A (restricted) quantifier elimination for security protocols *

R. Ramanujam †
The Institute of Mathematical Sciences, C.I.T. Campus,
Taramani, Chennai 600 113, India
E-mail: jam@imsc.res.in

S.P. Suresh
Chennai Mathematical Institute, SIPCOT IT Park,
Siruseri 603 103, India
E-mail: spsuresh@cmi.ac.in

Abstract

While reasoning about security protocols, most of the difficulty of reasoning relates to the complicated semantics (with freshness of nonces, multisessions etc). While logics for security protocols need to be abstract (without explicitly dealing with nonces, encryption etc), ignoring details may result in rendering any verification of abstract properties worthless. We would like the verification problem for the logic to be decidable as well, to allow for automated methods for detecting attacks. From this viewpoint, we study a logic with *session abstraction* and quantifiers over session names. We show that interesting security properties like secrecy and authentication can be described in the logic. We prove the existence of a normal form for runs of *tagged* protocols. This leads to a quantifier elimination result for the logic which establishes the decidability of the verification problem for tagged protocols, when we assume a fixed finite set of nonces.

1 Summary

Mechanisms for ensuring security typically use *encrypted communication*. However, even the use of the most perfect cryptographic tools does not always ensure the desired security goals. (See [AN95] for an illuminating account.) This situation arises primarily because of *logical flaws*

*We thank the anonymous referees whose insightful comments helped improve the presentation of this paper greatly.

†Corresponding author

in the design of protocols. It is widely acknowledged that security protocols are hard to analyze, bugs difficult to detect, and hence that it is desirable to look for automatic means by which *attacks* on protocols can be discovered.

Formally, this is a verification problem of the following kind: given a security protocol Pr and a security property α, we ask whether $\mathscr{M}(Pr) \models \alpha$: that is, whether all *runs* of the model $\mathscr{M}(Pr)$ associated with the protocol Pr satisfy α. There are two important issues here, that of how the model $\mathscr{M}(Pr)$ is defined, and the logic in which α is specified.

Models

The model $\mathscr{M}(Pr)$ typically defines an infinite state system, over which even simple reachability properties are undecidable. When there is no bound on the number of nonces the honest agents can generate, even though the (symbolic) terms generated during the runs are of bounded size, the intruder can force principals to code up complicated information using the nonces. On the other hand, when there is no bound on the size of the terms generated during a run, the intruder can get the agents to build longer and longer messages to code up complicated information, even when there is no generation of new data at all in any run of the protocol. These observations lie at the heart of the undecidability arguments endemic to the verification of security protocols (see [DLMS99] and [RS03c]). They also explain why most automated analysis of security protocols pertains to modelling intruder's knowledge (and therefore possible intruder actions).

Since even simple reachability properties are undecidable, we need to restrict the class of protocols studied, or verify security properties of a restricted model for protocols. When we impose such restrictions, we can hope to verify not just reachability properties, but a host of other properties as well. We do not yet have a characterisation of the exact combination of protocol subclass, protocol model, and class of properties, for which verification is decidable. But some approaches have been tried in the literature, and we highlight them here.

One approach is to bound the number of sessions that can appear in any run of the protocol. Essentially this approach is followed by [RT03], [MS01], and many others. Although a bound on the number of sessions automatically implies a bound on the number of nonces generated in each run, the problem is still nontrivial because the size of the terms involved need not be bounded. In [DLMS99] it is proved that if external bounds are placed *both* on the number of nonces generable in any run and on the size of the terms occurring in the runs, then one obtains decidability—in spite of not imposing explicit bounds on the number of sessions considered in each run.

A quite different approach is taken in [Low99]. Here syntactic restrictions on the protocol specifications are placed to derive bounds on the size of runs which constitute a violation of certain properties. But this work assumes *atomic substitutions* (nonce variables only replaced by nonces at run time, etc.). Later, in [HLS00] it is proved that with certain kinds of tagging schemes, one can altogether prevent *type flaw attacks* (which involve non-atomic substitutions). A much simpler tagging scheme is used in [BP03] (which contains the strongest results in this genre) to prove the termination of a logic-programming based algorithm to verify secrecy and some simple kinds of authentication.

In earlier work (see [RS03a], [RS03b], [Sur03] and [RS05a]) we proved that a restricted version of the secrecy problem (which is still useful in practice) is decidable for the class of tagged protocols even in the presence of unboundedly many nonces (assuming atomic substitutions). Our results involved proving some closure properties on the set of runs of a protocol. We proved that given a protocol, there exists a uniform bound K such that to every run of the protocol we can associate another run of the same protocol which uses only K new nonces, and which admits of an attack iff the original run does. The one drawback is that the decidability proofs do not (easily) generalise to properties other than the restricted secrecy we used in those papers.

Logic

This brings us to the consideration of the other side of the verification problem, to the security property being verified. Secrecy, confidentiality, integrity and authentication are typical properties studied, but there is a host of variations even among these. Often these properties are described in the context of specific protocols using elements particular to the structure of those protocols. Clearly, what is needed is an abstract logical language for specifying a range of security properties.

Ideally, we would like the logic to be *abstract*, in the sense that it should not refer to model elements (like encryption, which is really a mechanism for implementing security), and the model to be general enough so that the verification certificate is worthwhile.[1]

It is difficult to speak of any logic for security protocols without relating the work to BAN logic [BAN90], which initiated the study of belief logics for authentication. It also gave rise to a host of descendants (see [GNY90] and [AT91], for instance). While BAN logic has been greatly criticised ([Nes90], for example), its high level of abstraction and ease of use is acknowledged, and many protocol errors have been analysed using BAN logic. Verification tools based on BAN logic also exist [KW96]. One major limitation of BAN logic is its lack of a clear semantics, and even the subsequent attempts at giving it a precise semantics suffer from the fact that the details of the run generation mechanism are not explicated (which has a crucial bearing on decidability). It is this gap between what is specified and what is modelled that has led to what may be termed as "loss of faith" in belief logics.

This contribution

In this paper, we define a simple logic for security protocols with a decidable verification problem. One main aim has been to do this in such a way that the analysis techniques referred to above for automated analysis of specific properties like secrecy can be lifted to the range of properties definable in the logic. The most important issue in the design of the logic is that of expressiveness. We have been guided by two criteria in this matter: that of *decidable verification* and *minimality*. We have concentrated on (what we may call) a bare-bones logical framework, in which many interesting security properties can be expressed. In particular, we can consider protocols like the Needham-Schroeder protocol and specify properties like secrecy for the initiator

[1] This requirement of abstraction may be seen as similar to that in temporal logic in the context of verifying reactive systems.

in the logic.

Once again, since the models of formulas represent infinite state systems, we look for symbolic reasoning in the logic. This suggests an abstraction mechanism for the logic which preserves the essential details relevant to analysing intruder behaviour. For this, we propose the mechanism of *session abstraction*: basically, when we reason about the protocol, we have typical (legitimate) sessions in mind, and we postulate abstract session names as well as abstract names for the secrets used in that session. A system run is composed of (unboundedly) many such sessions, and we equip the logic with quantification over session names to constrain runs appropriately. To describe information transfer between sessions we use the equality relation.

A crucial implication of the session abstraction is that while temporal reasoning is limited to event occurrences within a session, quantification is used to constrain overall system behaviour. This corresponds to the standard specification of security protocols as a sequence of communications, thus denoting a typical session. Considerations by principals of intruder behaviour relates different potential instantiations of these sessions, and this is reflected in the quantified formulas of the logic proposed here.

Technically, our main idea is to prove a normal form for runs of so-called tagged protocols, when we can assume a fixed finite set of nonces, even when message length and number of parallel sessions are unbounded (yielding an infinite state system). The normal form allows us to place bounds on the witnesses attesting to the truth of session formulas, and hence leads to a quantifier elimination result for system formulas, thus yielding decidability.

It must be emphasized that the logic proposed here needs greater study before it can be touted as a formal language for the specification and design of protocols. In this paper, we only study the properties of the satisfaction relation $\mathscr{M}(Pr) \models \alpha$. What we need is a deduction system $Pr \vdash \alpha$, hopefully converging with the notion above. Discovery of such reasoning principles is important for guiding the structured design of security protocols.

Related work

In ([RS03a], [RS05a]), we showed the decidability of secrecy for a subclass of protocols in the presence of unbounded nonces, but when substitutions are constrained to be atomic (that is, only nonces can be substituted for nonces). In [RS03b], we showed that secrecy is decidable for tagged protocols in the presence of nonatomic substitutions and unbounded nonces.

On one hand, this paper can be seen as a generalization of the results of [RS03b] and [BP03] from secrecy to a range of security properties defined in the logic. On the other hand, the results here are obtained only in the presence of a fixed finite set of nonces, though there can be unboundedly many sessions, so these generalize The results of [RT03].

It is to be noted that while the analyses of [CLS03], [RT03], [BP03] and other similar work are carried out within logical frameworks, they cannot be considered logics for reasoning about security protocols. Every protocol is described as a theory and hence the specification of security properties intermingles with the security protocol being implemented. In contrast, we use logic as a specification language here and protocols are modelled as relational structures.

Coming to the logic, while there have been many logic-based analyses of security protocols

([ABKL93], [ADM03], [Pau98], [Bel03], [HP03], [HvdM03], [GMP92], [DMP03], [DDMP03], [ABV02] is a sample list), there have been few decidable logics which can be considered as specification logics.

In [RS05b], we study *logics of knowledge* for security protocols, and show decidability results similar to the one in this paper. The crucial departure here is that rather than use epistemic modalities, we use session abstraction and quantifiers over sessions in this paper. While the specifications are more elegantly stated in the knowledge logic, the logic here avoids many of the logical and philosophical complications arising from epistemic reasoning. A closely related work to [RS05b] is [HP03], which uses a logic of implicit and explicit knowledge to reason about adversaries. [HP03] go beyond Dolev-Yao adversaries and consider guessing by intruders as well.

2 The model

We briefly present our model for protocols in this section. Most of the elements of the model are standard in the literature on modelling security protocols. In particular, we use the Dolev-Yao adversary model [DY83]. In our model, protocols are specified just as in other models like strand spaces [FHG99] and multi-set rewriting [DLMS99].

Terms and actions

We start with a (potentially infinite) set of **agents** Ag, which includes the **intruder** I, and the others, who are called **honest agents**. Fix a countable set of **fresh secrets** N. (This includes *random, nonguessable nonces* as well as *temporary session keys*.) $\mathscr{B} \stackrel{\text{def}}{=} N \cup Ag$ is the set of **basic terms**.

The set of **information terms** is defined to be

$$\mathscr{T} \quad ::= \quad m \mid \mathit{public}(A) \mid \mathit{private}(A) \mid \mathit{shared}(A,B) \mid (t_1,t_2) \mid \{t_1\}_{t_2}$$

where m ranges over $\mathscr{B}$, A and B range over Ag, and t_1 and t_2 range over $\mathscr{T}$. Here $\mathit{public}(A)$, $\mathit{private}(A)$, and $\mathit{shared}(A,B)$ denote the public key of A, private key of A, and (long-term) shared key between A and B. Further, (t_1,t_2) denotes the pair consisting of t_1 and t_2, and $\{t_1\}_{t_2}$ denotes the term t_1 encrypted using t_2. These are the terms used in the message exchanges (which will be presently introduced).

The notion of **subterm** is defined as follows. For any term t, $ST(t)$, the set of its subterms, is the smallest set of terms T such that $t \in T$ and such that whenever $(r,r') \in T$ or $\{r\}_{r'} \in T$, $\{r,r'\} \subseteq T$. It is worth noting that, for instance, the encrypting key k is considered a subterm of $\{m\}_k$. An alternative definition which doesn't consider k to be a subterm of $\{m\}_k$ is also common in the literature. An **encrypted term** is a term of the form $\{r\}_{r'}$. The set of **encrypted subterms** of t is denoted $EST(t)$. We say that r' is the **encryptor** of the encrypted term $\{r\}_{r'}$. We use the letters $t, t', t_1, t_2, r, r', u, u', \ldots$ to range over $\mathscr{T}$. We further let m, n, etc. denote basic terms, and k, k', k_1, etc. denote arbitrary keys.

Note that we allow terms like $\{t\}_{\{t'\}_{t''}}$, where constructed terms like $\{t'\}_{t''}$ are used as encryption keys. Since any term can be used as an encryption key, we need to define the

notion of **inverse** for every term. This is along expected lines: $inv(public(A)) = private(A)$, $inv(private(A)) = public(A)$, and $inv(t) = t$ for all other terms t. Thus, $inv(inv(t)) = t$ for all terms t.

We model communication between agents by **actions**. An action is either a **send action** of the form $+(A, B, t)$ or a **receive action** of the form $-(A, B, t)$, where t is an arbitrary term, and A and B are agent names. For example, the first message exchange in the Needham-Schroeder protocol [NS78] is the sending of the message $\{A, x\}_{public(B)}$ by the *initiator* (denoted A here) to the *responder* (denoted B here). We model this message transfer by the action $+(A, B, \{A, x\}_{public(B)})$, where the first component of the tuple gives the sender's name, the second component gives the name of the recipient, and the third component is the message that is transferred. The receipt of the same message by B is modelled by the action $-(A, B, \{A, x\}_{public(B)})$. Here again, the first component denotes the sender and the second denotes the receiver. For any send action $a = +(A, B, t)$, $sender(a)$ denotes the agent who sends the message, namely A, and for any receive action $a = -(A, B, t)$, $receiver(a)$ denotes the agent who receives the message, namely B. We emphasize that while the sender name in a send action, and a receiver name in a receive action denote the actual agents that send and receive the messages, respectively, in a send action we can only name the *intended receiver*, and in a receive action we can only name the *purported sender*. As we will see later, every send action is an instantaneous receive by the intruder, and similarly, every receive action is an instantaneous send by the intruder.

We use the letters $a, b, a_1, b_1, a_2, b_2, a', b', \ldots$ to range over arbitrary actions. We denote the set of all actions by Act. For an action a of the form $+(A, B, t)$ or $-(A, B, t)$, we define $term(a)$ to be t.

Protocol specifications

A **role** is a finite sequence of actions. We consistently denote roles by the Greek letter η, with superscripts and/or subscripts. Typically, they represent the sequence of message exchanges of a particular agent participating in a protocol. A **parametrized role** $\eta[m_1, \cdots, m_k]$ is a role in which the *basic terms* $m_1, \ldots, m_k$ are singled out as parameters. The idea is that an agent participating in the protocol can execute many *sessions* of a role in the course of a single run, by instantiating the parameters in many different ways. All the basic terms occurring in a parametrized role that are not counted among the parameters are the **constants** of the role. They do not change their meaning over different sessions of the role. We denote the set of constants of a role η by $const(\eta)$. The notations $ST(\eta)$ and $EST(\eta)$ have the obvious meanings, given a role η. We also freely refer to encryptors occurring in η, with the obvious meaning.

Suppose $\eta = a_1 \cdots a_k$ is a parametrized role. We say that a nonce n **originates** at $i (\leq k)$ in η if:

- n is a parameter of η,
- a_i is a send action, and
- $n \in ST(term(a_i))$ and for all $j < i$, $n \notin ST(term(a_j))$.

If a nonce n originates at i in a role it means that the agent sending the message a_i uses n for the first time in the role. This usually means that, in any session of that role, the agent playing the role has to generate a fresh, nonguessable random number and send it as a challenge. Subsequent receipt of the same number in the same session plays a part in convincing the agent that the original message reached the intended recipient.

A **protocol** is just a finite set of parametrized roles $\{\eta_1, \ldots, \eta_n\}$. For example, the Needham-Schroeder protocol is given by two parametrized roles: the initiator role and the responder role. The initiator role $\eta_1[A, B, x, y]$ is as follows:

$$+(A, B, \{A, x\}_{public(B)}); \quad -(B, A, \{x, y\}_{public(A)}); \quad +(A, B, \{y\}_{public(B)}).$$

The responder role $\eta_2[A, B, x, y]$ is as follows:

$$-(A, B, \{A, x\}_{public(B)}); \quad +(B, A, \{x, y\}_{public(A)}); \quad -(A, B, \{y\}_{public(B)}).$$

Note that x originates at 1 in the initiator role and y originates at 2 in the responder role.

Protocols are denoted by Pr, with superscripts and/or subscripts. If $Pr = \{\eta_1, \ldots, \eta_n\}$, then $ST(Pr) \stackrel{\text{def}}{=} ST(\eta_1) \cup \cdots \cup ST(\eta_n)$. $EST(Pr)$ is defined similarly. Let $Pr = \{\eta_1[m_1^1, \cdots, m_{k_1}^1], \ \ldots, \ \eta_n[m_1^n, \cdots, m$ be a protocol. $const(Pr) \stackrel{\text{def}}{=} (ST(Pr) \cap \mathscr{B}) \setminus \{m_1^1, \ldots, m_{k_1}^1, \ldots, m_1^n, \ldots, m_{k_n}^n\}$ is the set of constants of Pr. These are terms whose interpretation stays constant during any run of the protocol.

