Journal of Computer Security 11 (2003) 677–721
IOS Press

A compositional logic for proving security properties of protocols *

Nancy Durgin [a], John Mitchell [b] and Dusko Pavlovic [c]
[a] *Sandia National Labs, P.O. Box 969, Livermore, CA 94551, USA*
Tel.: +1 925 294 4909; Fax: +1 925 294 3271; E-mail: nadurgi@sandia.gov
[b] *Computer Science Department, Stanford University, Stanford, CA 94305, USA*
E-mail: jcm@cs.stanford.edu
[c] *Kestrel Institute, Palo Alto, CA 94304, USA*
E-mail: dusko@kestrel.edu

We present a logic for proving security properties of protocols that use nonces (randomly generated numbers that uniquely identify a protocol session) and public-key cryptography. The logic, designed around a process calculus with actions for each possible protocol step, consists of axioms about protocol actions and inference rules that yield assertions about protocols composed of multiple steps. Although assertions are written using only steps of the protocol, the logic is sound in a stronger sense: each provable assertion about an action or sequence of actions holds in any run of the protocol that contains the given actions and arbitrary additional actions by a malicious attacker. This approach lets us prove security properties of protocols under attack while reasoning only about the sequence of actions taken by honest parties to the protocol. The main security-specific parts of the proof system are rules for reasoning about the set of messages that could reveal secret data and an invariant rule called the "honesty rule".

1. Introduction

There has been considerable research on formal analysis of security protocols, ranging from BAN logic and related approaches [3,10,27] to finite-state analysis [21,25] and proof methods based on higher-order logic [23]. Most approaches in current use are based on enumeration or reasoning about a set of protocol traces, each trace obtained by combining protocol actions with actions of a malicious intruder. Automated trace-based tools can be used to find protocol errors after a few days of human effort, but it remains significantly more time-consuming to prove protocols correct using logics that reason about traces. While it is difficult to give specific numbers, since efforts depend on the complexity of the protocol and the experience of those involved, it seems that most formal proofs require months of effort, even with assistance from powerful automated tools. We have therefore developed a formal logic capable of relatively abstract reasoning about protocol traces. In this

*Partially supported by the Kestrel Institute, ONR MURI "Semantic Consistency in Information Exchange", N00014-97-1-0505, and ONR Grant "Games and Security in Systems of Autonomous Agents", N0014-00-C-0495.

0926-227X/03/$8.00 © 2003 – IOS Press. All rights reserved

logic, we are able to prove properties of common authentication and secrecy protocols by derivations of twenty to sixty lines of proof. The reason for this succinctness is that the proof rules of the logic state general properties of protocol traces that can be reused for many different protocols.

The logic presented in this paper includes modal operators naming sequences of actions from a process calculus. This logic provides a method for attaching assertions to protocol actions, in a manner resembling dynamic logic for sequential imperative programs – applying Floyd–Hoare style annotations [9,11] so that the composition of the assertions associated with each action can provide the basis for a protocol correctness proof. The underlying logic is different from previous "belief" logics such as BAN and its descendants [3,10,27] and from explicit reasoning about protocol and intruder as in Paulson's inductive method [23]. The central idea is that assertions associated with an action will hold in any protocol execution that contains this action. This gives us the power to reason about all possible runs of a protocol, without explicitly reasoning about steps that might be carried out by an attacker. At the same time, the semantics of our logic is based on sets of traces of protocol execution (possibly including an attacker), not the kind of abstract idealization found in some previous logics.

Our logic uses five predicates: Sent, Decrypts, Knows, Source, and Honest. The first two make relatively simple statements about what has happened. For example, $\mathsf{Sent}(X, m)$ holds at some state in the execution of a protocol if principal X has sent the message m. The interpretation of Knows is also very mechanical, and much more elementary than in logics of knowledge. Specifically, a principal "knows" a datum if the principal either generated this datum or received it in a message in a form that is not encrypted under a key that is not known to the principal. The last two predicates are more novel. The central predicate for reasoning about secrecy, and authentication based on secrecy, is Source. Intuitively, Source is used to identify the "source" of some datum, i.e., the way that a principal might come to know the contents of some message. The predicate Honest is used primarily to assume that one party follows the prescribed steps of the protocol correctly. For example, if Alice initiates a transaction with Bob, and wishes to conclude that only Bob knows the data she sends, she must explicitly assume that Bob is honest. If Bob is not honest, meaning that Bob does not follow the protocol (or, equivalently, Bob's key is known to the attacker), then any data Alice sends to Bob could be read by the attacker and the attacker could forge all of the messages Alice receives from Bob. Therefore, many correctness assertions involve an assumption that one or more principals are honest.

Most of the axioms and inference rules of our logic are ways of attaching assertions to actions, and rules for combining these assertions when actions are combined in a role of a protocol. The main inference rule that is not of this form is a rule we refer to as the "honesty rule". This is a form of invariance rule, used to reason about all possible actions of honest principals. Suppose that in some protocol, whenever a principal receives a message of the form $\{\!|a, b|\!\}_K$, meaning the encryption of a pair (a, b) under key K, the principal then responds with $\{\!|b|\!\}_K$. Assume further that this

is the only situation in which the protocol specifies that a message consisting of a single encrypted datum is sent. Using the honesty rule, it is possible to prove that if a principal A is honest, and A sends a message of the form $\{\!|b|\!\}_K$, then A must have previously received a message of the form $\{\!|a, b|\!\}_K$. For certain authentication protocols, this form of reasoning allows us to prove that if one protocol participant completes the prescribed sequence of actions, and another principal named in one of the messages is honest, then some secret is shared between the two principals.

Section 2 describes the process calculus and Section 3 shows how we use the process calculus to express steps of a protocol. Section 4 describes the formulas and semantics of our logic. The proof system is presented in Section 5. A sample proof is given in Section 6, with discussion of related work appearing in Section 7 and concluding remarks in Section 8.

The example derivation given in Section 6 shows how to prove a significant property of Lowe's variant [14] of the Needham–Schroeder public key protocol [22]. A brief discussion in that section also shows how an attempt to prove the same property for the original Needham–Schroeder protocol fails in an insightful way. Specifically, since the main axioms and inference rules of our logic are each tied to a specific action, the outline of any possible proof is determined by the steps of the protocol. Therefore, we can reduce the problem of proving a protocol property to the problem of finding instances of axioms and rules that match in specific ways. For the well-studied Needham–Schroeder protocol, under the model of attacker capabilities used in this paper, proof of an authentication property fails precisely because initiator Alice cannot correctly establish the identity of the responder from the data she receives. In effect, the protocol logic presented in this paper leads directly to rediscovery of Lowe's observation [14].

The process calculus and logic presented in this paper support only public key encryption and cannot be used to reason about the source of an encrypted nonce if the principal who generated it sends more than one message containing the nonce. While we believe that the approach can be extended fairly easily to handle symmetric-key encryption and more general patterns of messages containing encrypted nonces, the more restricted form of logic used in this paper is simpler and easier to understand. We hope to explore extensions of the system in future work.

2. Communicating cords

Cords are the formalism we use to represent protocols and their parts. They form an action calculus [16,17,24], based on π-calculus [20], and related to spi-calculus [1]. The cords formalism is also similar to the approach of the Chemical Abstract Machine formalism [2], in that the communication actions can be viewed as reactions between "molecules". Other work that relates process calculus to strands and security protocol analysis is [5,6]. The basic idea of π-calculus is to represent communication by term reduction, so that the communication links can be created dynamically [19]. The idea of spi is to add to π the suitable constructors for encryption

and decryption, and analyze secure communication using process equivalence; some similar ideas also appear in [12]. We treat the encryption in a manner similar to spi-calculus, but decryption is reduced to term reduction. The idea of cord calculus is not so much to capture security within the meta-theory of processes, but rather to serve as a simple "protocol programming language", intuitive enough to support our Floyd–Hoare style logical annotations, and verifications in an axiomatic semantics. The formalism is designed to support protocol composition and synthesis, in addition to reasoning about protocol correctness. Although we do not explore protocol composition in this paper, the static interfaces and associated composition operators in 2.4 suggest certain forms of protocol composition that preserve logical reasoning about sequences of actions.

In fact, cords are first of all based on the informal language of arrows and messages, widely used in the security community. For instance, an arrows-and-messages picture of Lowe's variant [14] of the Needham–Schroeder public key protocol [22], which we will refer to as NSL, might look something like Fig. 1.

Strand spaces [8] have been developed in an effort towards formalizing this language. The messages are captured in a term calculus, and decorated by $+$ and $-$, respectively denoting the send and the receive actions. The roles are then presented as sequences of such actions, called *strands*. Viewed as a strand space, the above protocol run is shown in Fig. 2.

The fact that an agent only sees his or her own actions, *viz* sending and receiving messages, is reflected in the strand formalism. However, communication, the fact that by receiving a message, an agent may learn something new, is not reflected. E.g., in the above example, the strand B already contains the term $\{|A, m|\}_B$, and appears to

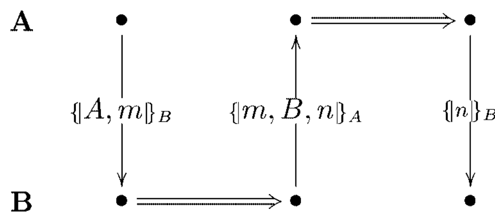

Fig. 1. NSL as arrows and messages.

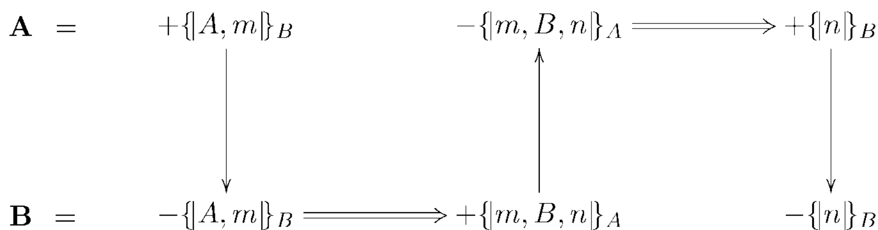

Fig. 2. NSL as a strand space.

know the exact form of the message that he is about to receive, including A's fresh nonce, before the communication is ever initiated. Indeed, the formalism is set up so that the particles $+\{\!|A, m|\!\}_B$ and $-\{\!|A, m|\!\}_B$ can react only when the terms involved exactly coincide.

In strand spaces, which provide the basis for a series of interesting results and applications, roles are treated as families of strands, parameterized by all the possible values that can be received. However, we found this approach not only somewhat artificial, but also technically insufficient for our form of reasoning about secure communication. For instance, it is difficult to identify the data "known" to a strand when an agent in a protocol is parameterized by values unknown to them. Such parameters come about, e.g., when a role is sent a term encrypted by someone else's key, which it should forward, rather than attempt to decrypt. More generally, a formal occurrence of a subterm in a strand is unrelated to the knowledge of the agent to whom that strand belongs.

Cord spaces are a result of our effort to overcome such shortcomings. In comparison with strands, we add variables to the term calculus. Of course, just like a parameter, a variable is just a placeholder for a family of values; but variables come with a formal binding and substitution mechanism. The action of receiving a value into a variable x is expressed by the operator (x), which binds the occurrences of x to the right of it. The action of sending a term t is now written $\langle t \rangle$, rather than $+t$. When the term t is closed, i.e., reducible to a value, the particles (x) and $\langle t \rangle$ can *react*: they are eliminated, and t is substituted for all occurrences of x that were bound to (x). The value propagation resulting from the communication is modelled by the substitution.

The cord space, corresponding to the above protocol is shown in Fig. 3. Here we introduce the notation (νm) which is a binding operation denoting the generation of a new nonce, m. A generates m and sends the term $\{\!|A, m|\!\}_B$ which B now receives into the variable x, and substitutes for it on the right. In particular, the pattern-matching operator $(x/\{\!|Y, z|\!\}_{\overline{B}})$, which represents asymmetric or public key encryption, is now instantiated to $(\{\!|A, m|\!\}_B/\{\!|Y, z|\!\}_{\overline{B}})$. The matching succeeds, and the values A and m get substituted for Y and z. The term $\{\!|z, B, n|\!\}_Y$ is thus instantiated to $\{\!|m, B, n|\!\}_A$, which contains no variables any more, and can be sent. Now A receives this term into the variable u, and substitutes it into $(u/\{\!|m, B, v|\!\}_{\overline{A}})$.

