

$$\frac{M \in A}{I, A, M \rhd^+ I, A}$$

$$\frac{I_1, A_1, M \rhd^+ I_2, A_2 \qquad I_2, A_2, N \rhd^+ I_3, A_3}{I_1, A_1, \langle M, N \rangle \rhd^+ I_3, A_3}$$

$$\frac{I_1, A_1, M \rhd^+ I_2, A_2 \qquad I_2, A_2, N \rhd^+ I_3, A_3}{I_1, A_1, \{\!|M|\!\}_N \rhd^+ I_3, A_3}[N : \mathsf{S} \text{ or } N : \mathsf{A}]$$

$$\frac{M \text{ is a basic value and not in } A}{I, A, M \rhd^+ \{M\} \cup I, \{M\} \cup A}$$

Fig. 12. Send Flow Analysis

$$\frac{I_1, A_1, M \rhd^- I_2, A_2 \qquad I_2, A_2, N \rhd^- I_3, A_3}{I_1, A_1, \langle M, N \rangle \rhd^- I_3, A_3}$$

$$\frac{I_1, A_1, N \rhd^- I_2, A_2 \qquad I_2, A_2, M \rhd^- I_3, A_3}{I_1, A_1, \langle M, N \rangle \rhd^- I_3, A_3}$$

$$\frac{I_1, \{\{\!|M|\!\}_N\} \cup A_1, N \rhd^+ I_2, A_2 \qquad I_2, A_2, M \rhd^- I_3, A_3}{I_1, A_1, \{\!|M|\!\}_N \rhd^- I_3, A_3}[N : \mathsf{S} \text{ or } N : \mathsf{A}]$$

$$\frac{I_1, A_1, \{\!|M|\!\}_N \rhd^+ I_2, A_2}{I_1, A_1, \{\!|M|\!\}_N \rhd^- I_2, A_2}$$

$$\frac{M \text{ is a basic value}}{I, A, M \rhd^- I, \{M\} \cup A}$$

Fig. 13. Receive Flow Analysis

where $b, a : \mathsf{N}$, $s : \mathsf{S}$, and $d : \mathsf{D}$. Notice the second solution makes little sense. It assumes that the initiator's initial knowledge includes d, the data it is seeking from a responder. We rely on human intervention to choose sensible sets of input terms.

A.5 Code Generation

$$[\,], E_1, N, \ell \gg 0, E_2$$

$$\frac{T_1, E_1 \gg^+ T_2 \qquad C, E_1, N, \ell' \gg P, E_2}{+T_1 :: C, E_1, N_1, \ell \gg \mathsf{pos}(N, \ell).\,\mathsf{out}(c, T_2).P, E_2}\ [\ell' := \ell + 1]$$

$$\frac{x, T, \mathsf{pos}(N, \ell).P_1, E_1 \gg^- P_2, E_2 \qquad C, E_2, N, \ell' \gg P_1, E_3}{-T :: C, E_1, N, \ell \gg \mathsf{in}(c, x).P_2, E_3}\ [x : \top \text{ fresh}, \ell' := \ell + 1]$$

Fig. 14. Code Generation

Code generation has the form $C, E_1, N, \ell \gg P, E_2$, where C is a strand space trace, E_1 and E_2 are maps from strand space terms to process calculus terms, N, ℓ are a natural numbers, and P is a process calculus term in a language that has been extended with one new production:

$$P ::= \mathsf{pos}(N_1, N_2).P \mid \ldots$$

where N_1 and N_2 are a natural numbers. The form $\mathsf{pos}(N, \ell)$ asserts we are translating the ℓ^{th} send or receive in the trace of the N^{th} role of the protocol.

An analysis begins with an environment E_0 mapping each input term computed by the flow analysis to itself. To compute the process calculus term P for a given strand space trace C and role number N, find P such that $C, E_0, N, 1 \gg P, E_2$ (See Fig. 14).

$$\frac{(T, x) \in E}{T, E \gg^+ x}$$

$$\frac{T_1, E \gg^+ x_1 \qquad T_2, E \gg^+ x_2}{\langle T_1, T_2 \rangle, E \gg^+ \langle x_1, x_2 \rangle}$$

$$\frac{T_1, E \gg^+ x_1 \qquad T_2, E \gg^+ x_2}{\{\!|T_1|\!\}_{T_2}, E \gg^+ \{\!|x_1|\!\}_{x_2}}[T_2 : \mathsf{S} \text{ or } T_2 : \mathsf{A}]$$

Fig. 15. Send Code Generation

To handle role unique origination assumptions, the send code generator in Fig. 15 must prefix the code with a new form for each name that uniquely originates in the transmitted message.