Typically protocols are presented as a sequence of communications of the form $A \to B : t$, which denotes the sending of the message t by A and its receipt by B. To formally model the fact that the intruder can block messages, and also fake messages, one typically extracts the roles of each agents from such a sequence of communications, and considers interleavings of various sessions of the roles. For simplicity, we directly present protocols as roles.

While the definition of a protocol is as simple as it gets, it is worth noting that under the standard semantics, which we shall present shortly, the above definition is general and hides great expressive power. In fact, many of the undecidability results in security protocol verification exploit the fact that the above presentation of protocols allows arbitrary unification across terms, thus enabling the intruder to repeatedly transfer nontrivial information across sessions and thus code up Turing machine configurations and the like. One way of preventing this from happening is to *tag* the elements of the protocols with appropriate typing information.

A **tagged protocol** is a protocol $Pr = \{\eta_1, \ldots, \eta_n\}$ which satisfies the following conditions:

1. for every $t \in EST(Pr)$ there exists $c \in const(Pr)$ and terms t' and t'' such that $t = \{(c, t')\}_{t''}$, and

2. for any two terms $t_1 = \{(c_1, t_1')\}_{t_1''}$ and $t_2 = \{(c_2, t_2')\}_{t_2''}$ in $EST(Pr)$, if $c_1 = c_2$ then $t_1 = t_2$.

For example, the tagged version of the Needham-Schroeder protocol is given by the following two roles. The initiator role $\eta_1[A, B, x, y]$ is as follows:

$$+(A, B, \{c_1, A, x\}_{public(B)}); \quad -(B, A, \{c_2, x, y\}_{public(A)}); \quad +(A, B, \{c_3, y\}_{public(B)}).$$

The responder role $\eta_2[A, B, x, y]$ is as follows:

$$-(A, B, \{c_1, A, x\}_{public(B)}); \quad +(B, A, \{c_2, x, y\}_{public(A)}); \quad -(A, B, \{c_3, y\}_{public(B)}).$$

The main technical results of this paper pertain to tagged protocols. We postpone a discussion on the *taggability* of protocols, and the range and extent of the applicability of the technical results, to the end of the paper.

Before we move on to the semantics of protocols, we present an important property of tagged protocols which immediately follows from the definitions. To present it, we need the notion of *substitutions*. Substitutions are a central element of any formal model for protocols. The agent names and nonces mentioned in roles of a protocol are simply placeholders—the actual agents that take part in a protocol are usually processes running on behalf of users, and nonces are usually 128-bit random numbers. The same process could be engaged in many parallel sessions, playing one or more roles of the protocol with other processes. There are unboundedly many such processes participating in sessions of a protocol—even a protocol which mentions only two agent names. A formal analysis of a protocol should consider combinations of the sessions mentioned above to check whether an undesirable state has been reached—a state where the intruder has learnt a secret it is not supposed to have learnt, for instance. We use substitutions to keep track of the bindings of the concrete names to the names mentioned in the protocol.

A **substitution** σ is a map from $\mathscr{B}$ to $\mathscr{T}$ such that $\sigma(Ag) \subseteq Ag$ and $\sigma(I) = I$. A substitution σ is said to be **well-typed** if $\sigma(N) \subseteq N$. For any $T \subseteq \mathscr{B}$, σ is said to be a T-substitution iff for all $x \in \mathscr{B}$, $\sigma(x) \in T$. A substitution σ is **suitable for a parametrized role** η if:

- $\sigma(m) = m$ for all $m \in const(\eta)$, and
- $\sigma(n) \in N$ for all nonces n such that:
  - n originates at some point in η, or
  - n is a subterm of some t that occurs as an encryptor in Pr.

We say that σ is **suitable for a protocol** Pr if $\sigma(m) = m$ for all $m \in const(Pr)$. Substitutions are extended to terms as follows: $\sigma(public(A)) = public(\sigma(A))$, $\sigma(private(A)) = private(\sigma(A))$, $\sigma(shared(A, B)) = shared(\sigma(A), \sigma(B))$; $\sigma((t, t')) = (\sigma(t), \sigma(t'))$; and $\sigma(\{t\}_{t'}) = \{\sigma(t)\}_{\sigma(t')}$. Substitutions are extended to sets of terms, actions and sequences of actions in a straightforward manner. We say that two terms t_1 and t_2 are **unifiable** via a substitution σ if $\sigma(t_1) = \sigma(t_2)$.

Some aspects of the above definitions need explanation. In our formal analysis, we wish to consider *type-flaw attacks*, which involve the intruder instantiating names with complex terms. ([DMTY97] is an interesting paper which has nice examples of design flaws in protocols whose exploitation crucially uses type-flaw attacks.) But some restrictions need to be placed on such ill-typed substitutions for it to be a faithful modelling. For instance, the honest agents are in control of the nonces they produce, and hence nonces that originate at some point in the roles do not get substituted by complex terms. We also disallow ill-typed substitutions in "key positions", for technical convenience. This has the consequence that protocol analysis based on our model misses some kinds of type-flaw attacks. But it is vitally used in the proofs of some of the key

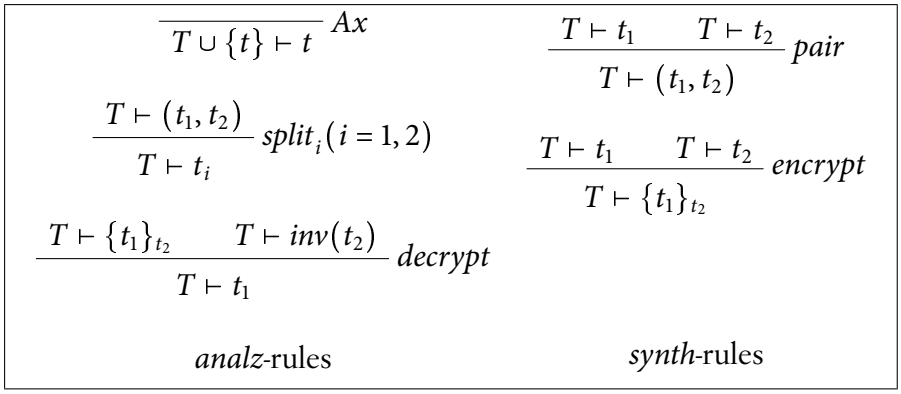


Figure 1: Message generation rules.

lemmas later. The same effect can be achieved by considering a slightly more stringent tagging scheme, at the cost of complicating some of the technical proofs.

Proposition 1 *Suppose* $Pr = \{\eta_1, \ldots, \eta_n\}$ *is a tagged protocol. Consider any two substitutions* σ *and* σ'*, and two terms* t *and* t' *from* $EST(Pr)$*. If* $\sigma(t) = \sigma'(t')$ *then* $t = t'$*.*

Proof: Suppose $\sigma(t) = \sigma'(t')$. Since we are considering encrypted subterms, it follows (by definition of tagged protocols) that $t = \{(c, t_1)\}_{t_2}$ and $t' = \{(c', t_1')\}_{t_2'}$ for some c, c', t_1, t_2, t_1' and t_2'. It follows that $\sigma(c) = \sigma'(c')$. But since $c, c' \in const(Pr)$ and σ and σ' are suitable for Pr, $\sigma(c) = c$ and $\sigma'(c') = c'$. It follows that $c = c'$. Therefore from the definition of tagged protocols, it follows that $t = t'$. ⊣

Message generation rules

We now detail the semantics of protocols. In subsequent sections, we give a transition system semantics for protocols, central to which are rules by which the agents generate new messages from old. There is a crucial difference between the manner in which the intruder generates new messages from old and the manner in which the honest agents do so. We first formalize the notion of message derivation for the intruder.

A **sequent** is of the form $T \vdash t$ where $T \subseteq \mathscr{T}$ and $t \in \mathscr{T}$. A **derivation** or a **proof** π of $T \vdash t$ is a tree whose nodes are labelled by sequents and connected by one of the *analz*-rules or *synth*-rules in Figure 1; whose root is labelled $T \vdash t$; and whose leaves are labelled by instances of the Ax rule. We will use the notation $T \vdash t$ to denote both the sequent, and the fact that it is derivable. For a set of terms T, $\overline{T} \stackrel{\text{def}}{=} \{t \mid T \vdash t\}$ is the *closure* of T.

When the honest agents participate in a protocol, the manner in which they deduce new messages is strictly in accordance with the protocol specification. For instance, suppose an agent A expects a term of the form $t = \{(x, \{y\}_p)\}_q$ at a particular point of its participation in a protocol. Let us say the term it actually receives at that point is $t' = \{(m, m'), \{(n, n')\}_p\}_q$, which is got by substituting (m, m') for x and (n, n') for y. Let us say that when A receives t', it has already learnt $inv(p)$ and $inv(q)$. If A followed the derivation system displayed in Figure 1, it

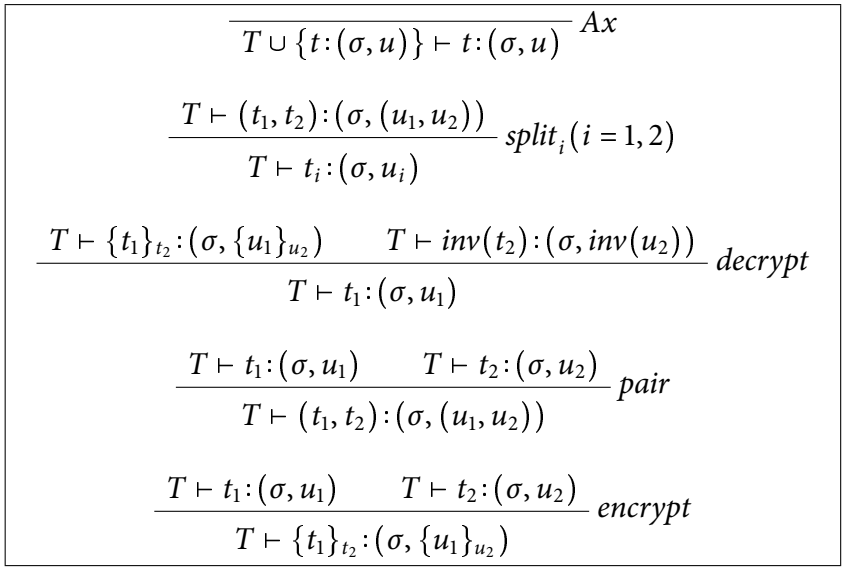


Figure 2: Deduction rules for honest agents (specific to *Pr*)

would be able to derive (m, m') and (n, n') and hence, also m, m', n, and n'. But that is not the intended behaviour of the honest agents. An honest agent shouldn't be able to derive m and m' in this case. It should stop at (m, m'), which replaces x, a term which is further unanalyzable. To formalize this we need the notion of a typed term and typed sequent specific to a protocol.

A **typed term** of a protocol *Pr* is of the form $t:(\sigma, u)$ where t is any term, σ is any substitution suitable for *Pr*, and $u \in ST(Pr)$ such that $t = \sigma(u)$. A **typed sequent** is of the form $T \vdash t:(\sigma, u)$, for a set $T \cup \{t:(\sigma, u)\}$ of typed terms of *Pr*. Figure 2 gives the message derivation rules to be followed by the honest agents.

Even though there is a distinction between the message derivation capabilities of the intruder and that of the honest agents, we will mostly be interested in the intruder's message derivation capabilities. Much of the technical work in the later sections involves reasoning about the intruder's derivations. There are a few places where the difference between the two derivation systems becomes manifest. We handle intruder derivations and honest agent derivations separately in such cases. But in general, we concern ourselves with intruder derivations. In fact, we even use the same notations for intruder derivations and honest agent derivations, relying on the context to remove any ambiguities.

We have introduced the notion of typed sequents and derivations to faithfully model the behaviour of the honest agents. While this is a typical approach in the literature, a case can be made that even these typed rules attribute more power to the honest agents than desired. Consider a protocol specification which says that *B* should receive from *A* a message of the form $\{x\}_{public(C)}$, which he should then pass on to *C*. These are called *blind copies* in the literature. Now *B* cannot decrypt any message encrypted by the public key of *C*, and thus cannot distinguish between $\{m\}_{public(C)}$ and an arbitrary term t. So, a session in which *B* receives an arbitrary n from the intruder and passes it on to *C* should also be deemed to be an acceptable behaviour

according to this protocol. Our model rules out such a scenario by allowing agents to send and receive only substitution instances of the messages occurring in the protocol.

One way to model such situations is to expand the set of typed terms used in honest agent deductions to include (anomalous) typed terms of the form $t : \{t'\}_{public(A)}$ (where t is an arbitrary term), and including rules which allow the honest agents to detect such type anamolies in some situations (where the agent has the corresponding key and knows what the result of the decryption should be). We do not explore this interesting direction in this paper.

Transition system of a protocol

We have informally referred to the notion of a session previously, to motivate some of the other definitions. We make the notion precise here.

Let $Pr = \{\eta_1, \ldots, \eta_n\}$ be a protocol. The tuple $\Pi = (id, \eta, \sigma)$ is a **session** of Pr if:

- $id \in \mathbb{N}$ is the *session identifier* of Π,
- $\eta = \eta_i$ for some $i \in \{1, \ldots, n\}$, and
- σ is a substitution suitable for Pr and η.

We use the notation $t@\Pi$ to denote $\sigma(t)$, the meaning of t in the session Π.

Let $\Pi = (id, \eta, \sigma)$. Π is said to be a well-typed session if σ is well-typed, and Π is said to be a T-session if σ is a T-substitution, for $T \subseteq \mathscr{B}$.

An **event** of $Pr = \{\eta_1, \ldots, \eta_n\}$ is a pair $e = (\Pi, lp)$ where $\Pi = (id, \eta, \sigma)$ is a session of Pr and $1 \leq lp \leq |\eta|$. For an event $e = (\Pi, lp)$ with $\Pi = (id, \eta, \sigma)$, we let $act(e)$ and $term(e)$ denote $\sigma(act(\eta(lp)))$ and $term(act(e))$, respectively. For events $e = (\Pi, lp)$ and $e' = (\Pi', lp')$ of Pr, we say that $e \prec e'$ (meaning that e is in the *local past* of e') if $\Pi = \Pi'$ and $lp < lp'$. An event $e = (\Pi, lp)$ is a well-typed event if Π is a well-typed session, and it is a T-event if Π is a T-session.

We say that a nonce n is **uniquely originating** in a set of events E of a protocol Pr if there is at most one event $((id, \eta, \sigma), lp)$ of E and at most one nonce m such that m originates at lp in η and $\sigma(m) = n$. (Note that the fact m originates in η implies that m is a parameter of η.)

An **information state** (or just *state*) is a tuple $(s_A)_{A \in Ag}$, where $s_A \subseteq \mathscr{T}$ for each $A \in Ag$. The **initial state** of Pr, denoted by $initstate(Pr)$, is the tuple $(s_A)_{A \in Ag}$ such that for all $A \in Ag$,

$$s_A = const(Pr) \cup Ag \cup \{private(A)\} \cup \{public(B), shared(A, B) \mid B \in Ag\}.$$

When the protocol is clear from the context, $initstate(Pr)$ is denoted simply by $initstate$.

Given a protocol $Pr = \{\eta_1, \ldots, \eta_n\}$, its **transition system**, $Sys(Pr)$, is given by $(\mathscr{S}, init, \longrightarrow)$, where:

- $\mathscr{S} \stackrel{\text{def}}{=} \{(E, (s_A)_{A \in Ag}) \mid E$ is a $\prec$-closed set of events such that every nonce is uniquely originating in E, and $s_A \subseteq \mathscr{T}$ for each $A \in Ag\}$ is the set of **control states**.
- $init = (\varnothing, initstate(Pr))$ is the **initial control state**.

- $\longrightarrow \subseteq \mathscr{S} \times Act \times \mathscr{S}$ is defined as follows: $(E, (s_A)_{A \in Ag}) \xrightarrow{a} (E', (s'_A)_{A \in Ag})$ iff there is an event e of Pr such that:
  - $a = act(e)$,
  - $e \notin E$,
  - $E' = E \cup \{e\}$,
  - for all e' such that $e' \prec e$, $e' \in E$,
  - if a is a send action, then
    - $s'_I = s_I \cup \{term(a)\}$, and
    - $s'_B = s_B$ for all agents B other than I,
  - if a is a receive action with $receiver(a) = A$, then
    - $term(a) \in \overline{s_I}$,
    - $s'_A = s_A \cup \{term(a)\}$, and
    - $s'_B = s_B$ for all agents B other than A.