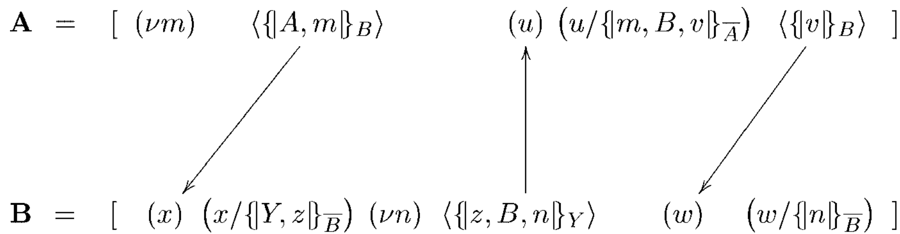

Fig. 3. NSL as a cord space.

The encryption is matched against the expected encryption, the first two encrypted components against the nonce m and the name B, whereas the third component, the nonce n is substituted for the variable v. The term $\{\!|v|\!\}_B$ now becomes a value, which is sent, received into w and pattern matched, *viz* decrypted, and tested to be equal to the nonce n.

A formal definition of cords requires several syntactical steps.

2.1. Terms and actions

The terms t are built starting from the variables x and the constants c. Moreover, the set of basic terms also contains the names N, which can be variables X, Y, Z, or constants A, B, C, and keys K which can be variables y and constants k.

Upon these basic sets, the term language is then generated by some given constructors p, which always include tupling, and the public key encryption $\{\!|t|\!\}_K$, of the term t by the key K.

The language of actions is then built upon the terms by further constructors. They include sending a term $\langle t\rangle$, receiving into a variable (x), matching a term against a pattern $(t/q(x))$, and creating a new value (νx). The extensions may allow other actions, such as reading time, or point-to-point communication. The language of terms and actions is summarized in Table 1.

A pattern is a term with variables, into which we substitute other terms. The syntax of patterns is summarized in Table 2. Just as a term t may be a tuple of terms, a variable x may be a tuple of variables. This allows us to write x instead of $\vec{x}$ for $x_1, \ldots, x_n$. We write $p(x_1, \ldots, x_n)$ if the list $x_1, \ldots, x_n$ contains all the variables in p. If $p(x_1, \ldots, x_n)$ is a pattern and $t = t_1, \ldots, t_n$ is a term, then $p(t) = [t_1, \ldots, t_n/x_1, \ldots, x_n]p$ is the term obtained by substituting $t_1, \ldots, t_n$ for $x_1, \ldots, x_n$ in p.

Some examples of patterns are:

$$\begin{aligned}
p_1(x_1, x_2) &= x_1, x_2 \\
p_2(x_1, x_2) &= x_1, A, x_2 \\
q_1(x_1, x_2) &= \{\!|p_1(x_1, x_2)|\!\}_{\overline{K}} \\
&= \{\!|x_1, x_2|\!\}_{\overline{K}} \\
q_2(x_1, x_2) &= \{\!|p_2(x_1, x_2)|\!\}_{\overline{K}} \\
&= \{\!|x_1, A, x_2|\!\}_{\overline{K}}
\end{aligned}$$

Here, the patterns p_1 and p_2 are basic tuple patterns, each taking two arguments, with p_2 containing the constant A. The patterns q_1 and q_2 are decryption patterns, again taking two arguments and containing the value K as the key.

We introduce decryption patterns in order to characterize asymmetric decryption. The decryption key $\overline{K}$ is required to decrypt a message $\{\!|m|\!\}_K$, where the notation

Table 1
Terms and actions

(names)	$N ::= X$	variable name
	A	constant name
(basic keys)	$K_0 ::= y$	variable key
	k	constant key
	N	name
(keys)	$K ::= K_0$	basic key
	$\overline{K_0}$	inverse key
(terms)	$t ::= x$	variable term
	c	constant term
	N	name
	K	key
	t, t	tuple of terms
	$\{\mid t \mid\}_K$	term encrypted with key K
(actions)	$a ::= \epsilon$	the null action
	$\langle t \rangle$	send a term t
	(x)	receive term into variable x
	(νx)	generate new term x
	$(t/q(x_1, \ldots, x_n))$	match term t to pattern $q(x_1, \ldots, x_n)$

Table 2
Patterns

(basic terms)	$b ::= x \mid c \mid N \mid K$	basic terms allowed in patterns
(basic patterns)	$p ::= b, \ldots, b$	tuple pattern
(patterns)	$q ::= p$	basic pattern
	$\{\mid p \mid\}_{\overline{K}}$	decryption pattern

$\{\mid _ \mid\}__$ represents asymmetric encryption (with the encryption key different from the decryption key). As we shall see in Section 2.3, the action of matching a pattern $(\{\mid x \mid\}_A / \{\mid z \mid\}_{\overline{A}})$ binds x to z in the strand to the right, thus decrypting the message $\{\mid x \mid\}_A$ to reveal the plaintext x.

Strands, defined by the following grammar, are lists of actions.

$$\text{(strands)} \quad S ::= aS \mid a$$

Strands include operations that are not efficiently computable. For example, the strand $(x)(y)(x/\{\mid z \mid\}_{\overline{y}})\langle z \rangle$ receives a message x and encryption key y, then decrypts x with the *decryption* key $\overline{y}$ associated with y. There is no known way to compute $\overline{y}$ efficiently from y. However, since we wish to give principal A access to a key pair K_A and $\overline{K_A}$, which we write A and $\overline{A}$ for notational convenience, it is useful to have syntax for inverse keys. In section Section 3, additional restrictions will be imposed

on strands that represent roles in a protocol (or attacks against a protocol) so that all actions performed in a protocol run are efficiently computable.

2.2. Strand order and cords

Although a strand lists actions in a specific order, one order may not be distinguishable from another. This occurs when a variable bound in one action does not occur free in another. For example, consider the strand

$$(x)(y)\langle x, y\rangle$$

which receives two inputs and then sends a pair comprised of the two inputs. Since we choose not to assume that network communication preserves message order, this strand is equivalent to

$$(y)(x)\langle x, y\rangle$$

To see that these are equivalent, imagine running each in parallel with a strand that outputs numbers 2 and 3. Since the outputs 2 and 3 can be received in either order by the first strand, the first strand could output $\langle 2, 3\rangle$ or $\langle 3, 2\rangle$, which is clearly the set of possible outputs of the second strand.

In general, the strands S and T will be considered independent if no values can be passed between them. Formally, we capture this by defining an equivalence relation $\approx$ that includes the relation $\sim$ defined by

$$ST \sim TS \iff \begin{cases} FV(S) \cap BV(T) = \emptyset \\ FV(T) \cap BV(S) = \emptyset \end{cases}$$

where the operators FV and BV, giving the sets of the free and the bound variables (respectively), are inductively defined as follows:

$$\begin{aligned} FV(\langle t\rangle S) &= FV(t) \cup FV(S) \\ FV((x)S) &= FV(S) \setminus \{x\} \\ FV((t/q(x))S) &= FV(t) \cup (FV(S) \setminus \{x\}) \\ FV((\nu x)S) &= FV(S) \setminus \{x\} \end{aligned}$$

$$\begin{aligned} BV(\langle t\rangle S) &= BV(S) \\ BV((x)S) &= BV(S) \cup \{x\} \\ BV((t/q(x))S) &= BV(S) \cup \{x\} \\ BV((\nu x)S) &= BV(S) \cup \{x\} \end{aligned}$$

The set $FV(t)$ of the free variables occurring in a term t is defined as usual: whenever a variable x is used in the formation of a term t, it is added to $FV(t)$.

The actions "receive" (x), "match" (or "test") $(t/q(x))$ and "new" (νx) thus bind the variable x. The scope is always to the right, and name clashes are avoided by renaming bound variables. A value is propagated through a strand by substituting it for x everywhere within the scope of a binding operator. In this way, the condition $FV(S) \cap BV(T) = \emptyset$ indeed ensures that S cannot depend on T.

The relation $\approx$ is the least congruence containing the transitive, reflexive closure of $\sim$, and closed under α-conversion (renaming the bound variables). The strands S and T will thus be $\approx$-equivalent if and only if one can be obtained from the other by renaming the bound variables, and permuting the actions *within the scopes of the bound variables*, i.e., in such a way that no free variable becomes bound, or *vice versa*. Note that $\approx$ preserves the free variables of a strand.

Using calligraphic $\mathcal{S}$ for the set of all possible strands, we let *cords* be equivalence classes of strands, modulo the relation $\approx$,

$$\mathcal{C} = \mathcal{S}/_{\approx}$$

To indicate a specific cord, we enclose the strand in brackets [ ], which serves to indicate both the scope of binding and that the cord encompasses all equivalent strands modulo $\approx$. That is

$$[S]_{\approx} = \{S' \mid S' \approx S\}$$

We omit the relation $\approx$ in the rest of the paper.

The relation $\approx$ allows renaming of bound variables and reordering of non-conflicting actions. For example, $\approx$ identifies strands like $(x)\langle c\rangle$ and $\langle c\rangle(x)$, with the independent actions permuted. In contrast, the actions (x) and $\langle \{\!|x|\!\}_A \rangle$ are not independent, because a value must be received into x *before* it can be sent out in a message. Sending a pair $\langle a, b\rangle$ preserves the order of terms a and b, while sending the two values one at a time, $\langle a\rangle\langle b\rangle$, does not.

2.3. Cord spaces and runs

A cord space is a multiset of cords that may interact via communication. We use $\otimes$ for multiset union and [ ] for the empty multiset. In the terminology associated with the Chemical Abstract Machine [2], a cord space is a "soup" in which the particles (cords) may react. We use a cord space consisting of a set of protocol roles (each represented by a cord) to represent a state and set of remaining actions of a protocol. For instance, one possible run of the NSL protocol, with Alice initiating a conversation with Bob and no other protocol roles (or intruder cords) involved, arises from the cord space

$$\mathbf{A} \otimes \mathbf{B} = [(\nu x)\langle \{\!|A, x|\!\}_B \rangle \ldots] \ \otimes \ [(x)(x/\{\!|Y, z|\!\}_{\overline{B}}) \ldots]$$

Table 3

Basic reaction steps

$$[S(x)S'] \otimes [T\langle t\rangle T'] \otimes C \rhd\rhd [SS'(t/x)] \otimes [TT'] \otimes C \quad (2)$$

$$[S(p(t)/p(x))S'] \otimes C \rhd\rhd [SS'(t/x)] \otimes C \quad (3)$$

$$[S(\{\!|p(t)|\!\}_y/\{\!|p(x)|\!\}_{\overline{y}})S'] \otimes C \rhd\rhd [SS'(t/x)] \otimes C \quad (4)$$

$$[S(\nu x)S'] \otimes C \rhd\rhd [SS'(m/x)] \otimes C \quad (5)$$

Where the following conditions must be satisfied:

(2) $FV(t) = \emptyset$

(3) $FV(t) = \emptyset$

(4) $FV(t) = \emptyset$ and y bound

(5) $x \notin FV(S)$ and $m \notin FV(C) \cup FV(S) \cup FV(S')$

Mathematically speaking, cord spaces form a free commutative monoid generated by cords, with $\otimes$ as the binary operation and [ ] as the unit. We sometimes write ellipses "..." inside a cord to indicate the presence of arbitrary additional actions.

Reactions

The basic reactions within a cord space are shown in Table 3, with the required side conditions for each reaction shown below them. The substitution (t/x) is assumed to act on the strand left of it, *viz* S'. As usual, it is assumed that no free variable becomes bound after substitution, which can always be achieved by renaming of the bound variables.

Reaction (2) is a send and receive interaction, showing the simultaneous sending of term t by the first cord, with the receiving of t into variable x by the second cord. We call this an *external action* because it involves an interaction between two cords.