$$\frac{(T, y) \in E}{x, T, P, E \gg^{-} \mathsf{let\,ok} = eq(x, y)\,\mathsf{in}\,P, E}$$

$$\frac{y, T_1, P_1, E_1 \gg^{-} P_2, E_2 \qquad z, T_2, P_2, E_2 \gg^{-} P_3, E_3}{x, \langle T_1, T_2\rangle, P_1, E_1 \gg \mathsf{let}\langle y, z\rangle = x\,\mathsf{in}\,P_3, E_3}\,[y, z : \top \text{ fresh}]$$

$$\frac{z, T_2, P_1, E_1 \gg^{-} P_2, E_2 \qquad y, T_1, P_2, E_2 \gg^{-} P_3, E_3}{x, \langle T_1, T_2\rangle, P_1, E_1 \gg \mathsf{let}\langle y, z\rangle = x\,\mathsf{in}\,P_3, E_3}\,[y, z : \top \text{ fresh}]$$

$$\frac{(T_2, y) \in E_1 \qquad z, T_1, P_1, \{(\{\!|T_1|\!\}_{T_2}, x)\} \cup E_1 \gg P_2, E_1}{x, \{\!|T_1|\!\}_{T_2}, P_1, E_1 \gg \mathsf{let}\, z = dec(x, y)\,\mathsf{in}\,P_2, E_2}\,[z : \top \text{ fresh}, T_2 : \mathsf{S}]$$

$$\frac{E \vdash \{\!|T_1|\!\}_{T_2}}{x, \{\!|T_1|\!\}_{T_2}, P, E \gg \mathsf{let\,ok} = eq(x, \{\!|T_1|\!\}_{T_2})\,\mathsf{in}\,P, \{(\{\!|T_1|\!\}_{T_2}, x)\} \cup E}$$

Analogous cases for asymmetric encryption omitted.

$$\frac{T : s \text{ is a variable}}{x, T, P, E \gg \mathsf{let}\ T : s = x\,\mathsf{in}\,P, \{(T, T)\} \cup E}$$

Fig. 16. Receive Code Generation

$$\frac{(T, x) \in E}{E \vdash T}$$

$$\frac{E \vdash T_1 \qquad E \vdash T_2}{E \vdash \langle T_1, T_2\rangle}$$

$$\frac{E \vdash T_1 \qquad E \vdash T_2}{E \vdash \{\!|T_1|\!\}_{T_2}}$$

Fig. 17. Term Synthesis

A.6 Translation Relation

The relation $comp(N, C, P)$ relates a role number and the role's strand space trace with a process calculus term if

1. $C \rhd I$,
2. E_0 is an environment generated from I, and
3. $C, E_0, N, 1 \gg P, E_2$.

Note that a translation is interesting only if I induces a sensible interpretation of C.

Blanchet Initiator Example. Assume the initiator is the second role in the protocol. The initiator trace C is defined in Eq. 3. The initial environment generated from the first input set in Eq. 4 is:

$$E_0 = \{(pk(b), pk(b)), (pk(a)^{-1}, pk(a)^{-1}), (s, s)\},$$

where $b, a : \mathsf{N}$ and $s : \mathsf{S}$.

The process term P that satisfies $C, E_0, 2, 1 \gg P, E_2$, is:

$$\begin{array}{l}
\mathsf{pos}(2, 1). \\
\mathsf{out}(c, \{|\{|s|\}_{pk(a)^{-1}}|\}_{pk(b)}). \\
\mathsf{in}(c, x_1). \\
\mathsf{let}\, x_2 : \top = dec(x_1, s)\, \mathsf{in} \\
\mathsf{let}\, d : \mathsf{D} = x_2\, \mathsf{in} \\
\mathsf{pos}(2, 2).\ 0
\end{array}$$

Blanchet Responder Example. Assume the responder is the first role in the protocol. The responder trace is the one in Eq. 3 after interchanging sends and receives. A sensible set of input basic values is $\{d, pk(a), pk(b)^{-1}\}$. After inserting the new form by hand, the process term is:

$$\begin{array}{l}
\mathsf{in}(c, x_1). \\
\mathsf{let}\, x_2 : \top = dec(x_1, pk(b)^{-1})\, \mathsf{in} \\
\mathsf{let}\, x_3 : \top = dec(x_2, pk(a))\, \mathsf{in} \\
\mathsf{let}\, s : \mathsf{S} = x_3\, \mathsf{in} \\
\mathsf{pos}(1, 1). \\
\mathsf{new}\, d : \mathsf{D}. \\
\mathsf{pos}(1, 2). \\
\mathsf{out}(c, \{|d|\}_s).\ 0
\end{array}$$