Let Pr be a protocol, and let $Sys(Pr) = (\mathscr{S}, init, \longrightarrow)$ be its transition system. A **run** of Pr is a path in $Sys(Pr)$, i.e., a sequence $\xi = (E_0, s_0)\ a_1\ (E_1, s_1)\ a_2\ \cdots\ a_k(E_k, s_k)$, where for each $i \leq k$, $(E_i, s_i) \in \mathscr{S}$; $(E_0, s_0) = init$; and for all $i : 0 < i \leq k$, $(E_{i-1}, s_{i-1}) \xrightarrow{a_i} (E_i, s_i)$.

As an example, we present a run of the Needham-Schroeder protocol, which essentially models the famous Lowe's attack [Low96] on the Needham-Schroeder protocol. Recall that the initiator role $\eta_1[A, B, x, y]$ is given by:

$$+(A, B, \{A, x\}_{public(B)}); \quad -(B, A, \{x, y\}_{public(A)}); \quad +(A, B, \{y\}_{public(B)}),$$

and that the responder role $\eta_2[A, B, x, y]$ is given by:

$$-(A, B, \{A, x\}_{public(B)}); \quad +(B, A, \{x, y\}_{public(A)}); \quad -(A, B, \{y\}_{public(B)}).$$

Let us define two sessions Π_1 and Π_2 by: $\Pi_1 \stackrel{\text{def}}{=} (1, \eta_1, \sigma_1)$ and $\Pi_2 \stackrel{\text{def}}{=} (2, \eta_2, \sigma_2)$, where $\sigma_1(A) = A$, $\sigma_1(B) = I$, $\sigma_1(x) = m$ and $\sigma_1(y) = n$, and $\sigma_2(A) = A$, $\sigma_2(B) = B$, $\sigma_2(x) = m$ and $\sigma_2(y) = n$. We see that the following is a run of the Needham-Schroeder protocol, for some appropriate information states $s_0, \ldots, s_6$: Instead of displaying the set of events in each control state, we just display the event which is added in that control state.

$$\begin{array}{ll}
(\varnothing, s_0) & \\
+(A, I, \{A, m\}_{public(I)}) & ((\Pi_1, 1), s_1) \\
-(A, B, \{A, m\}_{public(B)}) & ((\Pi_2, 1), s_2) \\
+(B, A, \{m, n\}_{public(A)}) & ((\Pi_2, 2), s_3) \\
-(I, A, \{m, n\}_{public(A)}) & ((\Pi_1, 2), s_4) \\
+(A, I, \{n\}_{public(I)}) & ((\Pi_1, 3), s_5) \\
-(A, B, \{n\}_{public(B)}) & ((\Pi_2, 3), s_6)
\end{array}$$

Note that as far as B is concerned, he is just having a normal session with A. But A is actually talking to I in parallel (as part of another legitimate session), who uses information from one run in the other cleverly. At the end of $((\Pi_1, 3), s_5)$, the intruder gets to know n, which was intended to be secret between A and B. Thus B has no guarantee of its nonce remaining secret in this run. *In fact, even when B is taking part in a legitimate run in which he participates in just one session supposedly with A, he cannot rule out the fact that the run he is taking part in is the above one in which A talks to I, and hence cannot be assured of the secrecy of n.*

The definition of the transition system of a protocol is fairly straightforward. The idea is that each agent maintains a control state, which records the messages learnt by the agent by previous communications. Further, a record is kept of all the sessions in which each agent is participating at present, and the progress made in each session. For technical convenience, we formally model this information as a global pool of events that have happened till now, rather than as a set of sessions for each agent. But there are some conditions on the set of events which is part of a control state. Since it represents the set of sessions all the agents are involved in, and the amount of progress in each session, it has to be $\prec$-closed. Further, we assume that there is a dependable *nonce generator* which generates a fresh nonce on each invocation (across agents). Thus, the unique origination property holds for every nonce with respect to this set of events.

There is only one initial state for a given protocol. No agent has started participating in any session initially, but has some *initial knowledge*. This consists of all the constant terms of the protocol, the names of all the agents, all the public keys, the agent's own private key, and all keys the agent shares with others.

Whenever a message communication happens, it is as part of some session. Thus the control state is updated by the addition of some appropriate event. Notice that all the previous actions in the same session should have already been carried out, and the current action shouldn't have already been performed in this session. Further, if the action is a send action, then it is always intercepted by the intruder (who plays the role of the network–perhaps much more!), who may later choose to pass it on to the intended receiver with or without modifications, and under any assumed name. Thus only the intruder's information state is updated on a send action.

A receive action by some agent is as a result of a send by the intruder, who is either sending the message on its own, or is relaying an earlier message sent by some other agent. Thus we require that the message actually be derivable from all the messages that the intruder has learnt. Further the information state of the receiver of the message gets updated. The condition on derivability of the message being communicated is crucial. We wish to consider all possible attack scenarios on the protocol, but if the model allowed the intruder to send arbitrary messages it wouldn't be a faithful modelling of intruder behaviour. Thus, while the communication capabilities of the intruder are very powerful, its message generation capabilities is limited by the rules we presented in the previous subsection.

An alternative view of runs

It can be noticed that the only source of non-determinism in the transition system of a protocol is in guessing the session of which the next action is part. Once we have determined the event

with which the next action is associated, the state update is deterministic. Thus we can with advantage view a run as given by a sequence of events. The relevant definitions are given below.

The notions of an action **enabled** at a state, and $update(s, a)$, the **update** of a state s on an action a, are defined as follows:

- A send action a is always enabled at any state s.
- A receive action a is enabled at s iff $term(a) \in \overline{s_I}$.
- $update(s, +(A, B, t)) \stackrel{\text{def}}{=} s'$ where $s'_I = s_I \cup \{t\}$, and for all agents C other than I, $s'_C = s_C$.
- $update(s, -(A, B, t)) \stackrel{\text{def}}{=} s'$ where $s'_B = s_B \cup \{t\}$ and for all agents C other than B, $s'_C = s_C$.

$update(s, \eta)$ for a state s and a sequence of actions η is defined in the obvious manner.

Given a protocol $Pr = \{\eta_1, \ldots, \eta_n\}$ and a sequence $\xi = e_1 \cdots e_k$ of events of Pr, $infstate(\xi)$ is defined to be $update(initstate(Pr), act(e_1) \cdots act(e_k))$.

Given a protocol $Pr = \{\eta_1, \ldots, \eta_n\}$, a sequence $e_1 \cdots e_k$ of events of Pr is said to be an **admissible sequence of events** of Pr iff the following conditions hold:

- for all $i, j \leq k$ such that $i \neq j$, $e_i \neq e_j$,
- for all $i \leq k$ and for all $e \prec e_i$, there exists $j < i$ such that $e_j = e$,
- for all $i \leq k$, $act(e_i)$ is enabled at $infstate(e_1 \cdots e_{i-1})$, and
- every nonce is uniquely originating in $\{e_1, \ldots, e_k\}$.

Runs and admissible sequences of events are closely related. With every admissible sequence of events $\xi = e_1 \cdots e_k$, we can associate the *unique* run $run(\xi) = (E_0, s_0)\ a_1\ (E_1, s_1)\ a_2\ \cdots\ a_k\ (E_k, s_k)$, where $E_i = \{e_1, \ldots, e_i\}$, $s_i = infstate(e_1 \cdots e_i)$, and $a_i = act(e_i)$.

On the other hand, given a run $\xi = (E_0, s_0)\ a_1\ (E_1, s_1)\ a_2\ \cdots\ a_k\ (E_k, s_k)$, it is clear from the definition that for every $i : 1 \leq i \leq k$, there is a unique e_i such that $E_i = E_{i-1} \cup \{e_i\}$, and such that $a_i = act(e_i)$. It is also clear from the definition of *infstate* that for each i, $s_i = infstate(e_1 \cdots e_i)$. Thus, with every such run ξ, we can associate a *unique* admissible sequence of events $evseq(\xi) = e_1 \cdots e_k$, where each $e_i \in E_i \setminus E_{i-1}$. The following proposition is an immediate consequence of the definitions.

Proposition 2 *For every run ξ, $evseq(\xi)$ is an admissible sequence of events. For every admissible sequence ξ of events, $run(\xi)$ is a run. Further, for every run ξ, $run(evseq(\xi)) = \xi$, and similarly for every admissible sequence ξ of events, $evseq(run(\xi)) = \xi$.*

We find it convenient to work with admissible sequences of events rather than directly with runs. But in the rest of the paper, we refer to them only as runs.

For any $T \subseteq \mathscr{B}$, ξ is said to be a T-**run** if e_i is a T-event for every $i \leq k$. It is said to be a **well-typed run** if e_i is a well-typed event for every $i \leq k$.

We say that Π is a *session of* ξ if an event of the form (Π, lp) occurs in ξ, for some lp. We let $Sessions(\xi)$ denote the set of all sessions of ξ.

Executable protocols

Consider a protocol which has the following role:

$$-(A, B, \{x\}_{public(C)}); \quad +(B, A, x).$$

There is nothing in our definitions that disallows protocols with such roles, and we need to ask what the runs of this protocol will look like. The above role can be performed to completion by B only if the instance of x in the first message he receives is an m which is already known to B (due to some earlier message exchanges). This is because there is no way B can decrypt the first message. Such a role can be performed to completion only by accident, as it were. In other words, these protocols are not always *executable*.

We can easily define the notion of executability of protocols. A protocol $Pr = \{\eta_1, \ldots, \eta_n\}$ is **executable** if for each of its roles $\eta = a_1 \cdots a_\ell$, each send action a_i of η is enabled at the state $update(initstate(Pr), a_1 \cdots a_{i-1})$.

While executability is reasonable, we do not make this assumption in our model, mainly because our technical results do not depend on it. However, most protocols arising in practice are executable in this sense.

3 A logic with session abstraction

Syntax

We assume a countable set *SesNames* of *abstract session names*. The set of formulas is given by a two-level syntax. The **session formulas** are given by the following syntax (where $A \in Ag$, $m \in N$ and a is an action):

$$\Phi_0 ::= A \; has \; m \mid \tau_a \mid \neg\alpha \mid \alpha \vee \beta \mid \mathbf{F}\alpha \mid \mathbf{P}\alpha.$$

The **run formulas** are given by the following syntax (where $ses \in SesNames$, $m \in \mathscr{B}$, and $\alpha \in \Phi_0$):

$$\Phi ::= \alpha@ses \mid m@ses = m'@ses' \mid \neg\varphi \mid \varphi \vee \psi \mid (\exists ses)\varphi.$$

The duals of the modalities and quantifiers are defined in the standard manner: $\mathbf{G}\alpha \stackrel{\text{def}}{=} \neg\mathbf{F}\neg\alpha$, $\mathbf{H}\alpha \stackrel{\text{def}}{=} \neg\mathbf{P}\neg\alpha$, and $(\forall ses)\varphi \stackrel{\text{def}}{=} \neg(\exists ses)\neg\varphi$.

The formula $A \; has \; m$ means that the agent playing A's role in the current session has the nonce (or session key, as the case may be) standing for the abstract nonce name m in its database. The formula τ_a (where a is an action) means that the current action in the current session is the action standing for a. $\mathbf{F}\alpha$ means that there is a later event of the run which also belongs to the same session as the current event, and in which α is true. $\mathbf{G}\alpha$ means that at all later events of the run which belong to the same session as the current event, α is true. $\mathbf{P}\alpha$ and $\mathbf{H}\alpha$ refer to the past and are the analogues of $\mathbf{F}\alpha$ and $\mathbf{G}\alpha$ respectively. The formula $\alpha@ses$ asserts that α holds at the beginning of the session named by ses. The equality assertion $m@ses = m'@ses'$ states that the same basic term has been used to instantiate both m in the session named by ses and m' in the session named by ses'. The meaning of the quantified formulas $(\exists ses)\varphi$ and $(\forall ses)\varphi$ is standard.

The set of all subformulas of a session formula α is denoted $sf(\alpha)$, and the set of all subformulas of a run formula φ is denoted $sf(\varphi)$. $sf(\alpha)$ is the least set of session formulas X such that $\alpha \in X$, $(\neg\beta \in X$ or $\mathbf{F}\beta \in X$ or $\mathbf{P}\beta \in X) \Rightarrow \beta \in X$, and $\beta \vee \beta' \in X \Rightarrow \{\beta, \beta'\} \subseteq X$. Similarly $sf(\varphi)$ is the least set of run formulas X such that $\varphi \in X$, $(\neg\psi \in X$ or $(\exists ses)\psi \in X) \Rightarrow \psi \in X$, and $\psi \vee \psi' \in X \Rightarrow \{\psi, \psi'\} \subseteq X$.

The set of *terms referred to by* φ, denoted $refterms(\varphi)$, is defined as follows:

- $refterms(\alpha@ses) \stackrel{\text{def}}{=} \{m@ses \mid A \text{ has } m \in sf(\alpha) \text{ for some } A \in Ag\}$.
- $refterms(m@ses = m'@ses') \stackrel{\text{def}}{=} \{m@ses, m'@ses'\}$.
- $refterms(\neg\varphi) \stackrel{\text{def}}{=} refterms(\varphi)$.
- $refterms(\varphi \vee \psi) \stackrel{\text{def}}{=} refterms(\varphi) \cup refterms(\psi)$.
- $refterms((\exists ses)\varphi) \stackrel{\text{def}}{=} refterms(\varphi)$.

Semantics

We first present the semantics of session formulas. They are interpreted over sessions of a run. But actions of a session do not occur consecutively in a run. Between two actions of a session, the other agents (or maybe the same agent) engage in actions of other sessions, thus learning more terms. In particular, the knowledge of the intruder has the potential to change between successive actions of a session. Moreover, this change is not completely determined by the current session under consideration. Thus, when we reason about a particular session, we need to consider the context in which the session is played. This leads us to the following definition.

For a protocol $Pr = \{\eta_1, \ldots, \eta_n\}$ and one of its sessions $\Pi = (id, \eta, \sigma)$, we define a Π-**scenario** to be a sequence $\zeta = s_1\ (\Pi, 1)\ \cdots\ s_r\ (\Pi, r)\ s_{r+1}$, where each s_i is an information state, $r \leq |\eta|$, and $s_{r+1} = update(s_r, act((\Pi, r)))$.

Given a run $\xi = e_1 \cdots e_k$ of Pr (with $e_i = (\Pi_i, lp_i)$ and $s_i = infstate(e_1 \cdots e_i)$ for all $i \leq k$) and a session Π of ξ, we define $\xi{\restriction}\Pi$ to be the sequence $s_{i_1}\ e_{i_1} \cdots s_{i_r}\ e_{i_r}\ s_{i_r+1}$, where $1 \leq i_1 < \cdots < i_r \leq k$ and $\{i_1, \ldots, i_r\} = \{j : 1 \leq j \leq k \mid \Pi_j = \Pi\}$. Observe that $\xi{\restriction}\Pi$ is a Π-scenario.

Given a Π-scenario $\zeta = s_1\ (\Pi, 1) \cdots s_r\ (\Pi, r) s_{r+1}$ (where $\Pi = (id, \eta, \sigma)$ is a session of Pr), any $i : 1 \leq i \leq r+1$, and a session formula α, we describe when it is the case that $\zeta, i \vDash \alpha$.

- $\zeta, i \vDash A \text{ has } m$ iff $m@\Pi \in \overline{(s_i)_{A@\Pi}}$.
- $\zeta, i \vDash \tau_a$ iff $i \leq r$ and $\eta(i) = a$.
- $\zeta, i \vDash \neg\alpha$ iff $\zeta, i \nvDash \alpha$.
- $\zeta, i \vDash \alpha \vee \beta$ iff $\zeta, i \vDash \alpha$ or $\zeta, i \vDash \beta$.
- $\zeta, i \vDash \mathbf{F}\alpha$ iff there exists $j : i \leq j \leq r+1$ such that $\zeta, j \vDash \alpha$.

- $\zeta, i \vDash \mathbf{P}\alpha$ iff there exists $j : 1 \leq j \leq i$ such that $\zeta, j \vDash \alpha$.