The other reactions all take place within a single cord. We call these *internal actions*. Reaction (3) is a basic pattern match action, where the cord matches the pattern $p(t)$ with the expected pattern $p(x)$, and substitutes t for x. Note that as mentioned in Section 2.1, the placeholder x in the pattern $p(x)$ can be a tuple of variables, $x_1, \ldots, x_n$, in which case the term must also be a tuple $t_1, \ldots, t_n$.

Reaction (4) is a decryption pattern match action, where the cord matches the pattern $\{\!|p(t)|\!\}_y$ with the decryption pattern $\{\!|p(x)|\!\}_{\overline{y}}$ and substitutes t for x.

Finally, reaction (5) shows the binding action where the cord creates a new value that doesn't appear elsewhere in the cordspace, and substitutes that value for x in the cord to the right.

The intuitive motive for the condition $FV(t) = \emptyset$ should be clear: a term cannot be sent, or tested, until all of its free variables are instantiated. In addition in a decryption action, the key y used in the pattern must be bound. In other words, our variables are not public names, or references that could be passed around, but strictly private stores. This security-specific feature distinguishes cord calculus from the closely related, but general-purpose process and action calculi, to which it otherwise owes the basic ideas and notations.

Table 4

NSL example reaction

$\mathbf{A} \otimes \mathbf{B} = [(\nu x)\langle \{\|A, x\|\}_B \rangle (u) \ldots] \otimes [(x)(x/\{\|Y, z\|\}_{\overline{B}}) \ldots]$	(a)
$\rhd\rhd [\langle \{\|A, m\|\}_B \rangle (u) \ldots] \otimes [(x)(x/\{\|Y, z\|\}_{\overline{B}})(\nu y)\langle \{\|z, B, y\|\}_Y \rangle \ldots]$	(b)
$\rhd\rhd [(u) \ldots] \otimes [(\{\|A, m\|\}_B/\{\|Y, z\|\}_{\overline{B}})(\nu y)\langle \{\|z, B, y\|\}_Y \rangle \ldots]$	(c)
$\rhd\rhd [(u)(u/\{\|m, B, v\|\}_{\overline{A}}) \ldots] \otimes [(\nu y)\langle \{\|m, B, y\|\}_A \rangle (w) \ldots]$	(d)
$\rhd\rhd [(u)(u/\{\|m, B, v\|\}_{\overline{A}}) \ldots] \otimes [\langle \{\|m, B, n\|\}_A \rangle (w) \ldots]$	(e)
$\rhd\rhd [(\{\|m, B, n\|\}_A/\{\|m, B, v\|\}_{\overline{A}})\langle \{\|v\|\}_B \rangle] \otimes [(w) \ldots]$	(f)
$\rhd\rhd [\langle \{\|n\|\}_B \rangle] \otimes [(w)(w/\{\|n\|\}_{\overline{B}})]$	(g)
$\rhd\rhd [\ ] \otimes [(\{\|n\|\}_B/\{\|n\|\}_{\overline{B}})]$	(h)
$\rhd\rhd [\ ]$	

Runs

The *runs* of a protocol arise as reaction sequences of cord spaces. The run of the NSL protocol displayed in Figs 1–3 can now be completely formalized as a sequence of syntactic reaction steps, which are shown in Table 4.

Steps (a) and (d) in Table 4 use rule (5), steps (b), (e) and (g) are based on rule (2), and steps (c), (f) and (h) on rule (4). In a sense, these eight reduction steps correspond to the five arrows in Figs 1 and 2, plus the explicit actions to create a new value and the final test of n that B performs, which were omitted in the diagrams. On the other hand, the three arrows that appear in Fig. 3 correspond to the applications of rule (2). The actions based on rules (4) and (5) were represented in Fig. 3 not by arrows, but by displaying the corresponding actions.

2.4. Static binding and cord category

Many protocols of interest consist of a set of roles. For example, the Needham–Schroeder public-key protocol, mentioned earlier, has an initiator role and a responder role. An initiator starts an exchange by sending a message to a responder. In a local area network, for example, with principals (users or machines) named Alice, Bob, Charley, and Dolores, the protocol could be used simultaneously or in succession by several principals for several purposes. Alice may initiate an exchange with Bob and, before completing the three messages associated with this exchange, respond to an exchange initiated by Charlie. Concurrently, Charlie may initiate an exchange with Bob, and respond to a request from Dolores. We consider the set of messages exchanged by these four parties, together with any actions performed by an attacker, a possible run of the protocol.

Since a run may consist of several instances of the protocol roles, executed concurrently, it is useful to have a notation for a role that has not been assigned to any specific principal. In this section, we summarize the basic idea. However, since the mechanisms described here do not play a central role in this paper, the presentation is informal, proceeding by example rather than focussing on the general definitions.

The main concepts that are needed in the remainder of the paper are that roles contain variables indicating principals, these variables are bound in the static interface of the role, and the variables are replaced by specific principals in constructing a run of the protocol.

Two cords of the NSL protocol, given in Fig. 3, are:

$$\mathbf{A} = [(\nu x)\langle \{\!|A, x|\!\}_B\rangle(u)(u/\{\!|x, B, v|\!\}_{\overline{A}})\langle \{\!|v|\!\}_B\rangle]$$
$$\mathbf{B} = [(x)(x/\{\!|Y, z|\!\}_{\overline{B}})(\nu y)\langle \{\!|z, B, y|\!\}_Y\rangle(w)(w/\{\!|y|\!\}_{\overline{B}})]$$

These are the initiator and responder roles of the NSL protocol, assigned to principals A and B, respectively. Replacing the names A and B with variables X and Y, and listing the "instantiable" variables outside the square brackets of the cord, we have:

$$\mathbf{Init} = (X\ Y)[(\nu x)\langle \{\!|X, x|\!\}_Y\rangle(u)(u/\{\!|x, Y, v|\!\}_{\overline{X}})\langle \{\!|v|\!\}_Y\rangle]\langle\rangle$$
$$\mathbf{Resp} = (X)[(x)(x/\{\!|Y, z|\!\}_{\overline{X}})(\nu y)\langle \{\!|z, X, y|\!\}_Y\rangle(w)(w/\{\!|y|\!\}_{\overline{X}})]\langle\rangle$$

We consider the lists $(X\ Y)$ and (X) binding operators, so that principal variables X and Y are bound in these cords. The list are called the static interfaces of the cords, since we model replacing X and Y with *Alice* and *Bob* as a static operation that occurs before the dynamic execution of a run of the protocol.

In the cords **Init** and **Resp** above, empty angle brackets, $\langle\rangle$ appear at the end of each cord. Generally, angle brackets may contain an export list which is used to compose cords. Composition, written using ";", is achieved by substitution. For example,

$$(X\ Y)[(\nu x)\langle x\rangle]\langle X\rangle; (Z)[(y)\langle \{\!|y|\!\}_Z\rangle]\langle\rangle = (X\ Y)[(\nu x)\langle x\rangle(y)\langle \{\!|y|\!\}_X\rangle]\langle\rangle$$

Note that in performing the composition, the exported X from the first cord is substituted for imported Z in the second. Since composition provides substitution, we introduce an abbreviation

$$\mathbf{Init}(A\ B) = (\,)[\,]\langle A\ B\rangle\ ;\ \mathbf{Init}$$

to indicate the cord with its static interface instantiated to the values A and B. This gives us $\mathbf{A} = \mathbf{Init}(A\ B)$ and $\mathbf{B} = \mathbf{Resp}(B)$.

For the reader familiar with action calculus, we observe that cords, taken as particles, generate an action category [18,24]. This is the source of the equivalence relation $\approx$. The idea is that a cord space C, displayed in the form

$$\gamma = (x_0 \dots x_{i-1})C\langle t_0 \dots t_{j-1}\rangle$$

can be viewed as an arrow $\gamma : i \to j$, where arities i,j are the objects of the category. The variables $x_\bullet$, assumed mutually different, form the input interface: the operator

$(x_0, \ldots, x_i)$ *binds* their occurrences to the right. The terms $t_\bullet$ in the output interface may not be mutually different, nor different from $x_\bullet$. Of course, all expressions are up to variable renaming (α-conversion).

Given a morphism $\delta : j \to k$, in the form

$$\delta = (u_0 \ldots u_{j-1}) D \langle v_0 \ldots v_{k-1} \rangle$$

the composite $\gamma\ ;\ \delta : i \to k$ will be the cord morphism

$$\gamma\ ;\ \delta = (x_0 \ldots x_{i-1}) C D(\vec{t}/\vec{u}) \langle v_0 \ldots v_{k-1} \rangle$$

where it is assumed that the names in the interfaces of γ and δ have been chosen so that no clashes occur when $\vec{t}$ is substituted for $\vec{u}$.

The idea is that the dynamic binding by (x) and (νx) captures value propagation by communication. The *static binding* of the input interface is now used to compose agents at design time. The static interfaces are thus *not* used for passing any actual messages, but for propagating the public keys, connecting the various roles of the same principal, and for static links in general, independent of and prior to the execution. The cord **Init** above can be designed as the composite $\mathbf{Init} = \mathbf{I_0}\ ;\ \mathbf{I_1}$, where

$$\mathbf{I_0} = (X\ Y)[(\nu x)\langle \{\!|X, x|\!\}_Y \rangle]\langle X\ Y\ x \rangle$$
$$\mathbf{I_1} = (\tilde{X}\ \tilde{Y}\ z)[(u)(u/\{\!|z, \tilde{Y}, v|\!\}_{\overline{\tilde{X}}})\langle \{\!|v|\!\}_{\tilde{Y}} \rangle]\langle\rangle$$

The constant values of A and B can be passed to it, at design time, by precomposing it with the morphism $(\ )[\ ]\langle A\ B \rangle$, i.e., $\mathbf{A} = (\ )[\ ]\langle A\ B \rangle; \mathbf{Init}$.

Cord morphisms represent *processes*:

$$\text{(processes)} \quad r ::= (x \ldots x) C \langle x \ldots x \rangle$$

In a cord space, the interfaces of processes combine in the natural way, that is

$$(\vec{x}) C \langle \vec{t} \rangle \otimes (\vec{u}) D \langle \vec{v} \rangle = (\vec{x}\vec{u}) C \otimes D \langle \vec{t}\vec{v} \rangle$$

where $\vec{x}$ is disjoint from $\vec{u}$ and the variables in $\vec{t}$ are disjoint from the variables in $\vec{v}$, to avoid naming conflicts. Since these disjointness conditions may be satisfied by suitable renaming, it is always possible to move the static interfaces to the outside of the cord space, and carry out all reactions within their scope.

Returning to the example NSL cords, if we define a substitution process $\sigma = (\ )[\ ]\langle ABB \rangle$, then

$$\mathbf{A} \otimes \mathbf{B} = \sigma\ ;\ (\mathbf{Init} \otimes \mathbf{Resp}).$$

Static binding plays a limited role in the present paper. We will display the interface only when it is relevant, and in other cases assume that all variables are bound.

3. Protocols

3.1. Protocol roles

A *protocol* is defined by a finite set of roles, such as initiator, responder and server, each carried out by one or more participants in any protocol execution. We identify the principal who is carrying out the role by writing the name of the principal as a subscript on the square brackets. The name of the principal is used to make sure that important cryptographics restrictions are handled properly. Formally, we distinguish roles from cords by calling a cord with identified principal, subject to the key condition given below, a *role*. Using variables from their static interface, the initiator and responder roles are now written precisely as

$$\textbf{Init} = (X\ Y)[(\nu x)\langle \{\!|X, x|\!\}_Y\rangle(u)(u/\{\!|x, Y, v|\!\}_{\overline{X}})\langle\{\!|v|\!\}_Y\rangle]_X$$
$$\textbf{Resp} = (X)[(x)(x/\{\!|Y, z|\!\}_{\overline{X}})(\nu y)\langle\{\!|z, X, y|\!\}_Y\rangle(w)(w/\{\!|y|\!\}_{\overline{X}})]_X$$

with the static output interfaces omitted since they are empty.

Formally, we define NSL protocol to be the set whose members are precisely these two roles,

$$\mathit{NSL} = \{\textbf{Init}, \textbf{Resp}\} \tag{1}$$

Roles are composed using substitution, in exactly the same way that cords are composed, with substitution performed on the subscript indicating the acting principal if it is a variable from the static interface.