Run formulas are interpreted over runs of protocols equipped with a *(session) assignment*. An assignment is a function *assign* which maps abstract session names to sessions of *Pr*. We say that *assign* is an assignment over ξ (where ξ is a run of *Pr*) if $assign(ses)$ is a session of ξ for every $ses \in SesNames$. For ease of notation, we sometimes say $assign_{ses}$ instead of $assign(ses)$. For an assignment *assign*, abstract session names $ses_1, \ldots, ses_n \in SesNames$, and sessions $\Pi_1, \ldots, \Pi_n$ of *Pr*, we define $assign[ses_1 := \Pi_1, \ldots, ses_n := \Pi_n]$, an **update of the assignment** *assign*, to be the assignment *assign*′ such that:

$$\text{for all } ses \in SesNames : assign'(ses) = \begin{cases} \Pi_i & \text{if } ses = ses_i \quad (1 \leq i \leq n) \\ assign(ses) & \text{if } ses \neq ses_i \text{ for all } i \leq n. \end{cases}$$

We say that an assignment is *compatible* with a run formula φ if:

$$\text{for all } m@ses \in refterms(\varphi) : m \in N \Rightarrow assign_{ses}(m) \in N.$$

Given a run $\xi = e_1 \cdots e_k$ of *Pr* (with $e_i = (\Pi_i, lp_i)$ for all $i \leq k$), and an assignment *assign* over ξ which is compatible with φ, we describe when it is the case that $\xi \vDash_{assign} \varphi$, for a run formula φ.

- $\xi \vDash_{assign} \alpha@ses$ iff $\xi\restriction\Pi, 1 \vDash \alpha$, where $\Pi = assign(ses)$.
- $\xi \vDash_{assign} m@ses = m'@ses'$ iff $m@assign_{ses} = m'@assign_{ses'}$.
- $\xi \vDash_{assign} \neg\varphi$ iff $\xi \nvDash_{assign} \varphi$.
- $\xi \vDash_{assign} \varphi \vee \psi$ iff $\xi \vDash_{assign} \varphi$ or $\xi \vDash_{assign} \psi$.
- $\xi \vDash_{assign} (\exists ses)\varphi$ iff there exists a session Π of ξ such that $\xi \vDash_{assign[ses:=\Pi]} \varphi$.

- We say that φ is *satisfiable over* ξ if $\xi \vDash_{assign} \varphi$ for some assignment *assign* over ξ which is compatible with φ.
- We say that φ is *valid over* ξ if $\xi \vDash_{assign} \varphi$ for all assignments *assign* over ξ which are compatible with φ.
- We say that φ is *Pr-satisfiable* if it is satisfiable over some run ξ of *Pr*, and that it is *Pr-valid* if it is valid over all runs ξ of *Pr*.
- We write $Pr \vDash \varphi$ to denote the fact that φ is *Pr*-valid.
- We say that $Pr \vDash_{wt} \varphi$ iff φ is valid over all *well-typed runs* ξ of *Pr*.
- We say that $Pr \vDash^{T} \varphi$ for a fixed set $T \subseteq \mathscr{B}$ if φ is valid over all *T-runs* ξ of *Pr*.
- We also denote by $Pr \vDash^{T}_{wt} \varphi$ the fact that φ is valid over all *well-typed T-runs* ξ of *Pr*.

Observe that the meaning of the quantified formula $(\exists ses)\varphi$ is given in terms of the sessions occurring within only one run. This might appear a little weak, since runs are of finite length and only finitely many sessions can occur in a run. But in fact, it is quite powerful. For instance, $(\exists ses)(\alpha \wedge A \text{ has } m)@ses$ asserts the occurrence of a session satisfying α in a context where A has learnt the secret m. This is a nontrivial assertion about the causal past of the session, and its truth varies depending on the run and the context in which the session appears.

Further, even though it appears that $(\exists ses)\varphi$ is equivalent to a disjunction of simpler formulas when considered in the context of a single run, we cannot simply eliminate the quantifier when we consider the problem of whether all runs of a protocol satisfies the formula. For that, we have to prove that the general problem reduces to a restricted version where we only consider runs whose sessions come from a fixed finite set of sessions. We handle this in later sections.

Properties of the Needham-Schroeder protocol

We now see how to state several desirable properties of the Needham-Schroeder protocol in our logic. Since $\sigma(I) = I$ for all substitutions σ, $I@\Pi = I$ for all sessions Π of a protocol Pr. This allows us to say I instead of $I@ses$ in all the formulas which follow. Recall that the initiator role $\eta_1[A, B, x, y]$ is given by:

$$+(A, B, \{A, x\}_{public(B)}); \quad -(B, A, \{x, y\}_{public(A)}); \quad +(A, B, \{y\}_{public(B)}),$$

and that the responder role $\eta_2[A, B, x, y]$ is given by:

$$-(A, B, \{A, x\}_{public(B)}); \quad +(B, A, \{x, y\}_{public(A)}); \quad -(A, B, \{y\}_{public(B)}).$$

We denote the actions of the initiator role by i_1, i_2, i_3, and the actions of the responder role by r_1, r_2, r_3.

We introduce the following abbreviations to improve the readability of the example formulas. For session formulas α and β, and abstract session names ses and ses':

- $(\neg\alpha)@ses \stackrel{\text{def}}{=} \neg(\alpha@ses)$.
- $(\alpha \vee \beta)@ses \stackrel{\text{def}}{=} \alpha@ses \vee \beta@ses$.
- $(A \neq I)@ses \stackrel{\text{def}}{=} \neg(A@ses = I)$.
- For a sequence of terms $t_1, \ldots, t_n$,

$$(t_1, \cdots, t_n){=}@(ses, ses') \stackrel{\text{def}}{=} \bigwedge_{i \leq n} (t_i@ses = t_i@ses').$$

An important property that we desire of this protocol is *secrecy*. There are two desirable secrecy requirements in this case. *Secrecy for the initiator* says that all fresh nonces that are instantiated

for x and not intended for the intruder are not leaked to the intruder. It is expressed by the following formula:

$$secrecy_{init} \stackrel{\text{def}}{=} (\forall ses)((B \neq I) \supset \mathbf{G}(\tau_{i_1} \supset \mathbf{G}\neg(I \ has \ x)))@ses.$$

Secrecy for the responder says that all fresh nonces that are instantiated for y and are not intended for the intruder are not leaked to the intruder. A simple way to express it is the following:

$$secrecy_{resp} \stackrel{\text{def}}{=} (\forall ses)((A \neq I) \supset \mathbf{G}(\tau_{r_2} \supset \mathbf{G}\neg(I \ has \ y)))@ses.$$

Authentication for the initiator says that every nonce received by the initiator purportedly from an honest agent is actually sent by the corresponding honest agent in the past, intended for initiator.

$$(\forall ses)[(B \neq I \wedge \mathbf{F}\tau_{i_2})@ses \supset (\exists ses')[(A, B, x, y)=@(ses, ses') \wedge (\mathbf{F}\tau_{r_2})@ses']].$$

Authentication for the responder says something similar for the responder.

$$(\forall ses)[(B \neq I \wedge \mathbf{F}\tau_{r_3})@ses \supset (\exists ses')[(A, B, x, y)=@(ses, ses') \wedge (\mathbf{F}\tau_{i_3})@ses']].$$

4 Facts about message derivations

The main result of this paper is that the problem of checking whether a run formula is true in all runs of a given *tagged protocol* is decidable. Crucial to the decidability result is an analysis of the message generation capabilities of the intruder, and an analysis of some weaker proof systems for generating messages in the case of tagged protocols. In this section, we present some basic results on the proof systems for message generation.

The first important property that we observe is that the problem of deciding whether $t \in \overline{T}$ for a given set of terms $T \cup \{t\}$ is decidable in polynomial time (see [RT03], for instance).

The derivation system presented in Figure 1 allows "non-normal" derivations like the following:

$$\dfrac{\dfrac{\dfrac{}{\{t\} \vdash t}\,Ax \quad \dfrac{}{\{t\} \vdash t}\,Ax}{\{t\} \vdash (t, t)}\,pair}{\{t\} \vdash t}\,split$$

The possibility of such derivations makes it much harder to analyse proofs structurally. Hence we would like to transform any proof into a proof that does not involve such a "non-normality".

For some of the technical results of the later sections, analysing proofs ending with a *synth*-rule is far simpler than analysing those ending with an *analz*-rule. Specifically, we will find it much easier to analyse applications of the *decrypt* rule if the principal premise (the premise of the form $T \vdash \{r\}_{r'}$) is got by the application of an *analz*-rule and the non-principal premise (the

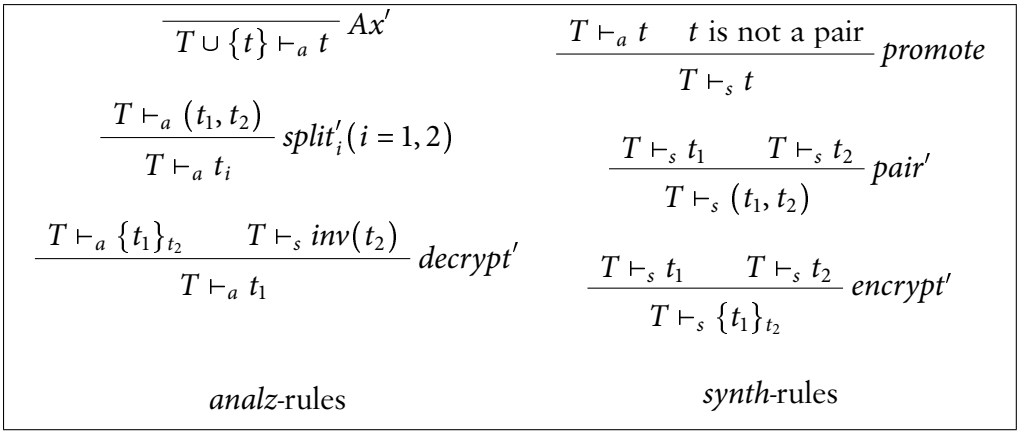


Figure 3: Alternative proof system.

premise of the form $T \vdash inv(r'))$ is got by the application of a *synth*-rule. This also implies that when we use the term (t, t') to decrypt $\{r\}_{(t,t')}$, and when (t, t') itself is derived by an *analz*-rule, we would like to first split it into t and t' and then pair them up to get (t, t') by a *synth*-rule, and only then use it to decrypt $\{r\}_{(t,t')}$. Fortunately, we can always transform any arbitrary derivation to such "normal" derivations.

Rather than work with the same derivation system, we find it more convenient to introduce a new derivation system, which will allow only normal proofs of the kind mentioned above. Further, it is equivalent to the old system. The new system involves two kinds of sequents: $T \vdash_a t$ and $T \vdash_s t$. The rules are given in Figure 3.

It is quite easy to see that whenever there is a proof of $T \vdash_s t$ there is also a proof of $T \vdash t$. The converse also holds, as stated by the following proposition, whose proof is presented in the Appendix.

Proposition 3 *If there is a proof π of $T \vdash t$ then there is a proof π' of $T \vdash_s t$.*

This normalisation result is not new. In fact, it is a basic result in the context of natural deduction for propositional logic [GLT89]. In the context of security protocols, essentially the same proof (presented in different formalisms) can be found in [CLS03], [RT03], [MCJ97], and [GF01]. The proofs in [MCJ97] and [GF01] assume atomic keys, but the results are easily generalisable.

The following two properties follow from an easy induction on proofs.

Proposition 4 *If $T \vdash_a t$ is derivable then $t \in ST(T)$.*

Proposition 5 *Suppose π is a proof of $T \vdash_s t$. For every term r occurring in π, $r \in ST(T \cup \{t\})$.*

Proposition 6 *For any set of terms T and any term t, if $t \in ST(\overline{T})$ then either $t \in ST(T)$ or $t \in \overline{T}$.*

The proof can be found in the Appendix.

5 *Decidability*

In this section, we prove the main theorem of the paper.

Theorem 7 *The problem of checking whether* $Pr \models^T \varphi$ *for a given tagged protocol Pr, a given finite set* $T \subseteq \mathscr{B}$*, and a run formula* φ *is decidable.*

For the rest of the section, we fix a tagged protocol $Pr = \{\eta_1, \ldots, \eta_n\}$, a finite set $T \subseteq \mathscr{B}$, and a run formula φ_0. We first observe that the set of T-runs of Pr is infinite. So we need a way of reducing the given problem to that of checking the truth of φ_0 over a finite set of runs. Usually in modal logics, this is done by some kind of filtration argument, in which we define an equivalence relation of finite index on runs such that the logic cannot distinguish between two equivalent runs.

We do something similar here. Let us first observe that since we are considering T-runs for a fixed finite T, the major source of unboundedness is non-atomic substitutions. If not for them, the space of terms used in T-runs will be finite, and we get a decision procedure. Therefore, one of the ways of obtaining decidability is to prove that every run is equivalent to a well-typed run (in the sense that either both satisfy φ_0, or neither of them satisfies φ_0). In essence, every formula φ defines an equivalence relation on T-runs of finite index, namely the equivalence relation which equates two runs exactly when they satisfy the same subformulas of φ. Moreover, every equivalence class of this relation contains a canonical representative, which is a well-typed T-run. Thus the original problem can be reduced to the problem of checking whether all *well-typed* T-runs of Pr satisfy φ_0.

The reduction to well-typed runs brings us close to decidability, but it does not go all the way. Even though any well-typed T-run of Pr is of bounded length (the bound depending only on the sizes of Pr, T, and φ_0), the set of session identifiers used in well-typed T-events is not finite, and thereby the set of well-typed T-runs is infinite.

Fortunately, this is easily handled. Let us say that L is a bound on the length of well-typed T-runs. We observe that a run of length L cannot have more than L sessions, and further that we can rename each distinct session identifier occurring in a run with a distinct session identifier between 1 and L, while still obtaining a run which satisfies the same formulas as the original run. (This is because the only role played by the session identifiers is to distinguish two sessions, and a fixed finite pool of session identifiers suffices to play the same role.) The set of well-typed T-runs which use only these session identifiers is finite. Thus we can reduce the original problem to one of checking the truth of φ_0 for a finite set of runs. Moreover, this set of runs can be computed given Pr and T.

Thus we only have to look at the problem of checking the truth of a run formula over a given finite run. The nontrivial case to handle is that of the quantifiers. This can be presented as quantifier elimination, as detailed below. Let $\{\Pi_1, \ldots, \Pi_K\}$ be the set of all sessions that use the session identifiers between 1 and L. We fix the set of *new* session names $\{ses_1, \ldots, ses_K\}$, which are not used in φ_0. We now translate the given formula by replacing each existential quantifier by a disjunction over the above session names. We then prove that for every well-typed run ξ which

has only the above sessions, φ_0 is true in ξ under an assignment *assign* iff its translation is true in ξ under a related assignment *assign*′. We make these ideas precise below.

For a given run formula φ, the set of *session names* occurring in φ, denoted $sn(\varphi)$, is defined as follows, by induction:

- $sn(\alpha@ses) \stackrel{\text{def}}{=} \{ses\}$.
- $sn(m@ses = m'@ses') \stackrel{\text{def}}{=} \{ses, ses'\}$.
- $sn(\neg\varphi) \stackrel{\text{def}}{=} sn(\varphi)$.
- $sn(\varphi \vee \psi) \stackrel{\text{def}}{=} sn(\varphi) \cup sn(\psi)$.
- $sn((\exists ses)\varphi) \stackrel{\text{def}}{=} sn(\varphi) \cup \{ses\}$.

For abstract session names ses, ses_1, and ses_2, we define $ses[ses_1 := ses_2]$ as follows:

$$ses[ses_1 := ses_2] \stackrel{\text{def}}{=} \begin{cases} ses_2 & \text{if } ses = ses_1 \\ ses & \text{otherwise.} \end{cases}$$

Given a run formula φ, and two abstract session names ses_1 and ses_2, we define $\varphi[ses_1 := ses_2]$, the **replacement** of ses_1 by ses_2 in φ, as follows, by induction:

- $(\alpha@ses)[ses_1 := ses_2] \stackrel{\text{def}}{=} \alpha@(ses[ses_1 := ses_2])$
- $(m@ses = m'@ses')[ses_1 := ses_2] \stackrel{\text{def}}{=} m@ses_3 = m'@ses_4$, where $ses_3 = ses[ses_1 := ses_2]$ and $ses_4 = ses'[ses_1 := ses_2]$.
- $(\neg\psi)[ses_1 := ses_2] \stackrel{\text{def}}{=} \neg(\psi[ses_1 := ses_2])$
- $(\psi \vee \psi')[ses_1 := ses_2] \stackrel{\text{def}}{=} \psi[ses_1 := ses_2] \vee \psi'[ses_1 := ses_2]$
- $((\exists ses)\psi)[ses_1 := ses_2] \stackrel{\text{def}}{=} \begin{cases} (\exists ses)\psi & \text{if } ses = ses_1 \\ (\exists ses)(\psi[ses_1 := ses_2]) & \text{otherwise.} \end{cases}$

Let $S = \{ses_1, \ldots, ses_n\}$ be a finite set of abstract session names, and let φ be a run formula such that $sn(\varphi) \cap S = \varnothing$. We define $(\psi)^{\#}_S$ for all $\psi \in sf(\varphi)$ as follows, by induction:

- $(\alpha@ses)^{\#}_S \stackrel{\text{def}}{=} \alpha@ses$.
- $(m@ses = m'@ses')^{\#}_S \stackrel{\text{def}}{=} m@ses = m'@ses'$.
- $(\neg\psi)^{\#}_S \stackrel{\text{def}}{=} \neg(\psi^{\#}_S)$
- $(\psi \vee \psi')^{\#}_S \stackrel{\text{def}}{=} (\psi)^{\#}_S \vee (\psi')^{\#}_S$

- $((\exists ses)\psi)^{\#}_S \stackrel{\text{def}}{=} (\psi^{\#}_S)[ses := ses_1] \vee \cdots \vee (\psi^{\#}_S)[ses := ses_n]$

Lemma 8 *Suppose Pr is a protocol, and* $\{\Pi_1, \ldots, \Pi_n\}$ *is a finite set of sessions. Fix a set of abstract session names* $S = \{ses_1, \ldots, ses_n\}$. *Let* $assign' \stackrel{\text{def}}{=} assign[ses_1 := \Pi_1, \ldots, ses_n := \Pi_n]$.