The technical reason for identifying the principal carrying out a role is to specify the correct use of private keys. A *private key* is a key of form $\overline{X}$, which represents the decryption key in a public key cryptosystem. In a protocol role, the only place $\overline{X}$ is allowed to occur is in the decryption position of a decryption pattern of a role belonging to principal X. For example, the following cord

$$(X\ Y)[(x)(x/\{\!|y|\!\}_{\overline{Y}})]_X$$

is not a well-formed role, because this role allows the principal X to decrypt a message using Y's private key. The syntactic key restriction on roles prevents a role from "computing" the value of a private key from the public key. The following cord, which applies $\overline{(\,)}$ to a key it receives,

$$(X)[(x)(y)(x/\{\!|z|\!\}_{\overline{y}})]_X$$

is not a well-formed protocol role because the private key $\overline{y}$ used in the role is not the private key of the acting principal.

Since private keys can only occur in the decryption key position of a decryption pattern match, the key restriction prevents private keys from being sent in a message. While some useful protocols might send private keys, we prevent roles from sending their private keys (in this paper) since this allows us to take secrecy of private keys as an axiom, shortening proofs of protocol properties. (Otherwise, we would have to prove that in any protocol of interest, each role maintains the secrecy of its private key.) The key restriction does not prevent a decryption key from being received dynamically and used in a decryption action. For example, the following cord

$$(X)[(x)(y)(y/\{\!|z|\!\}_x)]_X$$

does not violate the syntactic key restriction. We expect to consider roles that may generate key pairs and send decryption keys in later versions of the protocol logic.

3.2. Intruder role

The protocol intruder is capable of taking any of several possible actions, including receiving a message, decomposing it into parts, decrypting the parts if the key is known, remembering parts of messages, and generating and sending new messages. This is the standard "Dolev-Yao model", which appears to have developed from positions taken by Needham and Schroeder [22] and a model presented by Dolev and Yao [7]; see also [4]. Although we use cords to represent the actions of a protocol intruder, the intruder may have several private keys available to it – by referring to "the" intruder, we do not mean to suggest that an attack must be carried out by a single principal.

In each run of a protocol, the intruder will be represented by a cord with an arbitrary collection of cord calculus actions from Section 2.1, subject only to a restriction on decryption keys defined in this section. We specify the set of decryption actions available to the intruder using a subsidiary definition of the set of decryption keys used in a cord. If C is a cord, then *DKeys*(C) is the set of names that appear as private keys in the cord, that is:

$$\mathit{DKeys}(C) = \{X \mid \overline{X} \text{ appears in } C\}$$

Formally, an *intruder role with keys* $K_1, \ldots, K_n$ is a multiset I of cords such that *Dkeys*(I) contains only $K_1, \ldots, K_n$ and the private key function $\overline{(\,)}$ only occurs in the decryption position of a decryption pattern match action. This restriction will prevent an intruder from using any key that has not been corrupted, and prevents the intruder from computing the private key from a public key.

3.3. Protocol configurations and runs

A *run of a protocol* is a sequence of reaction steps from an *initial configuration*. An *initial configuration* is determined by a set of principals, a subset of which are

designated as honest, a cord space constructed by assigning one or more roles to each honest principal, and an intruder cord that may use only the secret keys of dishonest principals. We give an example without an intruder cord and a more complicated example with an intruder cord. Although we could assign protocol roles to corrupted keys, there is no need to do so. The reason is simply that when a key is available to the intruder, the intruder can simulate any number of protocol roles using that key.

Here is an example initial configuration allowing A to initiate a conversation with B, C a conversation with A, and allowing both A and B to respond using the responder role:

$$\mathbf{Init}(A\ B) \otimes \mathbf{Init}(C\ A) \otimes \mathbf{Resp}(A) \otimes \mathbf{Resp}(B) =$$
$$[(\nu x)\langle \{\!|A, x|\!\}_B\rangle \ldots]_A \otimes\ [(\nu x)\langle \{\!|C, x|\!\}_A\rangle \ldots]_C \otimes$$
$$[(x)(x/\{\!|Y, z|\!\}_{\overline{A}}) \ldots]_A \otimes [(x)(x/\{\!|Y, z|\!\}_{\overline{B}}) \ldots]_B$$

Using a substitution cord, $\sigma = (\ )[\ ]\langle A\ B\ C\ A\ A\ B\rangle$, the initial configuration can also be written as a composition of cords

$$\sigma\ ;\ (\mathbf{Init} \otimes \mathbf{Init} \otimes \mathbf{Resp} \otimes \mathbf{Resp}).$$

More generally, any initial configuration can be expressed using a selection of protocol roles, an intruder role, and a substitution cord morphism that connects the interfaces of all the roles. Each initial configuration has only one intruder role, but this role may be selected arbitrarily from all possible intruder roles.

For example, let $P_C = \{U_1, U_2, \ldots, U_\ell\}$ be a set of principals, and let subset *HONEST*$(\mathbf{C}) = \{U_1, \ldots, U_k\}$ be a set of honest principals. Let $\mathbf{I_C}$ be any intruder role with decryption keys among $U_{k+1}, \ldots, U_\ell$. To define an initial configuration with n **Init** roles, m **Resp** roles, and intruder $\mathbf{I_C}$, we define the substitution

$$\sigma_{\mathbf{C}} = (\ )[\ ]\langle X_1\ Z_1\ X_2\ Z_2 \ldots X_n\ Z_n\ Y_1\ Y_2 \ldots Y_m\ U_1\ U_2 \ldots U_l\rangle.$$

Then an initial configuration of the *NSL* protocol is given by the cord space

$$\mathbf{C} = \mathbf{Init}(X_1\ Z_1) \otimes \mathbf{Init}(X_2\ Z_2) \otimes \ldots \otimes \mathbf{Init}(X_n\ Z_n) \otimes$$
$$\mathbf{Resp}(Y_1) \otimes \mathbf{Resp}(Y_2) \otimes \ldots \otimes \mathbf{Resp}(Y_m) \otimes$$
$$\mathbf{I_C}(U_1\ U_2 \ldots U_l)$$

where $X_1, X_2, \ldots X_n, Y_1, Y_2, \ldots, Y_n$ are honest keys in P_C. The intruder role may use any encryption keys among $U_1\ U_2 \ldots U_\ell$, but only the decryption keys that are not in *HONEST*$(\mathbf{C})$.

Table 5

Initial configuration of Needham–Schroeder protocol

$$
\begin{aligned}
P_C &= \{A, B, E, I\}\\
\mathit{HONEST}(\mathbf{C}) &= \{A, B\}\\
\mathbf{I_C} &= (X\ Y\ Z)[(x)(x/\{\!|X, z|\!\}_{\overline{Z}})\langle\{\!|X, z|\!\}_Y\rangle(w)(w/\{\!|x|\!\}_{\overline{Z}})\langle\{\!|x|\!\}_Y\rangle]_I\\
\sigma_{\mathbf{C}} &= (\,)[\,]\langle A\ E\ B\ A\ B\ E\rangle\\
\mathbf{C} &= \sigma_{\mathbf{C}}; (\mathbf{Init_{NS}} \otimes \mathbf{Resp_{NS}} \otimes \mathbf{I_C})\\
&= \mathbf{Init_{NS}}(A\ E) \otimes \mathbf{Resp_{NS}}(B) \otimes \mathbf{I_C}(A\ B\ E)\\
&= [(\nu x)\langle\{\!|A, x|\!\}_E\rangle(u)(u/\{\!|x, v|\!\}_{\overline{A}})\langle\{\!|v|\!\}_E\rangle]_A \otimes\\
&\quad\ [(x)(x/\{\!|Y, z|\!\}_{\overline{B}})(\nu y)\langle\{\!|z, y|\!\}_Y\rangle(w)(w/\{\!|y|\!\}_{\overline{B}})]_B \otimes\\
&\quad\ [(x)(x/\{\!|A, z|\!\}_{\overline{E}})\langle\{\!|A, z|\!\}_B\rangle(w)(w/\{\!|x|\!\}_{\overline{E}})\langle\{\!|x|\!\}_B\rangle]_I
\end{aligned}
$$

Example: Lowe's run of the original Needham–Schroeder protocol. As an example, consider Lowe's anomaly in the original Needham–Schroeder protocol [13]. The original Needham–Schroeder protocol is represented by the following cords

$$
\begin{aligned}
\mathbf{Init_{NS}} &= (X\ Y)[(\nu x)\langle\{\!|X, x|\!\}_Y\rangle(u)(u/\{\!|x, v|\!\}_{\overline{X}})\langle\{\!|v|\!\}_Y\rangle]_X\\
\mathbf{Resp_{NS}} &= (X)[(x)(x/\{\!|Y, z|\!\}_{\overline{X}})(\nu y)\langle\{\!|z, y|\!\}_Y\rangle(w)(w/\{\!|y|\!\}_{\overline{X}})]_X
\end{aligned}
$$

It differs from *NSL* only in the second message, where the identity of the responder is omitted.

An initial configuration which will exhibit Lowe's anomaly is shown in Table 5. It has four principals: A and B who are honest, E whose private key is known to the intruder, and I who is the intruder. It is also possible for I and E to be the same principal, but we choose different entities to emphasize that the intruder can be unknown to the honest participants, but able to masquerade as any dishonest principal, in particular one whom A mistakenly believes to be honest. The cord space contains one initiator cord, in which A chooses to talk to E, and a responder cord belonging to B. The intruder cord, $\mathbf{I_C}$, contains the actions necessary to intercept the messages from A and make it look like they were intended for B. Note that in this configuration, $\mathbf{I_C}(A\ B\ E)$ is a legal intruder cord, because it does not contain any decryption actions that require the private keys of the honest principals A and B, *i.e.*, $\mathit{DKeys}(\mathbf{I_C}(A\ B\ E)) = \{E\}$.

3.4. Events and traces

A particular initial configuration may give rise to many possible runs. One possible run of the Needham–Schroeder configuration in Table 5 is shown in Table 6. In this run, all the actions shown in each cord take place. Another possible run of this configuration is one where A's first message is sent to B instead of I, and since B can't decrypt it, no further reactions are possible. In a more complex configuration, there would be many more possible runs.

Table 6

Run showing Lowe's anomaly in Needham–Schroeder

$$
\begin{aligned}
\mathbf{C} \;&=\; \mathbf{Init}_{\mathbf{NS}}(A\ E) \otimes \mathbf{Resp}_{\mathbf{NS}}(B) \otimes \mathbf{I}_{\mathbf{C}}(A\ B\ E)\\
&=\; [(\nu x)\langle \{\!|A,x|\!\}_E\rangle(u)\ldots]_A \otimes [\ldots]_B \otimes [(x)(x/\{\!|A,z|\!\}_{\overline{E}})\ldots]_I\\
&\rhd\!\rhd\; [\langle \{\!|A,m|\!\}_E\rangle(u)\ldots]_A \otimes [\ldots]_B \otimes [(x)(x/\{\!|A,z|\!\}_{\overline{E}})\ldots]_I\\
&\rhd\!\rhd\; [(u)\ldots]_A \otimes [\ldots]_B \otimes [(\{\!|A,m|\!\}_E/\{\!|A,z|\!\}_{\overline{E}})\langle \{\!|A,z|\!\}_B\rangle\ldots]_I\\
&\rhd\!\rhd\; [\ldots]_A \otimes [(x)(x/\{\!|Y,z|\!\}_{\overline{B}})\ldots]_B \otimes [\langle \{\!|A,m|\!\}_B\rangle(w)\ldots]_I\\
&\rhd\!\rhd\; [\ldots]_A \otimes [(\{\!|A,m|\!\}_B/\{\!|Y,z|\!\}_{\overline{B}})(\nu n)\langle \{\!|z,n|\!\}_Y\rangle\ldots]_B \otimes [(w)\ldots]_I\\
&\rhd\!\rhd\; [(u)(u/\{\!|m,v|\!\}_{\overline{A}})\ldots]_A \otimes [(\nu y)\langle \{\!|m,y|\!\}_A\rangle(w)\ldots]_B \otimes [\ldots]_I\\
&\rhd\!\rhd\; [(u)(u/\{\!|m,v|\!\}_{\overline{A}})\ldots]_A \otimes [\langle \{\!|m,n|\!\}_A\rangle(w)\ldots]_B \otimes [\ldots]_I\\
&\rhd\!\rhd\; [(\{\!|m,n|\!\}_A/\{\!|m,v|\!\}_{\overline{A}})\langle \{\!|v|\!\}_E\rangle]_A \otimes [(w)\ldots]_B \otimes [\ldots]_I\\
&\rhd\!\rhd\; [\langle \{\!|n|\!\}_E\rangle]_A \otimes [\ldots]_B \otimes [(w)(w/\{\!|x|\!\}_{\overline{E}})\ldots]_I\\
&\rhd\!\rhd\; [\ ]_A \otimes [\ldots]_B \otimes [(\{\!|n|\!\}_E/\{\!|x|\!\}_{\overline{E}})\langle \{\!|x|\!\}_B\rangle]_I\\
&\rhd\!\rhd\; [\ ]_A \otimes [(w)(w/\{\!|n|\!\}_{\overline{B}})]_B \otimes [\langle \{\!|n|\!\}_B\rangle]_I\\
&\rhd\!\rhd\; [\ ]_A \otimes [(\{\!|n|\!\}_B/\{\!|n|\!\}_{\overline{B}})]_B \otimes [\ ]_I\\
&\rhd\!\rhd\; [\ ]_A \otimes [\ ]_B \otimes [\ ]_I
\end{aligned}
$$

Since the protocol logic we introduce in Section 4 reasons about protocol runs, we need to introduce some additional notation for them.

An *event* is a ground substitution instance of an action, i.e., an action in which all variables have been replaced by terms containing only constants. An event represents the result of a reaction step, viewed from the perspective of a single cord that participated in it. For example, if A sends message m to B, then the event $\langle m\rangle$ is a send event of A and (m) is a receive event of B. The initial cord of A may have contained the send action $\langle m\rangle$, or may have contained a send event $\langle x\rangle$, with some action before the send replacing x by m. The initial cord of B must have contained a receive action (y), with the reaction step replacing y by m. Since receive actions contain variables, (m) is not a well-formed action, but we consider this expression an event.

A *trace* is a list of the events by some principal that occur in a run. We use events as a bookkeeping mechanism and as the syntax for writing out traces.

Let $R|_X$ denote the events that occurred for principal X in run R. For example, if R is the run shown in Table 6, then we can write $R|_X$ as a trace (for $X = A, B, I$) showing the actions taken by X in run R, as follows

$$
\begin{aligned}
R|_A &= [(\nu m)\langle \{\!|A,m|\!\}_E\rangle(\{\!|m,n|\!\}_A)(\{\!|m,n|\!\}_A/\{\!|m,v|\!\}_{\overline{A}})\langle \{\!|n|\!\}_E\rangle]\\
R|_B &= [(\{\!|A,m|\!\}_B)(\{\!|A,m|\!\}_B/\{\!|Y,z|\!\}_{\overline{B}})(\nu n)\langle \{\!|m,n|\!\}_A\rangle(\{\!|n|\!\}_B)(\{\!|n|\!\}_B/\{\!|n|\!\}_{\overline{B}})]\\
R|_I &= [(\{\!|A,m|\!\}_E)(\{\!|A,m|\!\}_E/\{\!|A,z|\!\}_{\overline{E}})\langle \{\!|A,m|\!\}_B\rangle(\{\!|n|\!\}_E)(\{\!|n|\!\}_E/\{\!|x|\!\}_{\overline{E}})\langle \{\!|n|\!\}_B\rangle]
\end{aligned}
$$

Note that in this simple example, the list of actions performed by each principal corresponds exactly to the three traces from the run. In a more complex example

where a principal acted in more than one role, actions from multiple roles could occur in the same run, and might be interleaved in an arbitrary fashion.

For actions P, protocol $\mathcal{Q}$, run R and principal X, we say "P *matches* $R|_X$ for $\mathcal{Q}$" precisely if $R|_X$ is the interleaving of a set of initial segments of traces of roles of $\mathcal{Q}$ carried out by X, and one of the segments of one of the roles ends in exactly σP, where σ is a substitution of values for variables. If P matches $R|_X$ using substitution σ, then σ is called the *matching substitution*.

Lemma 3.1. *For any configuration* **C** *of protocol $\mathcal{Q}$, and any run R, if principal $X \in$ HONEST(***C***), then $R|_X$ is an interleaving of traces of roles of $\mathcal{Q}$ executed by X.*

Proof. This follows from the definition of an initial configuration, which is constructed by assigning one or more roles from $\mathcal{Q}$ to each honest principal. □

A succinct notation for talking about specific events occurring in a run is

$$\mathit{EVENT}(R, X, P, \vec{n}, \vec{x}) \equiv$$
$$(([SPS']_X \otimes C \quad \triangleright\triangleright \quad [SS'(\vec{n}/\vec{x})]_X \otimes C') \in R)$$

In words, $\mathit{EVENT}(R, X, P, \vec{n}, \vec{x})$ means that in run R, principal X executes actions P, receiving data $\vec{n}$ into variables $\vec{x}$, where $\vec{n}$ and $\vec{x}$ are the same length. The predicate can be true for non-empty vectors $\vec{n}$ and $\vec{x}$ only if P is a receive or pattern match action. In the above example,

$$\mathit{EVENT}(R, A, \langle \{\!|A, m|\!\}_E \rangle, \emptyset, \emptyset),$$
$$\mathit{EVENT}(R, A, \langle \{\!|n|\!\}_E \rangle, \emptyset, \emptyset),$$
$$\mathit{EVENT}(R, B, (\{\!|A, m|\!\}_B / \{\!|Y, z|\!\}_{\overline{B}}), \{A, m\}, \{Y, z\}), \text{ and}$$
$$\mathit{EVENT}(R, A, (\nu x), m, x)$$

are all examples of true facts about run R.

3.5. *Protocol properties*

In this section, we collect some properties of the class of protocols we will reason about in the rest of the paper.

No Telepathy. It is straightforward to check that if a role sends a message, then all the subterms of the message must be values that were either created by the role, received by the role, or that were known to the role from its static parameters.

Lemma 3.2 (No Telepathy). *Let $\mathcal{Q}$ be a protocol, R be an arbitrary run, X be a principal, and $R|_X$ consist of initial segments of traces $\rho_1, \rho_2, \ldots, \rho_k$, where each ρ_i is a role of $\mathcal{Q}$. Let m be any message sent by X as part of role ρ_i. Then every symbol in the term m is either generated in ρ_i, received in ρ_i, or was in the static interface of ρ_i.*

Proof. This follows from the definition of the cords we use to represent roles. Each role is a closed cord, so all values must be bound. Symbols can be bound by the static interface, or by the ν, receive and pattern match actions. □

Asynchronous communication. Cord reactions, as we have defined them, require synchronous communication. That is, a message send action cannot happen in one cord unless a message receive action occurs simultaneously in another cord. Real network communication is asynchronous, though – the network itself effectively provides a buffer in which messages can be stored until somebody is ready to receive them. In order to model this with cords, we pad the intruder cord with an arbitrary number of "$(x)\langle x\rangle$" actions, which we call *forwarding actions*. We call this a *buffering intruder cord*. These forwarding actions model a message being received, and then eventually sent. Because these forwarding actions are independent from (share no variables with) the other actions in the cord, they can occur in any order, which effectively models the asynchronous nature of the network.

We define an *adequate buffering intruder cord* with respect to a configuration **C** as a buffering intruder cord that has enough forwarding actions to guarantee that every send action in the configuration can be taken immediately. For any configuration **C**, we can construct an adequate buffering intruder cord by counting all the send actions occurring in the configuration, and including at least that many forwarding actions in the intruder cord.

Lemma 3.3 (Asynchronous Communication). *In a configuration with an adequate buffering intruder cord, any role that wishes to send a message can always send it.*

Proof. Since the buffered intruder cord provides a corresponding receive action for each send in the configuration, this action is always available for reaction with a send action. □

This lemma allows us to schedule reactions runs so that the only place where protocol roles are required to pause is at a receive action. All intermediate actions between receives (i.e., new, pattern match and send actions) can be assumed to all occur in the run.

Standard form traces. One useful way of reasoning about protocols, as well as other kinds of computational systems, is by establishing invariance properties. The simplest form of invariance is a property that holds at every possible state of protocol execution. However, some useful properties might only hold at certain states. For

example, a property that says that whenever a message of one form is received, another one is sent, depends on the ability of a protocol agent to progress. The property is true if we look at certain "good" states, but may fail at intermediate states where a protocol agent is in the process of responding to a message. For the purpose of strengthening the invariance rule in Section 4, we define a set of preferred states and a standard form for traces that progress from one preferred state to another.

We say a protocol role is *hung* if it is unable to complete its next internal action. The only internal action that can hang is a pattern match action. For example, a role such as

$$[(x)(x/\{\!|y, B|\!\}_{\overline{A}}) \ldots]_A$$

can become hung if the data received in the (x) action is $\{\!|B, n|\!\}_A$, which doesn't match the pattern $\{\!|y, B|\!\}_A$.

We say a protocol configuration (a cord space resulting from an initial configuration by a sequence of reaction steps) is a *good state* if no internal actions are pending, and no protocol role is hung. Recall that the reaction steps are divided into communication steps and internal actions. An internal action is pending in a state if there is some role that is able to execute an internal reaction.

A *standard trace* is one where all internal actions occur as early as possible. Starting from an initial configuration, each role takes all available internal actions, bringing the configuration to a state where it is ready to perform an external action.

If we consider traces to be equivalent if they have the same external actions in the same order, then every trace is equivalent to a standard trace. A trace that contains a hung protocol role is equivalent to a trace in which the last message received by the hung role was received by the intruder instead. We can therefore consider only standard traces in our analysis. In keeping with this, whenever we write a protocol role, we may write it in the standard order, with all internal actions listed as early as possible. For example, the **Resp** role may be written

$$\mathbf{Resp} = (X)[(\nu y)(x)(x/\{\!|Y, z|\!\}_{\overline{X}})\langle\{\!|z, X, y|\!\}_Y\rangle(w)(w/\{\!|y|\!\}_{\overline{X}})]_X$$

with the (νy) action moved all the way to the left.

Secrecy Restrictions. For simplicity, our model places a secrecy restriction on runs. By design, it is impossible for any message encrypted with the public key of an honest principal to be decrypted by anybody other than that principal. This follows because (1) we do not allow the intruder cord to contain any static actions that decrypt using an honest principal's key (nor do we model any way for an intruder to "guess" a secret key), and (2) we reason only about protocols where the private keys can only appear in the decrypt pattern match action, meaning the roles can not send out their private keys in any message.

This leads to the following Lemma:

Lemma 3.4 (Secrecy). *For any configuration* **C** *of protocol* $\mathcal{Q}$, *and any run* R,

$$EVENT(R, X, (\{\!|t|\!\}_Y / \{\!|x|\!\}_{\overline{Y}}), t, x) \wedge Y \in HONEST(\mathbf{C}) \supset X = Y.$$

Proof. Let ρ be the role where the event occured in run R. Let σ_C be the substitution used in configuration **C**. First, suppose $Y \in FV(\rho)$.

If ρ is the intruder role, then ρ cannot contain this action, because $(DKeys(\sigma_C\ ;\ \rho) \cap HONEST(\mathbf{C})) = \emptyset$.

If ρ is a protocol role, then $Dkeys(\sigma_C\ ;\ \rho) = \{X\}$, from the restriction on decryption keys in well-formed protocol roles, so Y can't be in $DKeys(\sigma_C\ ;\ \rho)$ unless $X = Y$.

Suppose $Y \notin FV(\rho)$. Then somehow $\overline{Y}$ (the decryption key) must have been received in a message by ρ. But since the restriction on private keys appearing only in decryption actions means that a well formed protocol role cannot contain any actions that send private keys in a message, this is not possible. □

A direct consequence of this is Axiom **SEC** in Table 8, which basically says that "if X is honest and anyone Y decrypts a message with the private key of X, then in fact $X = Y$".