For all runs ξ *such that* $Sessions(\xi) = \{\Pi_1, \ldots, \Pi_n\}$, *all run formulas* φ *such that* $sn(\varphi) \cap S = \emptyset$, *and all session assignments assign over* ξ *which are compatible with* φ,

$$\xi \vDash_{assign} \varphi \text{ iff } \xi \vDash_{assign'} \varphi^{\#}_S.$$

The proof of the above lemma is presented in the Appendix.

To continue with the proof of the main theorem, fix a particular well-typed run ξ of *Pr*. Let $Sessions(\xi) = \{\Pi_{i_1}, \ldots, \Pi_{i_n}\}$, where $1 \leq i_j \leq K$, for all $j : 1 \leq j \leq n$. Let $S = \{ses_{i_1}, \ldots, ses_{i_n}\}$. We let $\varphi_\xi \stackrel{\text{def}}{=} \varphi^{\#}_S$ and $assign_\xi \stackrel{\text{def}}{=} assign[ses_{i_1} := \Pi_{i_1}, \ldots, ses_{i_n} := \Pi_{i_n}]$. Note that φ_ξ and $assign_\xi$ can be effectively computed for any well-typed T-run ξ. Since $sn(\varphi) \cap S = \emptyset$, it follows from Lemma 8 that $\xi \vDash_{assign} \varphi$ iff $\xi \vDash_{assign_\xi} \varphi_\xi$.

This completes the proof of decidability, pending details of the reduction to well-typed runs.

Reduction to well-typed runs

We now outline the reduction of the original problem to that of checking whether every well-typed T-run of *Pr* satisfies φ_0. We do this by associating to every T-run ξ of *Pr* a so-called *normal run* ξ_{norm} (which is a well-typed T-run of *Pr*) such that φ_0 is true of ξ iff it is true of ξ_{norm}.

The most obvious manner in which we can associate a well-typed run to a given run is to replace each event of the given run by a corresponding well-typed event, which is got by using a corresponding well-typed substitution instead of the ill-typed substitution used in the original event. The "corresponding" well-typed substitution is got as follows: whenever the original substitution maps a nonce to a non-nonce, map it instead to a fixed nonce (n_0, say). More formally, fix a nonce n_0 not used in the specification of *Pr*, and assume without loss of generality that n_0 belongs to $initstate(Pr)$. For any substitution σ, we define σ_{wt} as follows:

$$\text{for all } x \in \mathscr{B} : \sigma_{wt}(x) \stackrel{\text{def}}{=} \begin{cases} \sigma(x) & \text{if } \sigma(x) \in \mathscr{B} \\ n_0 & \text{otherwise} \end{cases}$$

For any session $\Pi = (id, \eta, \sigma)$, we define Π_{wt} to be (id, η, σ_{wt}). For any event $e = (\Pi, lp)$, e_{wt} is defined to be (Π_{wt}, lp), and for any sequence of events $\xi = e_1 \cdots e_k$, ξ_{wt} is defined to be $(e_1)_{wt} \cdots (e_k)_{wt}$.

We could choose ξ_{wt} as our "normal run" ξ_{norm}, except for a technical hurdle. It needn't be a run, because it can happen that for two different events e and e' occurring in ξ, $e_{wt} = e'_{wt}$. Thus there would be repetition of events in ξ_{wt}. But this is easily handled. We can define ξ_{norm} to be the subsequence of ξ got by retaining only the earliest occurrence of each event, and we can show that ξ_{wt} and ξ_{norm} satisfy the same formulas. These details are given in the Appendix.

Meanwhile, we will show that ξ_{wt} is a *pseudo-run* of *Pr* (a sequence of events that satisfies all the conditions of runs except unique occurrence of events), and that it satisfies the same formulas as ξ.

Theorem 9 *Suppose $\xi = e_1 \cdots e_k$ is a run of Pr. Then ξ_{wt} is a pseudo-run of Pr.*

The proof of this is in fact the most non-trivial of all. The main thing is to check that all the events of ξ_{wt} are enabled at the points where they occur. This nontrivial part here is to check that whenever the intruder generates and sends a message t in the original run, the corresponding well-typed message can be generated by the intruder in the well-typed run. The point to remember is that the messages the intruder has learnt at a point in ξ_{wt} are the well-typed counterparts of the messages learnt at the same point in the original run. We prove this fact by an inductive construction of message derivations. *Buried deep in this proof is the assumption of tagging.* Basically it is used to prove that whenever a non-nonce is used instead of a nonce at some point, then in fact it has been generated by the intruder at that point, and it is not the case that this is an accidental unification with an instance of an unrelated message sent by some honest agent. Some of the steps of the inductive construction of message derivations crucially depends on this property. The details of the proof are found in Appendix.

The next lemma requires a preliminary definition:

Fix a protocol $Pr = \{\eta_1, \ldots, \eta_n\}$ and a run $\xi = e_1 \cdots e_k$ of *Pr*, with $e_i = (\Pi_i, lp_i)$ for every $i \leq k$. Consider $\xi_{wt} = (e_1)_{wt} \cdots (e_k)_{wt}$. Let us denote $\mathit{infstate}(e_1 \cdots e_i)$ by s_i, and $\mathit{infstate}((e_1)_{wt} \cdots (e_i)_{wt})$ by s'_i. It is clear that whenever Π is a session of ξ, Π_{wt} is a session of ξ_{wt}. Let assign_{wt} be the session assignment defined as follows:

$$\text{for all } \mathit{ses} \in \mathit{SesNames} : \mathit{assign}_{wt}(\mathit{ses}) = \Pi_{wt}, \text{ where } \mathit{assign}(\mathit{ses}) = \Pi.$$

Lemma 10

1. *For all session formulas α, all sessions $\Pi = (id, \eta, \sigma)$ of ξ such that $m@\Pi \in N$ for all formulas of the form A has $m \in sf(\alpha)$, and all $j : 1 \leq i \leq |\Pi|$: $\xi\restriction\Pi, j \vDash \alpha$ iff $\xi_{wt}\restriction\Pi_{wt}, j \vDash \alpha$.*
2. *For all run formulas φ, and all session assignments assign over ξ which are compatible with φ, $\xi \vDash_{assign} \varphi$ iff $\xi_{wt} \vDash_{assign_{wt}} \varphi$.*

Proof:

1. We prove the lemma by an induction on the structure of α. For ease of notation, we let Π' denote Π_{wt}, ζ denote $\xi\restriction\Pi$ and ζ' denote $\xi_{wt}\restriction\Pi_{wt}$. We assume that $\zeta = s_{i_1} e_{i_1} \cdots s_{i_r} e_{i_r} s_{i_r+1}$ and $\zeta' = s'_{i_1} e'_{i_1} \cdots s'_{i_r} e'_{i_r} s'_{i_r+1}$. *Note that events of ξ with session identifier id get mapped to events of ξ_{wt} with the same session identifier. So ζ' is of the form stated above.* We present the only nontrivial case. The rest of the cases follow directly either from the definitions or from the induction hypothesis.

- Suppose α is of the form A *has* m. Then $\zeta, j \models A$ *has* m iff $m@\Pi \in \overline{(s_{i_j})_{A@\Pi}}$. If we prove that this is equivalent to saying that $m@\Pi' \in \overline{(s'_{i_j})_{A@\Pi'}}$, we are through. This is so because the last statement is the same as saying that $\zeta', j \models A$ *has* m. **We postpone the proof of the fact that for all $i, j \leq k$, for all agents A, and for all $m \in N$ such that $\sigma_i(m) \in N$, $\sigma_i(m) \in \overline{(s_j)_A}$ iff $\sigma'_i(m) \in \overline{(s'_j)_A}$.**

2. We prove this by induction on run formulas. We only present some representative cases.

   - Suppose φ is of the form $\alpha@ses$. Let $assign(ses) = \Pi$. Clearly $assign_{wt}(ses) = \Pi_{wt}$. Then $\xi \models_{assign} \varphi$ iff $\xi\restriction\Pi, 1 \models \alpha$ iff $\xi_{wt}\restriction\Pi_{wt}, 1 \models \alpha$ iff $\xi_{wt} \models_{assign_{wt}} \varphi$.
   - Suppose φ is of the form $m@ses = m'@ses'$. Let $assign(ses) = \Pi$ and $assign(ses') = \Pi'$. Then $\xi \models_{assign} \varphi$ iff $m@\Pi = m'@\Pi'$ iff (using the fact that *assign* is compatible with φ) $m@\Pi_{wt} = m'@\Pi'_{wt}$ iff $\xi_{wt} \models_{assign_{wt}} \varphi$.
   - Suppose φ is of the form $(\exists ses)\psi$. We first observe that $(assign[ses := \Pi])_{wt} = assign_{wt}[ses := \Pi_{wt}]$. Then $\xi \models_{assign} \varphi$ iff there exists a session Π of ξ such that $\xi \models_{assign[ses:=\Pi]} \psi$ iff (by induction hypothesis) $\xi_{wt} \models_{assign_{wt}[ses:=\Pi_{wt}]} \psi$ iff $\xi_{wt} \models_{assign_{wt}} \varphi$. $\dashv$

We are left with proving the statement on which the base case of the above proof hinges. We split it into two statements, one for the honest agents and one for the intruder.

Lemma 11 *For all $i, j \leq k$, for all honest agents A, and for all $m \in N$ such that $\sigma_i(m) \in N$, $\sigma_i(m) \in \overline{(s_j)_A}$ iff $\sigma'_i(m) \in \overline{(s'_j)_A}$.*

The proof of this is straightforward, and we omit it. The message derivation system for the honest agents is simple to analyze. We are assuming that the honest agents receive typed terms which indicate the substitution, so they do not need to perform any nontrivial unification. Thus it can be easily seen that the behaviour of such honest agents is impervious to the actual substitution being used.

Lemma 12 *For all $i, j \leq k$ and for all $m \in N$ such that $\sigma_i(m) \in N$, $\sigma_i(m) \in \overline{T_j}$ iff $\sigma'_i(m) \in \overline{T'_j}$.*

The proof of this is a bit more involved than the above, and is given in the Appendix.

6 Discussion

We have presented a logic for describing properties of security protocols and shown that the verification problem for tagged protocols is decidable when only finitely many nonces are assumed. The expressiveness of the logic is limited but many interesting security properties can be expressed in this logic.

Many natural questions arise in the context of such a decidability result. Firstly, how limiting is the assumption about finite nonces and the restriction to tagged protocols? The former points to a limitation of the analysis presented here rather than any inherent problem, and we believe that the result in fact holds without the assumption of finite nonces, though the proof is elusive as

yet. The latter is more crucial since the techniques work for a subclass of security protocols. This raises the question of deciding whether a given protocol is "taggable" or not, while preserving a security property φ stated in the logic. In fact, it can be shown that for any executable protocol one can find an "equivalent" tagged protocol such that any honest run of the original protocol (one not involving intruder actions) is also a run of the tagged protocol, and further for any run of the tagged protocol which violates secrecy, there is a run of the original protocol which also violates secrecy. It is to be explored whether this generalises to other properties specifiable in the logic. Another related question is whether the logic contains formulas that *separate* tagged protocols from untagged ones. Clearly, it would be much nicer to use syntactic devices in the logic using which attention could be restricted to a subclass of protocols. These are interesting issues not answered here.

Some other questions that arise relate to the models themselves. Though we have presented a transition system model and used runs (paths) in the system, the analysis works with a notion akin to causal dependencies. For instance, two send actions by distinct agents can occur in any order and even the intruder cannot distinguish one ordering from another. Thus, formulas are "trace consistent", in the sense that they cannot distinguish between runs which differ only in reordering causally independent event occurrences. While we have not set up the formalism for proving such results, this can be done. What would be more interesting is an investigation of attacks that can be found in a 'truly concurrent' model but missed by an interleaving model.

The most important limitation of this exercise is the lack of a *deduction system*. Clearly abstract descriptions of security properties in themselves do not constitute a logic, and we need to present inference mechanisms which clarify how agents (in particular, the intruder) learn secrets. An axiomatization of a deductive notion of the form $Pr \vdash \varphi$ which is complete with respect to the semantic notion $\mathscr{M}(Pr) \models \varphi$ is very much needed.

We end with another observation on the general nature of protocol verification based on formal (Dolev-Yao) models. It is that the verification certificates are inherently probabilistic, because there is always the possibility of someone guessing keys. This aspect is hidden inside the model, when we assume unguessability of nonces and keys.

References

[ABKL93] Martin Abadi, Michael Burrows, Charles Kaufman, and Butler Lampson. Authentication and Delegation with Smart-Cards. *Science of Computer Programming*, 21(2):93–113, October 1993.

[ABV02] Rafael Accorsi, David Basin, and Luca Viganò. Modal Specifications of Trace-Based Security Properties. In Klaus Fischer and Dieter Hutter, editors, *Proceedings of the Second International Workshop on Security of Mobile Multiagent Systems*, pages 1–11, July 2002.

[ADM03] Kamel Adi, Mourad Debbabi, and Mohamed Mejri. A new logic for electronic commerce protocols. *Theoretical Computer Science*, 29(3):223–283, 2003.

[AN95] Ross Anderson and Roger M. Needham. Programming Satan's computer. In *Computer Science Today*, volume 1000 of *Lecture Notes in Computer Science*, pages 426–441, 1995.

[AT91] Martin Abadi and Mark Tuttle. A Semantics for a Logic of Authentication. In *Proceedings of the Tenth Annual ACM Symposium on Principles of Distributed Computing*, pages 201–216, August 1991.

[BAN90] Michael Burrows, Martin Abadi, and Roger M. Needham. A logic of authentication. *ACM Transactions on Computer Systems*, 8(1):18–36, Feb 1990.

[Bel03] Giampaolo Bella. Inductive Verification of Smartcard Protocols. *Journal of Computer Security*, 11(1):87–132, 2003.

[BP03] Bruno Blanchet and Andreas Podelski. Verification of Cryptographic Protocols: Tagging Enforces Termination. In Andrew D. Gordon, editor, *Proceedings of FoSSaCS'03*, volume 2620 of *Lecture Notes in Computer Science*, pages 136–152, 2003.

[CLS03] Hubert Comon-Lundh and Vitaly Shmatikov. Intruder Deductions, Constraint Solving and Insecurity Decisions in Presence of Exclusive or. In *Proceedings of the 18th IEEE Symposium on Logic in Computer Science (LICS)*, pages 271–280, June 2003.

[DDMP03] Anupam Datta, A. Derek, John C. Mitchell, and Dusko Pavlovic. Secure protocol composition. In *Proceedings of 19th Conference on Mathematical Foundations of Programming Semantics*, volume 83 of *Electronic Notes in Theoretical Computer Science*, 2003.

[DLMS99] Nancy A. Durgin, Patrick D. Lincoln, John C. Mitchell, and Andre Scedrov. The undecidability of bounded security protocols. In *Proceedings of the Workshop on Formal Methods and Security Protocols (FMSP'99)*, 1999.

[DMP03] Nancy A. Durgin, John C. Mitchell, and Dusko Pavlovic. A compositional logic for proving security properties of protocols. *Journal of Computer Security*, 11(4):677–721, 2003.

[DMTY97] Mourad Debbabi, Mohamed Mejri, Nadia Tawbi, and Imed Yahmadi. Formal automatic verification of authentication protocols. In *Proceedings of the First IEEE International Conference on Formal Engineering Methods (ICFEM97)*, pages 50–59. IEEE Press, 1997.

[DY83] Danny Dolev and Andrew Yao. On the Security of public-key protocols. *IEEE Transactions on Information Theory*, 29:198–208, 1983.

[FHG99] F. Javier Thayer Fábrega, Jonathan Herzog, and Joshua Guttman. Strand Spaces: Proving Security Protocols Correct. *Journal of Computer Security*, 7:191–230, 1999.

[GF01] Joshua Guttman and F. Javier Thayer Fábrega. Authentication tests and the structure of bundles. *Theoretical Computer Science*, 282(3):333–380, 2001.

[GLT89] Jean-Yves Girard, Yves Lafont, and Paul Taylor. *Proofs and Types*, volume 7 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 1989.

[GMP92] Janice Glasgow, Glenn MacEwen, and Prakash Panangaden. A logic for reasoning about security. *ACM Transactions on Computer Systems*, 10(3):226–264, Aug 1992.

[GNY90] Li Gong, Roger Needham, and Raphael Yahalom. Reasoning About Belief in Cryptographic Protocols. In Deborah Cooper and Teresa Lunt, editors, *Proceedings 1990 IEEE Symposium on Research in Security and Privacy*, pages 234–248. IEEE Computer Society, 1990.

[HLS00] James Heather, Gavin Lowe, and Steve Schneider. How to Prevent Type Flaw Attacks on Security Protocols. In *Proceedings of the 13th IEEE Computer Security Foundations Workshop (CSFW 13)*, pages 255–268, July 2000.

[HP03] Joseph Y. Halpern and Riccardo Pucella. Modeling adversaries in a logic for security protocol analysis. In *Formal Aspects of Security, First International Conference, FASec 2002*, volume 2629 of *Lecture Notes in Computer Science*, pages 115–132, 2003.

[HvdM03] Joseph Y. Halpern and Ron van der Meyden. A logical reconstruction of SPKI. *Journal of Computer Security*, 11(4):581–613, 2003.

[KW96] Darrell Kindred and Jeannette Wing. Fast, automatic checking of security protocols. In *Proceedings of the 2nd USENIX workshop on e-commerce*, pages 41–52, 1996.

[Low96] Gavin Lowe. Breaking and fixing the Needham-Schroeder public key protocol using FDR. In *Proceedings of TACAS'96*, volume 1055 of *Lecture Notes in Computer Science*, pages 147–166, 1996.

[Low99] Gavin Lowe. Towards a completeness result for model checking of security protocols. *Journal of Computer Security*, 7:89–146, 1999.

[MCJ97] Wilfredo R. Marrero, Edmund M. Clarke, and Somesh Jha. Model checking for security protocols. In *Proceedings of the DIMACS Workshop on design and verificaton of security protocols*, 1997.

[MS01] Jonathan K. Millen and Vitaly Shmatikov. Constraint solving for bounded-process cryptographic protocol analysis. In *ACM Conference on Computer and Communications Security*, pages 166–175, 2001.

[Nes90] D. M. Nessett. A critique of the Burrows, Abadi and Needham logic. *ACM Operating systems review*, 24(2):35–38, 1990.

[NS78] Roger M. Needham and Michael D. Schroeder. Using Encryption for Authentication in Large Networks of Computers. *Communications of the ACM*, 21(12):993–999, 1978.

[Pau98] Lawrence C. Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Computer Security*, 6:85–128, 1998.

[RS03a] R. Ramanujam and S. P. Suresh. A decidable subclass of unbounded security protocols. In Roberto Gorrieri, editor, *Proceedings of WITS'03*, pages 11–20, Poland, Warsaw, April 2003.

[RS03b] R. Ramanujam and S. P. Suresh. Tagging makes secrecy decidable for unbounded nonces as well. In *Proceedings of 23rd FST&TCS*, number 2914 in Lecture Notes in Computer Science, pages 363–374, Mumbai, India, December 2003.

[RS03c] R. Ramanujam and S. P. Suresh. Undecidability of secrecy for security protocols. Technical report, Matscience, July 2003.

[RS05a] R. Ramanujam and S. P. Suresh. Decidability of context-explicit security protocols. *Journal of Computer Security*, 13(1):135–165, 2005.

[RS05b] R. Ramanujam and S. P. Suresh. Deciding knowledge properties of security protocols. In *Proceedings of TARK X*, pages 219–235, Singapore, July 2005.

[RT03] Michaël Rusinowitch and Mathieu Turuani. Protocol Insecurity with Finite Number of Sessions and Composed Keys is NP-complete. *Theoretical Computer Science*, 299:451–475, 2003.

[Sur03] S.P. Suresh. *Foundations of Security Protocol Analysis*. PhD thesis, The Institute of Mathematical Sciences, Chennai, India, November 2003. Madras University. Available at `http://www.cmi.ac.in/~spsuresh`.

A Some facts about message derivations

In this section, we present some of the proofs for the statements in Section 4.

Proposition 13 *If there is a proof of $T \vdash_a t$ there is also a proof of $T \vdash_s t$.*

Proof: Suppose π is a proof of $T \vdash_a t$. Let $\{t_1, \ldots, t_N\}$ be the set of all subterms of t such that for all $i : 1 \leq i \leq N$, t_i is not a pair and t_i occurs unencrypted in t. Then it is clear that each of the t_i's is derivable from t using only a sequence of applications of the *split'* rule. Hence the following is a valid proof of $T \vdash_s t$.

$$\dfrac{\begin{array}{ccc} (\pi) & & (\pi) \\ \vdots & & \vdots \\ T \vdash_a t & & T \vdash_a t \\ \hline\hline T \vdash_a t_1 & & T \vdash_a t_N \\ \hline T \vdash_s t_1 & \cdots & T \vdash_s t_N \end{array}}{T \vdash_s t}\ \begin{array}{l} \text{sequence of splits} \\ \textit{promote} \\ \\ \text{sequence of splits} \\ \textit{promote} \\ \text{sequence of pairings} \end{array}$$

⊣

Proposition 14 *If there is a proof π of $T \vdash t$ then there is a proof π' of $T \vdash_s t$.*

Proof: Suppose π is a proof of $T \vdash t$. We prove by induction that for all subproofs ϖ[2] of π with root $T \vdash r$, there is a proof of $T \vdash_s r$. Further if ϖ ends with an *analz*-rule and is a minimal proof [3] of $T \vdash r$, then $T \vdash_a r$.

Consider any subproof ϖ of π with root labelled $T \vdash r$. Suppose it ends with a *synth*-rule, and suppose $T \vdash r_1$ and $T \vdash r_2$ are the premises of the last rule. Then $T \vdash_s r_1$ and $T \vdash_s r_2$ by induction hypothesis, whence $T \vdash_s r$ also follows.

Now suppose it ends with an *analz*-rule. We will assume the (strengthened) induction hypothesis for all proper subproofs of ϖ, and prove it for ϖ. We need to consider the following cases. In each case, we will show that $T \vdash_a r$. By Proposition 13, it follows that $T \vdash_s r$.

- Suppose ϖ is of the following form:

$$\dfrac{}{T \vdash r}\ Ax$$

  Then it is clear that $r \in T$ and therefore the following is a proof of $T \vdash_a r$.

$$\dfrac{}{T \vdash_a r}\ Ax'$$

- Suppose ϖ is of the following form:

[2] This is a variant of π, and not of ω. We can read it as *varpi*.

[3] A minimal proof of $T \vdash r$ is a proof of $T \vdash r$, none of whose proper subproofs is a proof of $T \vdash r$.

$$\dfrac{\begin{array}{c}(\varpi_1)\\ \vdots \\ T \vdash (r, r')\end{array}}{T \vdash r}\, split_1$$

If $T \vdash r$ occurs in ϖ_1, then of course $T \vdash_s r$ is derivable, by induction hypothesis. Further, if there is a minimal proof of $T \vdash r$ that occurs as a subproof of ϖ_1 and ends with an *analz*-rule, then $T \vdash_a r$ as well, again by induction hypothesis.

If $T \vdash r$ does not occur in ϖ_1 then consider a minimal proof ϖ_2 of $T \vdash (r, r')$ occurring as a subproof of ϖ_1. Suppose ϖ_2 ends with a *synth*-rule. This means that the premises of the last rule in ϖ_2 are $T \vdash r$ and $T \vdash r'$. But this is a contradiction, since we are considering the case when $T \vdash r$ doesn't occur in ϖ_1. Therefore ϖ_2 ends with an *analz*-rule, and hence by induction hypothesis there is a proof ϖ_2' of $T \vdash_a (r, r')$. Now the following is a proof of $T \vdash_a r$.

$$\dfrac{\begin{array}{c}(\varpi_2')\\ \vdots \\ T \vdash_a (r, r')\end{array}}{T \vdash_a r}\, split_1'$$

- Suppose ϖ is of the following form:

$$\dfrac{\begin{array}{cc}(\varpi_1) & (\varpi_2)\\ \vdots & \vdots \\ T \vdash \{r\}_{r'} & T \vdash inv(r')\end{array}}{T \vdash r}\, decrypt$$

If $T \vdash r$ occurs in ϖ_1, then of course $T \vdash_s r$ is derivable, by induction hypothesis. Further, if there is a minimal proof of $T \vdash r$ that occurs as a subproof of ϖ_1 and ends with an *analz*-rule, then $T \vdash_a r$ as well, again by induction hypothesis.

If $T \vdash r$ does not occur in ϖ_1 then consider a minimal proof ϖ_3 of $T \vdash \{r\}_{r'}$ occurring as a subproof of ϖ_1. Suppose ϖ_3 ends with a *synth*-rule. This means that the premises of the last rule in ϖ_3 are $T \vdash r$ and $T \vdash r'$. But this is a contradiction, since we are considering the case when $T \vdash r$ doesn't occur in ϖ_1. Therefore ϖ_3 ends with an *analz*-rule, and hence by induction hypothesis there is a proof ϖ_3' of $T \vdash_a \{r\}_{r'}$.

Let us now consider ϖ_2. If it ends with a *synth*-rule, then we have already proved that $T \vdash_s inv(r')$. If it ends with an *analz*-rule, then $T \vdash_a inv(r')$ by induction hypothesis. By Proposition 13 there is a proof ϖ_2' of $T \vdash_s inv(r')$. Now the following is a proof of $T \vdash_a r$.

$$\dfrac{\begin{array}{cc}(\varpi_3') & (\varpi_2')\\ \vdots & \vdots \\ T \vdash_a \{r\}_{r'} & T \vdash_s inv(r')\end{array}}{T \vdash_a r}\, decrypt'$$

⊣

Proposition 15 *For any set of terms T and any term t, if $t \in ST(\overline{T})$ then either $t \in ST(T)$ or $t \in \overline{T}$.*

Proof: For any term of the form (r, r') or $\{r\}_{r'}$, we say that r and r' are its *immediate subterms*. We now observe that $t \in ST(\overline{T})$ iff there exists a sequence of terms $t_1, \ldots, t_n$ such that $t_1 \in \overline{T}$, $t_n = t$ and for all $i : 1 \leq i < n$, t_{i+1} is an immediate subterm of t_i.

We now prove by induction that $t_i \in ST(T) \cup \overline{T}$ for all $i \leq n$. The base case is trivial, since $t_1 \in \overline{T}$. The induction case hinges on the following observation.

$$\text{If } (t, t') \in ST(T) \cup \overline{T} \text{ or } \{t\}_{t'} \in ST(T) \cup \overline{T}, \text{ then } \{t, t'\} \subseteq ST(T) \cup \overline{T}.$$

To prove this, we first note that if $(t, t') \in ST(T)$ or $\{t\}_{t'} \in ST(T)$ then $\{t, t'\} \subseteq ST(T)$. If $(t, t') \in \overline{T}$ then clearly $\{t, t'\} \subseteq \overline{T}$. If $\{t\}_{t'} \in \overline{T}$, then consider a minimal proof π of $T \vdash \{t\}_{t'}$. Either π ends with an *analz*-rule, in which case $T \vdash_a \{t\}_{t'}$ is derivable, whence it follows from Proposition 4 that $\{t\}_{t'} \in ST(T)$ (and hence $\{t, t'\} \subseteq ST(T)$); or it ends with a *synth*-rule, in which case t and t' occur as premises of the last rule of π, whence $\{t, t'\} \subseteq \overline{T}$. ⊣

B Proof of quantifier elimination

In this section, we prove the lemma on quantifier elimination. To prove it, we need the following useful relationship between replacements in formulas, and updates of assignments, whose proof is a straightforward induction using the definitions.

Proposition 16 *For all runs ξ of a protocol Pr, all run formulas φ, all assignments assign over ξ compatible with φ, and any two abstract session names ses_1 and ses_2, we have the following.*

$$\xi \vDash_{assign} \varphi[ses_1 := ses_2] \quad \textit{iff} \quad \xi \vDash_{assign[ses_1 := assign(ses_2)]} \varphi.$$

In words, what the above proposition says is that ξ satisfies a replacement of the formula φ under an assignment iff it satisfies φ under the corresponding updated assignment.

Lemma 17 *Suppose Pr is a protocol, and $\{\Pi_1, \ldots, \Pi_n\}$ is a finite set of sessions. Fix a set of abstract session names $S = \{ses_1, \ldots, ses_n\}$. Let $assign' \stackrel{\text{def}}{=} assign[ses_1 := \Pi_1, \ldots, ses_n := \Pi_n]$.*

For all runs ξ such that $Sessions(\xi) = \{\Pi_1, \ldots, \Pi_n\}$, all run formulas φ such that $sn(\varphi) \cap S = \emptyset$, and all session assignments assign over ξ which are compatible with φ,

$$\xi \vDash_{assign} \varphi \textit{ iff } \xi \vDash_{assign'} \varphi^{\#}_{S}.$$

Proof: We prove the claim by induction for all subformulas ψ of φ. We present the more interesting cases of the induction.

- Suppose ψ is of the form $\alpha@ses$. Since $ses \in sn(\varphi)$, $ses \notin S$, and so $assign_{ses} = assign'_{ses}$ and $\psi^{\#}_S = \psi$. Thus, $\xi \vDash_{assign} \psi$ iff $\xi \vDash_{assign'} \psi^{\#}_S$.

- Suppose ψ is of the form $m@ses = m'@ses'$. Since $ses, ses' \in sn(\varphi)$, $ses \notin S$ and $ses' \notin S$. Hence $assign_{ses} = assign'_{ses}$, $assign_{ses'} = assign'_{ses'}$, and $\psi^{\#}_{SesNames} = \psi$. Thus $\xi \vDash_{assign} \psi$ iff $\xi \vDash_{assign'} \psi^{\#}_S$.

- Suppose ψ is of the form $(\exists ses)\psi'$. Let $assign^i$ denote $(assign[ses := \Pi_i])[ses_1 := \Pi_1, \ldots, ses_n := \Pi_n]$.

  Suppose $\xi \vDash_{assign} \psi$. Then there exists a session Π_i of ξ such that $\xi \vDash_{assign[ses:=\Pi_i]} \psi'$. But then by induction hypothesis $\xi \vDash_{assign^i} (\psi')^{\#}_S$. Since $ses \in sn(\varphi)$, $ses \notin S$, and hence $assign^i = assign'[ses := \Pi_i]$. Thus, $\xi \vDash_{assign'[ses:=\Pi_i]} (\psi')^{\#}_S$. This implies, by Proposition 16, that $\xi \vDash_{assign'} (\psi')^{\#}_S[ses := ses_i]$. This immediately implies that $\xi \vDash_{assign'} \psi^{\#}_S$.

  Suppose, on the other hand, that $\xi \vDash_{assign'} \psi^{\#}_S$. This means that for some $i : 1 \leq i \leq n$, $\xi \vDash_{assign'} (\psi')^{\#}_S[ses := ses_i]$. But then, by Proposition 16, this implies that $\xi \vDash_{assign'[ses:=\Pi_i]} (\psi')^{\#}_S$. As we have already observed, $assign^i = assign'[ses := \Pi_i]$, and hence $\xi \vDash_{assign^i} (\psi')^{\#}_S$. By induction hypothesis, we now have $\xi \vDash_{assign[ses:=\Pi_i]} \psi'$, and this implies that $\xi \vDash_{assign} \psi$. ⊣

C *Reduction to well-typed runs*

We restate Theorem 9 below, and prove it in the rest of the section.