While this lemma holds for the system presented in this paper, we believe it is possible to relax the secrecy assumptions and add explicit rules to the logic that allow us to prove this property for appropriate protocols. For example, if the restriction on the use of private keys was removed, then it would still be possible to prove for a certain protocol that private keys were never sent in any message. So, Axiom **SEC** could still be proven to hold for a specific protocol.

4. A protocol logic

The protocol logic lets us reason about properties that are guaranteed to hold after a principal has performed a certain sequence of actions.

4.1. Syntax

The formulas of the logic are given by the following grammar, where ρ may be any role, written using the notation of the cord calculus (Section 3.3):

$$\begin{array}{lll}
\text{(predicate formulas)} & \phi ::= & \mathsf{Sent}(N, t) \\
 & & \mathsf{Knows}(N, t) \\
 & & \mathsf{Source}(t, N, t, N) \\
 & & \mathsf{Decrypts}(N, t) \\
 & & \mathsf{Honest}(N) \\
 & & \phi \wedge \phi \\
 & & \neg\phi \\
 & & \forall x.\phi \\
\text{(modal forms)} & \Psi ::= & \rho\ \phi
\end{array}$$

Here, t and N are terms and names, as defined in Section 2.1. We use ϕ and ψ to indicate predicate formulas, and m to indicate a generic term we call a "message". Quantification $\forall x.\phi$ means ϕ is true for all messages x, which includes all names and keys.

Most protocol proofs use formulas of the form $[P]_X \phi$, which means that after X executes actions P, formula ϕ is true about the resulting state of X. Here are the informal interpretations of the predicates, with precise semantics given in the next section:

The formula $\mathsf{Knows}(X, x)$ means that principal X knows information x. This is "knows" in the limited sense of having either generated the data or received it in the clear or received it under encryption where the decryption key is known. If X knows $\{\!|m|\!\}_K$ and does not know $\overline{K}$, then X does *not* know m from this message.

The formula $\mathsf{Sent}(X, m)$ means that principal X sent message m. This implies that X knows the message m.

The formula $\mathsf{Decrypts}(X, t)$ means that principal X received a message t in a role that expected the message and was able to decrypt it. Note that $\mathsf{Decrypts}(X, t)$ can only be true if t is of form $\{\!|m|\!\}_K$. This implies that X knows the message $\{\!|m|\!\}_K$, and the decrypted text, m.

The formula $\mathsf{Honest}(X)$ means that the actions of principal X in the current run are precisely an interleaving of initial segments of traces of a set of roles of the protocol. In other words, X assumes some set of roles and does exactly the actions prescribed by them. A principal may be considered honest in a run if either its key is not corrupted, or the key is corrupted but the intruder only behaves (with this key) according to the roles of the protocol.

The Source predicate is used to reason about the source of a piece of information, such as a nonce. Intuitively, the formula

$$\mathsf{Source}(n, X, t, K)$$

means that the only way for a principal Y different from X to know n is if Y learned n from the message $\{\!|t|\!\}_K$, possibly by some indirect path. Note that $\mathsf{Source}(n, X, t, K)$ can only be true if n is a term generated by a (νx) action, and if n is never sent out unencrypted by X.

4.2. Semantics

A formula may be true or false at a run of a protocol. More precisely, the main semantic relation, $\mathcal{Q}, R \models \phi$, may be read, "formula ϕ holds for run R of protocol $\mathcal{Q}$". In this relation, R may be a complete run, with all roles that are started in the run completed, or an incomplete run with some principals waiting for additional messages to complete one or more roles.

As preliminaries to the inductive definition of $\mathcal{Q}, R \models \phi$, we make a few definitions regarding runs and the intruder. If $\mathcal{Q}$ is a protocol, then let $\bar{\mathcal{Q}}$ be the set of

all initial configurations of protocol $\mathcal{Q}$, each including a possible intruder cord. Let $\mathsf{Runs}(\bar{\mathcal{Q}})$ be the set of all runs of protocol $\mathcal{Q}$ with intruder, as described in Section 3.3, each a sequence of reaction steps within a cord space.

We use $m \subseteq m'$ to indicate that m is a subterm of m'.

We define satisfaction $\mathcal{Q}, R \models \phi$, for ϕ without free variables, by induction on ϕ as follows:

- $\mathcal{Q}, R \models \mathsf{Sent}(A, m)$ if $\mathit{EVENT}(R, A, \langle m \rangle, \emptyset, \emptyset)$.
- $\mathcal{Q}, R \models \mathsf{Knows}(A, m)$ if there exists an i such that $\mathsf{Know}_i(A, m)$ where Know_j is defined by induction on j as follows:

$$
\begin{aligned}
&(\mathsf{Know}_0(A, m) \text{ if } ((m \in FV(R|_A)) \\
&\quad \vee \mathit{EVENT}(R, A, (\nu x), m, x) \\
&\quad \vee \mathit{EVENT}(R, A, (x), m, x) \\
&\text{and } \mathsf{Know}_{i+1}(A, m) \text{ if } (\mathsf{Know}_i(A, m') \\
&\quad \wedge ((m' = \{\!|p(t)|\!\}_K \wedge m \subseteq t \\
&\qquad \wedge \mathit{EVENT}(R, A, (m'/\{\!|p(y)|\!\}_{\overline{K}}), t, y)) \\
&\quad \vee (m' = p(t) \wedge m \subseteq t \\
&\qquad \wedge \mathit{EVENT}(R, A, (m'/p(y)), t, y)))) \\
&\text{or } (\mathsf{Know}_i(A, m') \wedge \mathsf{Know}_i(A, m'') \\
&\quad \wedge ((m = m', m'') \vee (m = m'', m'))) \\
&\text{or } (\mathsf{Know}_i(A, m') \wedge \mathsf{Know}_i(A, K) \\
&\quad \wedge m = \{\!|m'|\!\}_K)
\end{aligned}
$$

  Intuitively, Know_0 holds for terms that are known directly, either as a free variable of the role, or as the direct result of receiving or generating the term. Know_{i+1} holds for terms that are known by applying i operations (decomposing via pattern matching, or composing via encryption or tupling) to terms known directly.
- $\mathcal{Q}, R \models \mathsf{Decrypts}(A, \{\!|m|\!\}_K)$ if $\mathcal{Q}, R \models \mathsf{Knows}(A, \{\!|m|\!\}_K)$

$$\wedge \mathit{EVENT}(R, A, (\{\!|m|\!\}_K / \{\!|x|\!\}_{\overline{K}}), m, x)$$

  Note: $\mathsf{Decrypts}(A, n)$ is *false* if $n \neq \{\!|m|\!\}_K$ for some m and K.
- $\mathcal{Q}, R \models \mathsf{Source}(m, A, t, K)$ if

$$
\begin{aligned}
&\mathit{EVENT}(R, A, (\nu x), m, x) \wedge \\
&\mathcal{Q}, R \models \forall Z.((Z \neq A \wedge \mathsf{Knows}(Z, m)) \supset \mathsf{Decrypts}(Z, \{\!|t|\!\}_K) \vee
\end{aligned}
$$

$(\exists Y.\mathsf{Decrypts}(Y, \{|t|\}_K) \wedge \mathsf{Sent}(Y, t')))$

where $m \subseteq t'$

- $\mathcal{Q}, R \models \mathsf{Honest}(A)$ if $A \in$ *HONEST*(**C**) in initial configuration **C** for R.
- $\mathcal{Q}, R \models (\phi_1 \wedge \phi_2)$ if $\mathcal{Q}, R \models \phi_1$ and $\mathcal{Q}, R \models \phi_2$
- $\mathcal{Q}, R \models \neg\phi$ if $\mathcal{Q}, R \not\models \phi$
- $\mathcal{Q}, R \models \forall x.\phi$ if for every message m generated from the names and keys by term operations, $\mathcal{Q}, R \models [m/x]\phi$.
- $\mathcal{Q}, R \models [P]_A\phi$ if either P does not match $R|_A$ or P matches $R|_A$ and $\mathcal{Q}, R \models \sigma\phi$, where σ is the substitution matching P to $R|_A$.

If ϕ has free variables, then $Q, R \models \phi$ if we have $Q, R \models \sigma\phi$ for all substitutions σ that eliminate all the free variables in ϕ. We write $\mathcal{Q} \models \phi$ if $\mathcal{Q}, R \models \phi$ for all $R \in \mathsf{Runs}(\bar{\mathcal{Q}})$.

Note that the Source predicate only mentions a single message that contains a nonce created by principal X. We have investigated extensions which support reasoning about the situation where principal X may have sent more than one message containing a nonce he created, but we do not present the semantics here.

5. Proof system

5.1. Axioms and rules about protocol actions

The axioms and inference rules about protocol actions are listed in Table 7. For the most part, these are relatively simple "translations" of actions into atomic formulas of the logic.

Intuitively, Axiom **AN1** says that if X generates m, then the only principal that knows m is X. Axiom **AN2** says that if principal X generates a new value m and does no further actions in this role, then X knows m, while Axiom **AN3** is a somewhat vacuous statement (needed to provide a base case for Axiom **S1**), which states

Table 7

Axioms and rules for protocol actions

AN1	$[(\nu m)]_X\ \mathsf{Knows}(Y, m) \supset (Y = X)$
AN2	$[(\nu m)]_X\ \mathsf{Knows}(X, m)$
AN3	$[(\nu m)]_X\ \forall t.\forall K.\mathsf{Source}(m, X, t, K)$
AS1	$[\langle m\rangle]_X\ \mathsf{Sent}(X, m)$
AR1	$[(m)]_X\ \exists Y.\mathsf{Sent}(Y, m)$
AR2	$[(m)]_X\ \mathsf{Knows}(X, m)$
AD1	$[(\{|m|\}_K / \{|x|\}_{\overline{K}})]_X\ \mathsf{Decrypts}(X, \{|m|\}_K)$
AM1	$(x_1 \ldots x_n)[\ ]_X\ \mathsf{Knows}(X, x_1) \wedge \ldots \wedge \mathsf{Knows}(X, x_n)$
S1	$\dfrac{[P]_X \mathsf{Source}(m,X,t,K)}{[P\langle\{|t|\}_K\rangle]_X \mathsf{Source}(m,X,t,K)} m \subseteq t$

that if principal X generates a new value m and does no further actions in this role, then any encrypted message could be a possible source for another principal to have learned m.

Axioms **AR1** and **AR2** are about receiving messages. After X receives a message m, he knows that somebody must have sent it, and he also knows the message m.

Axiom **AD1** says that after a principal performs a pattern match action containing a decryption pattern, then the principal has decrypted the message.

Axiom **AM1** is about the binding of static variables of a protocol role. In this case the x_i in $(x_1 \ldots x_n)[\ ]_X$, is a value determined when the roles for each participant were assigned. Typically this will be the identity of the participant, and possibly the identity of other participants an initiator will try to talk to, along with shared keys, etc.

Perhaps the most subtle is the inference rule **S1**, which says that if X sends a message containing a nonce m, then that message becomes the possible source from which another principal might learn m.

Here is a sample soundness proof, for Axiom **AN1**. An important part of the soundness argument is that if $[(\nu m)]_X$ matches some run, then (νm) is the *last* action in some role carried out by X. In more detail,

$$\mathcal{Q}, R \models [(\nu m)]_X \ \mathsf{Knows}(Y, m) \supset (Y = X)$$

if $\mathcal{Q}, R \models \sigma\mathsf{Knows}(Y, m) \supset (Y = X)$ whenever (νm) matches $R|_X$, where σ is a matching substitution. By definition of *matches*, (νm) matches $R|_X$ with $\sigma(x) = m$ only if (νx) is the last action of one of the roles of X in R. But since this is the last action from that role in R, X cannot have sent m to any other principal, and from the side conditions for the ν reaction in Table 3 we know that m does not occur elsewhere in the cord space. Since no events of form $\mathit{EVENT}(R, X, \langle m' \rangle, \emptyset, \emptyset)$, where $m \subseteq m'$, have occurred, we know from Lemma B.1 that there can be no events $\mathit{EVENT}(R, Y, (x), m', x)$. So from the semantics of Knows, $\mathsf{Knows}(Y, m)$ can't be true for any principal besides X. Therefore, from the semantics of Knows, only X knows m. This shows that axiom AN1 is sound.