Theorem 18 *Suppose $\xi = e_1 \cdots e_k$ is a run of Pr. Then ξ_{wt} is a pseudo-run of Pr.*

Let $\xi = e_1 \cdots e_k$ be a run of *Pr*, with $e_i = (\Pi_i, lp_i)$ and $\Pi_i = (id_i, \eta_i, \sigma_i)$ for each $i \leq k$. For notational ease, we let σ'_i denote $(\sigma_i)_{wt}$, Π'_i denote $(\Pi_i)_{wt}$, e'_i denote $(e_i)_{wt}$, and ξ' denote ξ_{wt}. It is easy to see that the unique origination property holds in ξ'. So the only nontrivial thing is to show that for all $i \leq k$ such that $act(e'_i)$ is a receive action, $act(e'_i)$ is enabled at $infstate(e'_1 \cdots e'_{i-1})$.

For ease of notation, we use the following: For $i : 0 \leq i \leq k$, we let g_i denote $term(\eta_i(lp_i))$, t_i denote $\sigma_i(g_i)$, and t'_i denote $\sigma'_i(g_i)$. T_0 denotes $(initstate)_I$, G_i denotes $T_0 \cup \{g_1, \ldots, g_i\}$, T_i denote $T_0 \cup \{t_1, \ldots, t_i\}$, and T'_i denote $T_0 \cup \{t'_1, \ldots, t'_i\}$.

We first define the notion of *origination* of a term, in the context of a run. We say that a term *originates* at $i \leq k$ in ξ iff $t \in ST(T_i) \setminus ST(T_{i-1})$.

Lemma 19 *Suppose e_i is a send event and there exists $x \in ST(g_i) \cap \mathscr{B}$ such that $\sigma_i(x)$ is not in $\mathscr{B}$. Then $i > 1$ and there exists $j \leq i$ such that $x \in ST(g_j) \cap \mathscr{B}$ and $\sigma_i(x) = \sigma_j(x)$.*

Proof: Since $\sigma(x) \notin \mathscr{B}$, we conclude from the definition of runs that x does not originate at lp_i in η_i. But this means that there is $lp < lp_i$ such that $x \in ST(\eta_i(lp))$. Thus there is an event $j < i$ such that $(\Pi_j, lp_j) = (\Pi_i, lp)$. Clearly $\sigma_i = \sigma_j$, and hence $\sigma_i(x) = \sigma_j(x)$. Of course $x \in ST(g_j) \cap \mathscr{B}$. ⊣

Lemma 20 *Suppose a term t originates at i and e_i is a receive. Then $t \in \overline{T_{i-1}}$, and further, if $t = \{u\}_{u'}$ then $\{u, u'\} \subseteq \overline{T_{i-1}}$.*

Proof: Since e_i is a receive event it is clear that $t_i \in \overline{T_{i-1}}$. But $t \in ST(t_i)$, so $t \in ST(\overline{T_{i-1}})$. But then $t \in ST(T_{i-1}) \cup \overline{T_{i-1}}$, from Proposition 6. But t originates at i and so $t \notin ST(T_{i-1})$, and therefore $t \in \overline{T_{i-1}}$. Further no proof of $T_{i-1} \vdash t$ can end with an *analz*-rule, since t is not a subterm of T_{i-1}. This means that if $t = \{u\}_{u'}$, it has to be constructed using a *synth*-rule at the end, which implies that both u and u' belong to $\overline{T_{i-1}}$ as well. ⊣

Lemma 21 *If $\sigma_i(x)$ is not in $\mathscr{B}$ for some $i \leq k$ and $x \in ST(g_i) \cap \mathscr{B}$, then $\sigma_i(x) \in \overline{T_{i-1}}$.*

Proof: We prove this by induction on i.

The base case is when i is 1. Suppose there exists $x \in ST(g_1) \cap \mathscr{B}$ such that $\sigma_1(x)$ is not in $\mathscr{B}$. It follows from Lemma 19 that e_1 is not a send. Thus it is a receive and $\sigma_1(x)$ originates at 1 (since T_0 consists entirely of names). Hence by Lemma 20 it follows that $\sigma_1(x) \in \overline{T_0}$.

Let us now consider the case when i is greater than 1. There are two cases to be considered:

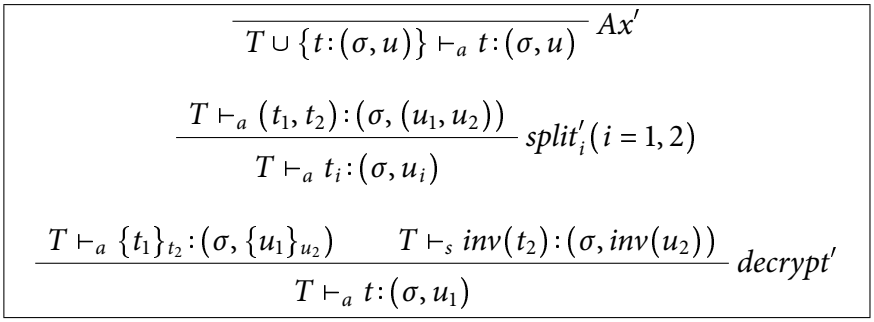
t_i is a send message: In this case it easily follows from Lemma 19 that there is a $j < i$ such that $x \in ST(g_j) \cap \mathscr{B}$ and $\sigma_i(x) = \sigma_j(x)$. By induction hypothesis $\sigma_j(x) \in \overline{T_{j-1}}$. From this it follows that $\sigma_i(x) \in \overline{T_{i-1}}$.

t_i is a receive message: Let us first note that $t_i \in \overline{T_{i-1}}$. If x occurs unencrypted in g_i then $\sigma_i(x) \in \overline{T_{i-1}}$. (Here we make crucial use of the fact that ill-typed substitutions are disallowed in "key positions". If we allowed them, then one can concoct examples which falsify the above statement.) If not, then let $\{u\}_{u'}$ be a minimal encrypted subterm of g_i which contains x as a subterm. If $\sigma_i(\{u\}_{u'})$ originates at i then it is immediate from Lemma 20 that $\sigma_i(u)$, and hence $\sigma_i(x)$ (since x occurs unencrypted in u), is in $\overline{T_{i-1}}$.

If $\sigma_i(\{u\}_{u'})$ does not originate at i, let it originate at $j < i$. There are again two cases to consider:

- **t_j is a receive message:** By Lemma 20 it immediately follows that $\sigma_i(\{u\}_{u'})$, as well as $\sigma_i(u)$, are in $\overline{T_{j-1}}$. But x occurs unencrypted in u and hence $\sigma_i(x)$ belongs to $\overline{T_{j-1}}$ as well. From this it is immediate that $\sigma_i(x)$ belongs to $\overline{T_{i-1}}$.
- **t_j is a send message:** Now it cannot be the case that $\sigma_i(\{u\}_{u'}) \in ST(\sigma_j(y))$ for some $y \in ST(g_j) \cap \mathscr{B}$, since then the nonatomic term $\sigma_j(y)$ would be originating at the send event e_j, contradicting Lemma 19. Therefore there exists $\{w\}_{w'} \in ST(g_j)$ such that $\sigma_i(\{u\}_{u'}) = \sigma_j(\{w\}_{w'})$. It then follows that $\{u\}_{u'} = \{w\}_{w'}$, by Proposition 1, *which is an immediate consequence of the tagging scheme we use.* But then $\sigma_i(x)$ is the same as $\sigma_j(x)$, which belongs to $\overline{T_{j-1}}$ by induction hypothesis. From this it follows that $\sigma_i(x) \in \overline{T_{i-1}}$.

⊣



$$\frac{}{T \cup \{t:(\sigma,u)\} \vdash_a t:(\sigma,u)}\ Ax'$$

$$\frac{T \vdash_a (t_1,t_2):(\sigma,(u_1,u_2))}{T \vdash_a t_i:(\sigma,u_i)}\ split'_i\,(i=1,2)$$

$$\frac{T \vdash_a \{t_1\}_{t_2}:(\sigma,\{u_1\}_{u_2}) \qquad T \vdash_s inv(t_2):(\sigma,inv(u_2))}{T \vdash_a t:(\sigma,u_1)}\ decrypt'$$

Figure 4: Typed *analz*-rules

We are on our way to proving that for all $i \leq k$: if e'_i is a receive event then $t'_i \in \overline{T'_{i-1}}$. The key property to be proved is that if $\sigma_i(t) \in \overline{T_j}$ for some $t \in ST(Pr)$, then $\sigma'_i(t) \in \overline{T'_j}$. This is quite tricky because of the nontrivial information transfer between sessions of the protocol. For instance, suppose there are two terms $u_1 = (\{x\}_y, p)$ and $u_2 = z$ in $ST(Pr)$, and two substitutions σ_1 and σ_2 suitable for Pr such that $\sigma_1(u_1) = \sigma_2(u_2)$. Clearly we need to replace σ_2 with a well-typed substitution σ'_2. But notice that while $\{\sigma_2(u_2)\} \vdash \sigma_1(u_1)$, it might not be the case that $\{\sigma'_2(u_2)\} \vdash \sigma'_1(u_1)$. Thus we need to make a finer analysis of the intruder derivations. In particular, we need to consider the "type" of each term, and replace it by n_0 only when it is nonatomic and it is typed as an atomic term. The rest of the section introduces a system of typed derivations, and uses it to prove the desired result.

Typed proofs for the intruder

We introduced typed terms and sequents in the context of deduction rules for the honest agents. While that is part of the model, what we are about to present is a device to prove decidability. We introduce the notion of typed proofs for the intruder, and prove that *for tagged protocols* any message that can be derived by the intruder at all can be derived using a typed proof (even though typed proofs are more restrictive in general). The use of typed proofs simplifies the proof that $\sigma'_i(t) \in \overline{T'_j}$ whenever $\sigma_i(t) \in \overline{T_j}$. A *typed term* is of the form $t:(\sigma,u)$ where t is any term, σ is a substitution, and u is a term such that $t = \sigma(u)$. A typed term $t:(\sigma,u)$ is said to be *top-level matching* if t is a nonce iff u is. A *typed sequent* is either of the form $T \vdash_a t:(\sigma,u)$ or of the form $T \vdash_s t:(\sigma,u)$, where $T \cup \{t:(\sigma,u)\}$ is a set of typed terms. We introduce a deduction system for deriving typed sequents, in Figure 4 and Figure 5.

For ease of presentation we introduce the following notation. We let σ_{id} denote the identity substitution, and we define P_i as follows:

$$P_i \stackrel{\text{def}}{=} \{t:(\sigma_{id},t) \mid t \in T_0\} \cup \{t_j:(\sigma_j,g_j) \mid 1 \leq j \leq i\}.$$

The next step is to show that for tagged protocols, the deduction system of Figure 4 and Figure 5 is as powerful as the original deduction system of untyped proofs.

Lemma 22 *Consider any T_i and any term $t \in \overline{T_i} \setminus \overline{T_{i-1}}$.*

$$\frac{T \vdash_a t : (\sigma_1, u_1) \quad t \text{ is not a pair}}{T \vdash_s t : (\sigma_2, u_2)}\ \textit{unify}$$

$$\frac{T \vdash_s t : (\sigma_1, u_1) \quad u_1 \text{ is not a nonce} \quad n \text{ is a nonce}}{T \vdash_s t : (\sigma_2, n)}\ \textit{simplify}$$

$$\frac{T \vdash_s t_1 : (\sigma, u_1) \qquad T \vdash_s t_2 : (\sigma, u_2)}{T \vdash_s (t_1, t_2) : (\sigma, (u_1, u_2))}\ \textit{pair}'$$

$$\frac{T \vdash_s t_1 : (\sigma, u_1) \qquad T \vdash_s t_2 : (\sigma, u_2)}{T \vdash_s \{t_1\}_{t_2} : (\sigma, \{u_1\}_{u_2})}\ \textit{encrypt}'$$

Figure 5: Typed *synth*-rules

If $T_i \vdash_a t$, then there is $u \in ST(G_i)$ and $i' \leq i$ such that $t : (\sigma_{i'}, u)$ is top-level matching and $P_i \vdash_a t : (\sigma_{i'}, u)$.

Further, whenever $T_i \vdash_s t$, it is also the case that $P_i \vdash_s t : (\tau, w)$ for all τ and w such that $t = \tau(w)$.

Proof: Fix a T_i and a term $t \in \overline{T_i} \setminus \overline{T_{i-1}}$.

Consider a proof π of $T_i \vdash_s t$ such that all subproofs ϖ of π satisfy the following property: If the root of ϖ is labelled $T_i \vdash_a r$ then ϖ contains leaves with labels only from T_j (where j is least such that $T_j \vdash_a r$ is derivable), and if the root of ϖ is labelled $T_i \vdash_s r$ then ϖ contains leaves with labels only from T_j (where j is least such that $T_j \vdash_s r$ is derivable).

We prove the claim for all subproofs ϖ of π, thus proving it for π as well.

- Suppose ϖ is the following proof:

$$\frac{}{T_i \vdash_a t}\ Ax'$$

  Then it is clear that $t \in T_i$ and $t \notin T_j$ for all $j < i$. If $i = 0$ we can choose u to be t itself, otherwise we can choose u to be g_i. It is clear that $t : (\sigma_i, u)$ is top-level matching. The following is a proof of $P_i \vdash_a t : (\sigma_i, u)$.

$$\frac{}{P_i \vdash_a t : (\sigma_i, u)}\ Ax'$$

- Suppose ϖ is the following proof:

$$\frac{\begin{array}{c}(\varpi_1)\\ \vdots \\ T_i \vdash_a (t, t')\end{array}}{T_i \vdash_a t}\ \textit{split}'_1$$

It is clear that (t, t') is not in $\overline{T_j}$ for any $j < i$. Thus by induction hypothesis there is a term $(u, u') \in ST(G_i)$ and $i' \leq i$ such that there is a proof ϖ_1' of $P_i \vdash_a (t, t') : (\sigma_{i'}, (u, u'))$. Now if u is a nonce and t is not, then by Lemma 21 it follows that $t = \sigma_{i'}(u) \in \overline{T_{i-1}}$, which contradicts our assumption about t. Thus $t : (\sigma_i, u)$ is top-level matching. The following is a proof of $P_i \vdash_a t : (\sigma_{i'}, u)$.

$$\dfrac{\begin{array}{c}(\varpi_1')\\ \vdots \\ P_i \vdash_a (t, t') : (\sigma_{i'}, (u, u'))\end{array}}{P_i \vdash_a t : (\sigma_{i'}, u)}\ split_1'$$

- Suppose ϖ is the following proof:

$$\dfrac{\begin{array}{cc}(\varpi_1) & (\varpi_2)\\ \vdots & \vdots \\ T_i \vdash_a \{t\}_{t'} & T_i \vdash_s inv(t')\end{array}}{T_i \vdash_a t}\ decrypt'$$

Suppose j is least such that $\{t\}_{t'} \in \overline{T_j}$. If $j < i$ then, since $t \notin \overline{T_j}$, no proof of $T_j \vdash \{t\}_{t'}$ ends with a *synth*-rule (since then t would occur in a premise of the last rule), and hence ends with an *analz*-rule. Thus in fact j is least such that $T_j \vdash_a \{t\}_{t'}$ is derivable. By our assumption on subproofs of π, this means that ϖ_1 is in effect a proof of $T_j \vdash_a \{t\}_{t'}$. If $j = i$, then also ϖ_1 is a proof of $T_j \vdash_a \{t\}_{t'}$.