5.2. Axioms relating atomic predicates

Table 8 lists axioms relating various propsitional properties, most of which follow naturally from the semantics of propositional formulas. For example, if X decrypts $\{\!|m|\!\}_K$, then X knows m because that is the result of the decryption, and if a principal knows a tuple x, y then he also knows x and y.

The **SRC** axiom is important for reasoning about the source of secret values. If a value m was created by X and has only been sent in the message $\{\!|t|\!\}_K$, then if somebody else has learned m, then somebody must have decrypted the message $\{\!|t|\!\}_K$.

Table 8

Relationships between properties

DEC1	$\mathsf{Decrypts}(X, \{\!|m|\!\}_K) \supset \mathsf{Knows}(X, \{\!|m|\!\}_K)$
DEC2	$\mathsf{Decrypts}(X, \{\!|m|\!\}_K) \supset \mathsf{Knows}(X, m)$
K1	$\mathsf{Knows}(X, (x, y)) \supset \mathsf{Knows}(X, x) \wedge \mathsf{Knows}(X, y)$
SRC	$\mathsf{Source}(m, X, t, K) \wedge \mathsf{Knows}(Z, m) \wedge Z \neq X \supset$ $\exists Y.\mathsf{Decrypts}(Y, \{\!|t|\!\}_K)$
SEC	$\mathsf{Honest}(X) \wedge \mathsf{Decrypts}(Y, \{\!|m|\!\}_X) \supset (Y = X)$

$\mathsf{CSent}(X, \{\!|m|\!\}_K) \equiv \mathsf{Knows}(X, m) \wedge \mathsf{Knows}(X, K) \wedge \mathsf{Sent}(X, \{\!|m|\!\}_K)$

Table 9

Inference rules

Generic Rules:

$$\frac{[P]_A\phi \quad [P]_A\psi}{[P]_A\phi\wedge\psi}\mathbf{G1} \qquad \frac{[P]_A\phi \quad \phi\supset\psi}{[P]_A\psi}\mathbf{G2} \qquad \frac{\phi}{[P]_A\phi}\mathbf{G3}$$

Preservation Rules: (For $\mathsf{Persist} \in \{\mathsf{Knows}, \mathsf{Sent}, \mathsf{Decrypts}\}$)

$$\frac{[P]_A\mathsf{Persist}(X,t)}{[P\langle m\rangle]_A\mathsf{Persist}(X,t)}\mathbf{P1} \qquad \frac{[P]_A\mathsf{Persist}(X,t)}{[P(x)]_A\mathsf{Persist}(X,t)}\mathbf{P2}$$

$$\frac{[P]_A\mathsf{Persist}(X,t)}{[P(\nu x)]_A\mathsf{Persist}(X,t)}\mathbf{P3} \qquad \frac{[P]_A\mathsf{Persist}(X,t)}{[P(u/q(x))]_A\mathsf{Persist}(X,t)}\mathbf{P4}$$

$$\frac{[P]_A\mathsf{Source}(m,Y,t,K)}{[P\langle\langle t'\rangle\rangle]_A\mathsf{Source}(m,Y,t,K)}(m \not\subseteq t')\ \mathbf{P5} \qquad \frac{[P]_A\mathsf{Source}(m,Y,t,K)}{[P(x)]_A\mathsf{Source}(m,Y,t,K)}\mathbf{P6}$$

$$\frac{[P]_A\mathsf{Source}(m,Y,t,K)}{[P(\nu x)]_A\mathsf{Source}(m,Y,t,K)}\mathbf{P7} \qquad \frac{[P]_A\mathsf{Source}(m,Y,t,K)}{[P(u/q(x))]_A\mathsf{Source}(m,Y,t,K)}\mathbf{P8}$$

Axiom **SEC** follows immediately from Lemma 3.4. This axiom says that the decryption keys of honest principals are secret.

Note that CSent is an abbreviation which will be useful in our proofs. It can be thought of as meaning “created and sent”.

5.3. *Preservation rules*

Most predicates are preserved by additional actions. For example, if X knows m before an action, then X will also know m after the action, regardless of what the action is. The reason is that we define the knowledge of X to include all data available to X at any step of the protocol execution. Inference rules showing preservation of properties are shown in Table 9.

The exception to preservation is Source. Since $\mathsf{Source}(m, X, t, K)$ means the only way for a principal other than X to know m is from the message $\{\!|t|\!\}_K$, this atomic formula may become false if X sends another message containing m. We see this reflected in the side condition for preservation rule **P5** – the Source predicate is only preserved for m after a send operation if the message t' that was sent does not contain m as a subterm.

In all of these formulas, the "principal" referred to is a single principal who may be participating in multiple roles. A consequence of our formulation of protocols is that, while a principal Alice may participate in many instances of many roles at the same time, there will be no communication (other than via message sends and receives) between the various instances if Alice is honest. This is the "No Telepathy Lemma", Lemma 3.2.

5.4. The honesty rule

Intuitively, the honesty rule is used to combine facts about one role with inferred actions of other roles. For example, suppose Alice receives a response from a message sent to Bob. Alice may wish to use properties of Bob's role to reason about how Bob generated his reply, for example. In order to do so, Alice may assume that Bob is honest and derive consequences from this assumption. Perhaps an analogy will help. In the game of bridge, one player may call out a series of bids. The player's partner may then draw conclusions about the cards in the player's hand. If we were to formalize the partner's reasoning, we might use implications such as, "if the player is following the Blackwood bidding convention, then she has three aces". The intuition is that a bid provides a signal to the other player, and the exact meaning of the signal is determined by the bidding conventions that the partners have established. In the same way, a message may imply something specific if the principal sending the message is following the protocol. But if the principal has revealed his private key to the attacker, for example, then receipt of that message does not provide the same information about the principal.

The honesty rule is essentially an invariance rule for proving properties of all roles of a protocol. Since honesty, by definition in our framework, means "following one or more roles of the protocol", honest principals must satisfy every property that is a provable invariant of the protocol roles.

Our notion of "honesty" is a generalization of the notion of "faithfulness" in [26]. There, a participant is *faithful* if they only proceed when they receive the message they are expecting to receive, and if they always start the protocol at the beginning of their role (i.e., a faithful participant will not respond to message 3 of a protocol unless they have already received and responded to messages 1 and 2).

Recall that a protocol $\mathcal{Q}$ is a set of roles, $\mathcal{Q} = \{\rho_1, \rho_2, \ldots, \rho_k\}$. If $\rho \in \mathcal{Q}$ is a role of protocol $\mathcal{Q}$, we write $P \subseteq \rho$ if P is an initial segment of the actions of role ρ such that the next action of ρ after P is a receive, or P is a complete execution of the role. The reason for only considering initial segments up to reads is that we know from the Asynchronous Communication Lemma (Lemma 3.3), that if a role contains a send, the send may be done asynchronously without waiting for another role to receive. Therefore, we can assume without loss of generality that the only "pausing" states of a principal are those where the role is waiting for input. If a role calls for a message to be sent, then we dictate that the principal following this role must complete the send before pausing.

Table 10

Honesty rule for *NSL*

$$\frac{\begin{array}{l} \mathit{NSL} \vdash (X\ Y)[(\nu x)\langle \{\!|X, x|\!\}_Y \rangle]_X\ \phi(X) \\ \mathit{NSL} \vdash (X\ Y)[(\nu x)\langle \{\!|X, x|\!\}_Y \rangle(u)(u/\{\!|x, Y, y|\!\}_{\overline{X}})\langle \{\!|y|\!\}_Y \rangle]_X\ \phi(X) \\ \mathit{NSL} \vdash (X)[(\nu y)]_X\ \phi(X) \\ \mathit{NSL} \vdash (X)[(\nu y)(u)(u/\{\!|Y, x|\!\}_{\overline{X}})\langle \{\!|x, X, y|\!\}_Y \rangle]_X\ \phi(X) \\ \mathit{NSL} \vdash (X)[(\nu y)(u)(u/\{\!|Y, x|\!\}_{\overline{X}})\langle \{\!|x, X, y|\!\}_Y \rangle(w)(w/\{\!|y|\!\}_{\overline{X}})]_X\ \phi(X) \end{array}}{\mathit{NSL} \vdash \mathsf{Honest}(X) \supset \phi(X)}\ \mathbf{HON}$$

Since the honesty rule depends on the protocol, we write $\mathcal{Q} \vdash [P]\phi$ if $[P]\phi$ is provable using the honesty rule for $\mathcal{Q}$ and the other axioms and proof rules. If the honesty rule is not needed to prove $[P]\phi$, then we can also write $\mathcal{Q} \vdash [P]\phi$, since $\mathcal{Q} \vdash [P]\phi$ for any protocol Q.

Using the notation just introduced, the honesty rule may be written

$$\frac{\forall \rho \in \mathcal{Q}. \forall P \subseteq \rho. \mathcal{Q} \vdash (\vec{Z})[P]_X\ \phi}{\mathcal{Q} \vdash \mathsf{Honest}(X) \supset \phi}\mathbf{HON} \qquad \begin{array}{l}\text{no free variable in } \phi \\ \text{except } X \text{ bound in} \\ (\vec{Z})[P]_X\end{array}$$

Where $(\vec{Z})[P]_X$ is the initial steps P of role ρ with static variables $(\vec{Z})$, and no free variable in ϕ other than X is bound by $(\vec{Z})[P]_X$. In words, if every role of $\mathcal{Q}$, run either to completion or to a receiving state, satisfies ϕ, then every honest principal executing protocol $\mathcal{Q}$ must satisfy ϕ. The side condition prevents free variables in the conclusion $\mathsf{Honest}(X) \supset \phi$ from becoming bound in any hypothesis.

More concretely, the honesty rule for the *NSL* protocol defined in equation (1) is shown in Table 10. Here, the antecedents of the rule enumerate the intermediate "waiting for input" states and final completed states of each of the roles of the protocol. If some formula ϕ expressible in our logic holds in these five local traces, then ϕ will hold for any honest principal executing this protocol.

The honesty rule is used in the proof of correctness of *NSL* that is given in full in Section 6. One place the honesty rule is used is to prove that if B completes the responder role, and the principal A sending the final message to B is honest, then A also sent the initial message to B. The key part of this deduction is to prove the formula

$$\begin{aligned} \phi(A) = \forall m, n, B. &\mathsf{Decrypts}(A, \{\!|m, B, n|\!\}_A) \supset \\ &(\mathsf{CSent}(A, \{\!|A, m|\!\}_B) \wedge \mathsf{CSent}(A, \{\!|n|\!\}_B)) \end{aligned}$$

holds for honest participant A. This formula is proved using the honesty rule in line 10 of B's deduction in Table 13.