Thus by induction hypothesis there is $\{u\}_{u'} \in ST(G_j)$, $j' \leq j$ and a proof of $P_j \vdash_a \{t\}_{t'} : (\sigma_{j'}, \{u\}_{u'})$. Since $P_j \subseteq P_i$, it follows that there is a proof ϖ_1' of $P_i \vdash_a \{t\}_{t'} : (\sigma_{j'}, \{u\}_{u'})$. Since $\sigma_{j'}(u') = t'$, $\sigma_{j'}(inv(u')) = inv(t')$. Thus, again by induction hypothesis, there is a proof ϖ_2' of $P_i \vdash_s inv(t') : (\sigma_{j'}, inv(u'))$. Suppose now that u is a nonce and t is not. Then by Lemma 21 it follows that $t = \sigma_{j'}(u) \in \overline{T_{j'-1}}$, which contradicts our assumption about t. Thus $t : (\sigma_{j'}, u)$ is top-level matching. The following is a proof of $P_i \vdash_a t : (\sigma_{j'}, u)$.

$$\dfrac{\begin{array}{cc}(\varpi_1') & (\varpi_2')\\ \vdots & \vdots \\ P_i \vdash_a \{t\}_{t'} : (\sigma_{j'}, \{u\}_{u'}) & P_i \vdash_s inv(t') : (\sigma_{j'}, inv(u'))\end{array}}{P_i \vdash_a t : (\sigma_{j'}, u)}\ decrypt'$$

- Suppose ϖ is of the following form:

$$\dfrac{\begin{array}{c}(\varpi_1)\\ \vdots \\ T_i \vdash_a t\end{array}}{T_i \vdash_s t}\ promote$$

By induction hypothesis there is a proof ϖ_1' of $P_i \vdash_a t : (\sigma_j, u)$ for some $u \in ST(G_i)$ and $j \leq i$ such that $t : (\sigma_j, u)$ is top-level matching. Since t occurs as a premise of the *promote* rule, it is not a pair. Hence it can be freely used as a premise of the *unify* rule. Hence we have the following proof of $P_i \vdash_s t : (\sigma, w)$ for any w and σ such that $t = \sigma(w)$.

$$\dfrac{\begin{array}{c}(\varpi_1')\\ \vdots \\ P_i \vdash_a t : (\sigma_j, u)\end{array}}{P_i \vdash_s t : (\sigma, w)}\ \mathit{unify}$$

- Suppose $t = (t_1, t_2)$ and ϖ is the following proof:

$$\dfrac{\begin{array}{cc}(\varpi_1) & (\varpi_2)\\ \vdots & \vdots \\ T_i \vdash_s t_1 & T_i \vdash_s t_2\end{array}}{T_i \vdash_s (t_1, t_2)}\ \mathit{pair}'$$

Consider any u and σ such that $\sigma(u) = (t_1, t_2)$. If u is a nonce then we note that t_1 and t_2 are instances of themselves, and therefore by induction hypothesis there are proofs ϖ_1' and ϖ_2' of $P_i \vdash_s t_1 : (\sigma_{id}, t_1)$ and $P_i \vdash_s t_2 : (\sigma_{id}, t_2)$ respectively, where σ_{id} is the identity substitution. So we have the following proof of $P_i \vdash_s t : (\sigma, u)$.

$$\dfrac{\dfrac{\begin{array}{cc}(\varpi_1') & (\varpi_2')\\ \vdots & \vdots \\ P_i \vdash_s t_1 : (\sigma_{id}, t_1) & P_i \vdash_s t_2 : (\sigma_{id}, t_2)\end{array}}{P_i \vdash_s (t_1, t_2) : (\sigma_{id}, (t_1, t_2))}\ \mathit{pair}'}{P_i \vdash_s (t_1, t_2) : (\sigma, u)}\ \mathit{simplify}$$

If, on the other hand, u is of the form (u_1, u_2) then clearly $\sigma(u_1) = t_1$ and $\sigma(u_2) = t_2$. Thus by induction hypothesis there are proofs ϖ_1' and ϖ_2' of $P_i \vdash_s t_1 : (\sigma, u_1)$ and $P_i \vdash_s t_2 : (\sigma, u_2)$, respectively. So we have the following proof of $P_i \vdash_s t : (\sigma, u)$.

$$\dfrac{\begin{array}{cc}(\varpi_1') & (\varpi_2')\\ \vdots & \vdots \\ P_i \vdash_s t_1 : (\sigma, u_1) & P_i \vdash_s t_2 : (\sigma, u_2)\end{array}}{P_i \vdash_s (t_1, t_2) : (\sigma, (u_1, u_2))}\ \mathit{pair}'$$

- Suppose $t = \{t_1\}_{t_2}$ and ϖ is the following proof:

$$\dfrac{\begin{array}{cc}(\varpi_1) & (\varpi_2)\\ \vdots & \vdots \\ T_i \vdash_s t_1 & T_i \vdash_s t_2\end{array}}{T_i \vdash_s \{t_1\}_{t_2}}\ \mathit{encrypt}'$$

Then we proceed exactly as in the above case. ⊣

Lemma 23 *For every term $t \in ST(T_k)$, every $u \in ST(G_k)$, and every $i, j \leq k$, if $P_i \vdash_a t:(\sigma_j, u)$ or $P_i \vdash_s t:(\sigma_j, u)$, then $\sigma'_j(u) \in \overline{T'_i}$.*

Proof: We do an induction on the structure of proofs. Suppose π is a proof of either $P_i \vdash_a t:(\sigma_j, u)$ or $P_i \vdash_s t:(\sigma_j, u)$. We will assume that for every sequent $P_i \vdash_a r:(\sigma, w)$ occurring in π, $r:(\sigma, w)$ is top-level matching. (This can always be achieved without loss of generality.) Then there are the following cases to consider:

- Suppose π is the following proof:

$$\frac{}{P_i \vdash_a t:(\sigma_j, u)}\, Ax'$$

  Either $t \in T_0$, in which case $\sigma'_j(t) = t \in T_0 \subseteq T'_i$; or $j \leq i$ and $u = t_j$, in which case we observe that $\sigma'_j(u) \in T'_i$.

- Suppose π is the following proof:

$$\frac{\begin{array}{c}(\pi_1)\\ \vdots \\ P_i \vdash_a (t, t'):(\sigma_j, (u, u'))\end{array}}{P_i \vdash_a t:(\sigma_j, u)}\, split'_1$$

  By induction hypothesis $\sigma'_j((u, u')) \in \overline{T'_i}$. But $\sigma'_j((u, u')) = (\sigma'_j(u), \sigma'_j(u'))$, and hence it easily follows that $\sigma'_j(u) \in \overline{T'_i}$.

- Suppose π is the following proof:

$$\frac{\begin{array}{cc}(\pi_1) & (\pi_2)\\ \vdots & \vdots \\ P_i \vdash_a \{t\}_{t'}:(\sigma_j, \{u\}_{u'}) & P_i \vdash_s inv(t'):(\sigma_j, inv(u'))\end{array}}{P_i \vdash_a t:(\sigma_j, u)}\, decrypt'$$

  By induction hypothesis $\{\sigma'_j(u)\}_{\sigma'_j(u')} = \sigma'_j(\{u\}_{u'}) \in \overline{T'_i}$. If $u' = inv(u')$, then $inv(u') \in ST(G_k)$. Thus by induction hypothesis, $\sigma'_j(inv(u')) \in \overline{T'_i}$. But $inv(\sigma'_j(u'))$ is the same as $\sigma'_j(inv(u'))$ and hence $inv(\sigma'_j(u')) \in \overline{T'_i}$.

  On the other hand, if $u' \neq inv(u')$, u' is of the form $public(A)$ or $private(A)$ for some $A \in Ag$. Without loss of generality, let $u' = public(A)$. Then $inv(t') = private(B)$ for some B. It has to be the case that π_2 ends with an application of the *unify* rule. Thus there is a subproof π_3 of π_2 which proves $P_i \vdash_a inv(t'):(\sigma_{j'}, u'')$. But it follows from our assumption

about subproofs of π that $inv(t') : (\sigma_{j'}, u'')$ is top-level matching. This means that $u'' = private(C)$ for some C, and hence $\sigma'_{j'}(u'') = \sigma_{j'}(u'') = private(B)$. By induction hypothesis $private(B) \in \overline{T'_i}$. Note that $\sigma'_j(u') = \sigma_j(u') = public(B)$. Thus $\{\sigma'_j(u)\}_{public(B)} \in \overline{T'_i}$. It immediately follows that $\sigma'_j(u) \in \overline{T'_i}$.

- Suppose π is the following proof:

$$\dfrac{\begin{array}{c}(\pi_1)\\ \vdots \\ P_i \vdash_a t : (\sigma_{j'}, u') \qquad t \text{ is not a pair}\end{array}}{P_i \vdash_s t : (\sigma_j, u)}\ unify$$

  It clearly follows that $\sigma_{j'}(u') = \sigma_j(u)$. But now it is clear that neither u nor u' is a pair. Further, $t : (\sigma_{j'}, u')$ is top-level matching. So there are three cases to consider.

  t is a basic term: Now u and u' are also basic terms. Hence $\sigma'_j(u) = \sigma_j(u)$ and $\sigma'_{j'}(u') = \sigma_{j'}(u')$. From the fact that $\sigma_j(u) = \sigma_{j'}(u')$, it follows that $\sigma'_j(u) = \sigma'_{j'}(u')$. But then by induction hypothesis $\sigma'_{j'}(u') \in \overline{T'_i}$, and hence we are through.

  both u and u' are encrypted terms: Since $\sigma_j(u) = \sigma_{j'}(u')$, from Proposition 1 (*which is an immediate consequence of the tagging scheme we use*, that $u = u'$. But then clearly $\sigma'_j(u) = \sigma'_{j'}(u')$ as well. By induction hypothesis $\sigma'_{j'}(u') \in \overline{T'_i}$, and hence we are through.

  u is a basic term and u' is an encrypted term: In this case, quite clearly $\sigma'_j(u)$ is the same as n_0, which belongs to T_0 and hence, to T'_i.

- Suppose π is the following proof:

$$\dfrac{\begin{array}{c}(\pi_1)\\ \vdots \\ P_i \vdash_s t : (\sigma_{j'}, u')\end{array}}{P_i \vdash_s t : (\sigma_j, u)}\ simplify$$

  In this case u is a nonce and t is not, and so quite clearly $\sigma'_j(u)$ is the same as n_0, which belongs to T_0 and hence, to T'_i.

- Suppose $t = (t_1, t_2)$, $u = (u_1, u_2)$, and π is the following proof:

$$\dfrac{\begin{array}{cc}(\pi_1) & (\pi_1)\\ \vdots & \vdots \\ P_i \vdash_s t_1 : (\sigma_j, u_1) & P_i \vdash_s t_2 : (\sigma_j, u_2)\end{array}}{P_i \vdash_s (t_1, t_2) : (\sigma_j, (u_1, u_2))}\ pair'$$

By induction hypothesis both $\sigma'_j(u_1)$ and $\sigma'_j(u_2)$ belong to $\overline{T'_i}$ and the conclusion immediately follows from this.

- Suppose $t = \{t_1\}_{t_2}$, $u = \{u_1\}_{u_2}$, and π is the following proof:

$$\dfrac{\begin{array}{cc}(\pi_1) & (\pi_1)\\ \vdots & \vdots \\ P_i \vdash_s t_1 : (\sigma_j, u_1) & P_i \vdash_s t_2 : (\sigma_j, u_2)\end{array}}{P_i \vdash_s \{t_1\}_{t_2} : (\sigma_j, \{u_1\}_{u_2})}\ encrypt'$$

By induction hypothesis both $\sigma'_j(u_1)$ and $\sigma'_j(u_2)$ belong to $\overline{T'_i}$ and the conclusion immediately follows from this. ⊣

We now have all the ingredients to prove Theorem 9.

Lemma 24 *For all $i \leq k$ such that e'_i is a receive event, $t'_i \in \overline{T'_{i-1}}$.*

Proof: Consider any receive event e'_i. Since e_i is also a receive event, it is clear that $t_i \in \overline{T_{i-1}}$. Notice that $t_i = \sigma(g_i)$. Thus, by Lemma 22, it follows that there is a proof of $P_{i-1} \vdash_s t_i : (\sigma_i, g_i)$. It now follows by Lemma 23 that $t'_i = \sigma'_i(g_i) \in \overline{T'_{i-1}}$. ⊣

The next lemma is important in proving the "equivalence" of ξ and ξ'.

Lemma 25 *For all $i, j \leq k$ and for all $m \in N$ such that $\sigma_i(m) \in N$, $\sigma_i(m) \in \overline{T_j}$ iff $\sigma'_i(m) \in \overline{T'_j}$.*

Proof: We have already shown (by Lemmas 22 and 23) that whenever $\sigma_i(m) \in \overline{T_j}$ then $\sigma'_i(m) \in \overline{T'_j}$.

Let π' be a proof of $T'_j \vdash \sigma'_i(m)$. Since $\sigma'_i(m)$ is a nonce, π' can end only with an *analz*-rule, and hence $\sigma'_i(m) \in ST(T'_j)$. Now every term r occurring in π' is a subterm of $T'_j \cup \{\sigma'_i(m)\}$, and thus is a subterm of T'_j. Since the σ'_i's are well-typed substitutions, we conclude that all terms occurring in π' are of the form $\sigma'_k(u)$ for some $u \in G_j$ and some $k \leq j$. We can easily show for each subproof ϖ' of π' with root $T'_j \vdash \sigma'_k(u)$ that $\sigma_{k'}(u) \in \overline{T_j}$ (for a k' such that whenever $\sigma_k(t)$ is a nonce, $\sigma_k(t) = \sigma_{k'}(t)$). We look at the nontrivial cases:

- Suppose ϖ' is the following proof:

$$\dfrac{}{T'_j \vdash \sigma'_k(u)}\ Ax$$

It is clear that $\sigma'_k(u) \in T'_j$. Now there are two cases to consider. The first is when $\sigma'_k(u) \neq n_0$. In this case it is clear that $\sigma_k(u) \in T_j$ as well, and thus $\sigma_k(u) \in \overline{T_j}$. On the other hand if $\sigma_k(u) \neq n_0$ and $\sigma'_k(u) = n_0$, then we observe that u itself is a nonce and $\sigma_k(u)$ is not one. Thus by Lemma 21, $\sigma_k(u)$ belongs to $\overline{T_{k-1}}$, which is a subset of $\overline{T_j}$, and we are through.

- Suppose ϖ' is the following proof:

$$\dfrac{\begin{array}{cc}(\varpi_1') & (\varpi_2') \\ \vdots & \vdots \\ T_j' \vdash \sigma_k'(\{u\}_{u'}) & T_j' \vdash \sigma_{k'}'(u')\end{array}}{T_j' \vdash \sigma_k'(u)}\ \mathit{decrypt}$$

We first observe that u' occurs as an encryptor in Pr, and thus there are no ill-typed substitutions involving subterms of u'. Thus by induction hypothesis there are k_1, k_2 such that $\sigma_{k_1}(u') = \sigma_{k_2}(u') = \sigma_k'(u')$, and that $\sigma_{k_1}(u') \in \overline{T_j}$. Also by induction hypothesis we see that $\sigma_{k_1}(\{u\}_{u'}) \in \overline{T_j}$. Thus, it follows that $\sigma_{k_1}(u) \in \overline{T_j}$, and of course, by induction hypothesis it follows that for all t such that $\sigma_k(t)$ is a nonce, $\sigma_k(t) = \sigma_{k_1}(t)$. ⊣

D From pseudo-runs to runs

In this section, we show how to associate a well-typed run ξ_{norm} to a well-typed pseudo-run ξ_{wt}.

Consider a tagged protocol $Pr = \{\eta_1, \ldots, \eta_n\}$, and a run ξ of Pr. Let $\xi = e_1 \cdots e_k$ with $e_i = (\Pi_i, lp_i)$ for each $i \leq k$. For any set of session formulas Ψ and any $i \leq k$, we define $\mathrm{Th}_\Psi(i)$ to be the set $\{\alpha \in \Psi \mid \xi{\restriction}\Pi_i, lp_i \vDash \alpha\}$.

We define ξ_{norm} as the subsequence $e_{i_1} \cdots e_{i_r}$ of ξ such that $i_1 = 1$ and for all $j > 1$, the following property holds:

$$\text{for all } i' : 1 \leq i' \leq i_j, \text{ either } s_{i'} \neq s_{i_j} \text{ or } e_{i'} \neq e_{i_j} \text{ or } \mathrm{Th}_\Psi(i') \neq \mathrm{Th}_\Psi(i_j).$$

It is clear that ξ_{norm} is indeed a run of Pr. Observe that Π is a session of ξ *if and only if* it is a session of ξ_{norm}.

For every $i : 1 \leq i \leq k$, we define i_{norm} as follows:

$$i_{norm} \stackrel{\text{def}}{=} \min\{i' \leq k \mid s_{i'} = s_i \text{ and } e_{i'} = e_i \text{ and } \mathrm{Th}_\Psi(i) = \mathrm{Th}_\Psi(i')\}.$$

Lemma 26

1. *For all sessions* Π *of* ξ*, all* $i \leq k$*, and all* $\alpha \in \Psi$*,*

$$\xi{\restriction}\Pi, lp_i \vDash \alpha \textit{ iff } \xi_{norm}{\restriction}\Pi, lp_{i_{norm}} \vDash \alpha.$$

2. *For all run formulas* φ *such that* $sf(\varphi) \cap \Phi_0 = \Psi$*, and all session assignments assign over* ξ *which are compatible with* φ*,*

$$\xi \vDash_{assign} \varphi \textit{ iff } \xi_{norm} \vDash_{assign} \varphi.$$

Proof:

1. This simply follows from the definitions. Note that $\xi{\restriction}\Pi, lp_i \vDash \alpha$ iff $\alpha \in \mathrm{Th}_\Psi(i)$ iff $\alpha \in \mathrm{Th}_\Psi(i_{norm})$ iff $\xi_{norm}{\restriction}\Pi, lp_{i_{norm}} \vDash \alpha$.

2. The proof of this is identical to that of item 2 of Lemma 10. ⊣