To give some feel for how the honesty rule works in this case, we give an informal, intuitive explanation of how the rule yields ϕ. Referring to the rule as instantiated for *NSL*, we can see that the antecedent $\mathsf{Decrypts}(A, \{\!|m, B, n|\!\}_A)$ is never true

Table 11

$\mathcal{R}$(**Init**) – Deductions from **Init** role

AM1	$(X\ Y)[\ ]_X$ Knows$(X, X) \wedge$ Knows(X, Y)	(1)
AN2	$[(\nu m)]_X$ Knows(X, m)	(2)
AS1	$[\langle \{\!\mid X, m \mid\!\}\langle_Y\rangle]_X$ Sent$(X, \{\!\mid X, m \mid\!\}_Y)$	(3)
AD1	$[(\{\!\mid m, Y, n \mid\!\}_X / \{\!\mid m, Y, x \mid\!\}_{\overline{X}})]_X$ Decrypts$(X, \{\!\mid m, Y, n \mid\!\}_X)$	(4)
AS1	$[\langle \{\!\mid n \mid\!\}_Y \rangle]_X$ Sent$(X, \{\!\mid n \mid\!\}_Y)$	(5)
1, 2, **P3**	$(X\ Y)[(\nu m)]_X$ Knows$(X, X) \wedge$ Knows$(X, Y) \wedge$ Knows(X, m)	(6)
3, 6, **P1**	$(X\ Y)[(\nu m)\langle \{\!\mid X, m \mid\!\}_Y \rangle]_X$ Knows$(X, X) \wedge$ Knows$(X, Y) \wedge$ Knows$(X, m) \wedge$ Sent$(X, \{\!\mid X, m \mid\!\}_Y)$	(7)
7, **P2**	$(X\ Y)[(\nu m)\langle \{\!\mid X, m \mid\!\}_Y \rangle(\{\!\mid m, Y, n \mid\!\}_X)]_X$ CSent$(X, \{\!\mid X, m \mid\!\}_Y)$	(8)
4, 8, **P4**	$(X\ Y)[(\nu m)\langle \{\!\mid X, m \mid\!\}_Y \rangle(\{\!\mid m, Y, n \mid\!\}_X)(\{\!\mid m, Y, n \mid\!\}_X / \{\!\mid m, Y, x \mid\!\}_{\overline{X}})]_X$ CSent$(X, \{\!\mid X, m \mid\!\}_Y) \wedge$ Decrypts$(X, \{\!\mid m, Y, n \mid\!\}_X)$	(9)
9, **DEC2**	$(X\ Y)[(\nu m)\langle \{\!\mid X, m \mid\!\}_Y \rangle(\{\!\mid m, Y, n \mid\!\}_X)(\{\!\mid m, Y, n \mid\!\}_X / \{\!\mid m, Y, x \mid\!\}_{\overline{X}})]_X$ CSent$(X, \{\!\mid X, m \mid\!\}_Y) \wedge$ Decrypts$(X, \{\!\mid m, Y, n \mid\!\}_X) \wedge$ Knows(X, n)	(10)
5, 10, **P1**	$(X\ Y)[(\nu m)\langle \{\!\mid X, m \mid\!\}_Y \rangle(\{\!\mid m, Y, n \mid\!\}_X)$ $(\{\!\mid m, Y, n \mid\!\}_X / \{\!\mid m, Y, x \mid\!\}_{\overline{X}})\langle \{\!\mid n \mid\!\}_Y \rangle]_X$ CSent$(X, \{\!\mid X, m \mid\!\}_Y) \wedge$ Decrypts$(X, \{\!\mid m, Y, n \mid\!\}_X) \wedge$ Knows$(X, n) \wedge$ Sent$(X, \{\!\mid n \mid\!\}_Y)$	(11)
11	$(X\ Y)[(\nu m)\langle \{\!\mid X, m \mid\!\}_Y \rangle(\{\!\mid m, Y, n \mid\!\}_X)$ $(\{\!\mid m, Y, n \mid\!\}_X / \{\!\mid m, Y, x \mid\!\}_{\overline{X}})\langle \{\!\mid n \mid\!\}_Y \rangle]_X$ CSent$(X, \{\!\mid X, m \mid\!\}_Y) \wedge$ Decrypts$(X, \{\!\mid m, Y, n \mid\!\}_X) \wedge$ CSent$(X, \{\!\mid n \mid\!\}_Y)$	(12)

for A in the responder role, so ϕ trivially holds in those cases. To prove ϕ for the initiator role, we have two cases. For the case where the initiator never receives a reply to his message, Decrypts$(A, \{\!\mid m, B, n \mid\!\}_A)$ is never true, so ϕ holds. For the case where the initiator receives a reply, we look at the $\mathcal{R}$(**Init**) deduction shown in Table 11, which shows that Decrypts$(A, \{\!\mid m, B, n \mid\!\}_A)$, CSent$(A, \{\!\mid a, m \mid\!\}_B)$, and CSent$(A, \{\!\mid n \mid\!\}_B)$ are all true, so ϕ holds. Since ϕ holds for any valid sequence of steps that an honest participant A would make, then Honest$(A) \supset \phi(A)$.

5.5. *Soundness*

Theorem 5.1 (Soundness). *If $\mathcal{Q} \vdash \phi$ then $\mathcal{Q} \models \phi$.*

The proof is an induction on the structure of proofs. Several cases are sketched in Section 5.1. The details are included in Appendix B.

6. Sample proof

We have constructed proofs for a few different protocols, including public-key and symmetric key protocols (with appropriate modification of the decryption action). In addition, our attempt to prove an important property that holds for the Needham–Schroeder–Lowe protocol, fails for the original Needham–Schroeder protocol in an insightful way, due to an inability of the responder to identify the principal who decrypted the second message with the principal who sent the first and third.

Tables 11 through 13 contain a formal proof of the property ϕ for the *NSL* protocol.

In an informal sense, Table 11 shows the initiator's understanding of what has happened at the end of a successful run of the protocol. The final formula of Table 11 is a formula $[P]_X\,\psi$ where P is the trace of actions of the initiator's role in the protocol, and ψ is a formula collecting together a set of the initiator's observations.

In the same informal sense, Table 12 shows what the responder can observe by the end of a run of the responder's role of the protocol.

Informally, Table 13 shows how for a specific run where Bob thinks he has talked to Alice, Bob can combine reasoning about the protocol roles with his own observations to conclude that if Alice is honest (i.e., the decryption key $\overline{A}$ is not known to the attacker, and Alice has followed her role under *NSL*) then he has communicated with Alice.

The property ϕ is an important correspondence property, but it does not prove the full correctness of *NSL*. The goal here is to demonstrate a simple proof of a property that is provable under our system for *NSL* but is not provable for *NS*.

Table 12

$\mathcal{R}$(**Resp**) – deductions from **Resp** role

AR1	$[(\{\!\vert Y,m\vert\!\}_X)]_X\ \exists Z.\mathsf{Sent}(Z,\{\!\vert Y,m\vert\!\}_X)$	(1)
AN3	$[(\nu n)]_X\ \mathsf{Source}(n,X,(m,X,n),Y)$	(2)
AR1	$[(\{\!\vert n\vert\!\}_X)]_X\,\exists Z.\mathsf{Sent}(Z,\{\!\vert n\vert\!\}_X)$	(3)
1, 2, **P6**	$(X)[(\nu n)(\{\!\vert Y,m\vert\!\}_X)]_X$ $\mathsf{Source}(n,X,(m,X,n),Y)\wedge\exists Z.\mathsf{Sent}(Z,\{\!\vert Y,m\vert\!\}_X)$	(4)
4, **P4**, **P8**	$(X)[(\nu n)(\{\!\vert Y,m\vert\!\}_X)(\{\!\vert Y,m\vert\!\}_X/\{\!\vert Y',x\vert\!\}_{\overline{X}})]_X$ $\mathsf{Source}(n,X,(m,X,n),Y)\wedge\exists Z.\mathsf{Sent}(Z,\{\!\vert Y,m\vert\!\}_X)$	(5)
5, **P1**, **S1**	$(X)[(\nu n)(\{\!\vert Y,m\vert\!\}_X)(\{\!\vert Y,m\vert\!\}_X/\{\!\vert Y',x\vert\!\}_{\overline{X}})\langle\{\!\vert m,X,n\vert\!\}_Y\rangle]_X$ $\mathsf{Source}(n,X,(m,X,n),Y)\wedge\exists Z.\mathsf{Sent}(Z,\{\!\vert Y,m\vert\!\}_X)$	(6)
3, 6, **P2**, **P6**	$(X)[(\nu n)(\{\!\vert Y,m\vert\!\}_X)(\{\!\vert Y,m\vert\!\}_X/\{\!\vert Y',x\vert\!\}_{\overline{X}})\langle\{\!\vert m,X,n\vert\!\}_Y\rangle(\{\!\vert n\vert\!\}_X)]_X$ $\mathsf{Source}(n,X,(m,X,n),Y)\wedge\exists Z.\mathsf{Sent}(Z,\{\!\vert Y,m\vert\!\}_X)\wedge$ $\exists Z'.\mathsf{Sent}(Z',\{\!\vert n\vert\!\}_X)$	(7)
7, **P4**, **P8**	$(X)[(\nu n)(\{\!\vert Y,m\vert\!\}_X)(\{\!\vert Y,m\vert\!\}_X/\{\!\vert Y',x\vert\!\}_{\overline{X}})$ $\langle\{\!\vert m,X,n\vert\!\}_Y\rangle(\{\!\vert n\vert\!\}_X)(\{\!\vert n\vert\!\}_X/\{\!\vert n\vert\!\}_{\overline{X}})]_X$ $\mathsf{Source}(n,X,(m,X,n),Y)\wedge\exists Z.\mathsf{Sent}(Z,\{\!\vert Y,m\vert\!\}_X)\wedge$ $\exists Z'.\mathsf{Sent}(Z',\{\!\vert n\vert\!\}_X)$	(8)

Table 13

B's completed deduction with honesty rule

$\mathcal{R}$(**Resp**), **G2**	$(B)[(\nu n)(\{\!\!\{A, m\}\!\!\}_B)(\{\!\!\{A, m\}\!\!\}_B / \{\!\!\{Y, x\}\!\!\}_{\overline{B}})$ $\langle \{\!\!\{m, B, n\}\!\!\}_A \rangle (\{\!\!\{n\}\!\!\}_B)(\{\!\!\{n\}\!\!\}_B / \{\!\!\{n\}\!\!\}_{\overline{B}})]_B$ $\mathsf{Source}(n, B, (m, B, n), A) \wedge$ $\exists X.(\mathsf{Sent}(X, \{\!\!\{n\}\!\!\}_B)$	(1)
$\mathcal{R}$(**Resp**)	$(B)[(\nu n)(\{\!\!\{A, m\}\!\!\}_B)\ldots]_B$ $\mathsf{Honest}(B) \supset \neg\mathsf{Sent}(B, \{\!\!\{n\}\!\!\}_B)$	(2)
1, 2, **G1**	$(B)[(\nu n)(\{\!\!\{A, m\}\!\!\}_B)\ldots]_B\ \mathsf{Source}(n, B, (m, B, n), A) \wedge$ $\exists X.(\mathsf{Sent}(X, \{\!\!\{n\}\!\!\}_B) \wedge (X \neq B))$	(3)
$\mathcal{R}$(**Init**), **HON**	$\mathsf{Honest}(X) \supset (\mathsf{Sent}(X, \{\!\!\{n\}\!\!\}_B) \supset$ $\mathsf{Knows}(X, n) \wedge \mathsf{Knows}(X, B))$	(4)
3, 4	$(B)[(\nu n)(\{\!\!\{A, m\}\!\!\}_B)\ldots]_B\ \mathsf{Source}(n, B, (m, B, n), A) \wedge$ $\exists X.\mathsf{Honest}(X) \supset$ $(\mathsf{Sent}(X, \{\!\!\{n\}\!\!\}_B) \wedge (X \neq B) \wedge \mathsf{Knows}(X, n))$	(5)
SRC	$\mathsf{Source}(n, B, (m, B, n), A) \wedge \exists X.(X \neq B) \wedge$ $\mathsf{Knows}(X, n) \supset \exists Y.\mathsf{Decrypts}(Y, \{\!\!\{m, B, n\}\!\!\}_A)$	(6)
SEC	$\mathsf{Honest}(A) \wedge \mathsf{Decrypts}(Y, \{\!\!\{m, B, n\}\!\!\}_A) \supset (Y = A)$	(7)
6, 7	$\mathsf{Honest}(A) \wedge \mathsf{Source}(n, B, (m, B, n), A) \wedge$ $\exists X.(X \neq B) \wedge \mathsf{Knows}(X, n) \supset$ $\mathsf{Decrypts}(A, \{\!\!\{m, B, n\}\!\!\}_A)$	(8)
5, 8, **G1 − 3**	$(B)[(\nu n)(\{\!\!\{A, m\}\!\!\}_B)\ldots]_B$ $\mathsf{Honest}(A) \supset \mathsf{Decrypts}(A, \{\!\!\{m, B, n\}\!\!\}_A)$	(9)
$\mathcal{R}$(**Init**), **HON**	$\mathsf{Honest}(A) \supset (\mathsf{Decrypts}(A, \{\!\!\{m, B, n\}\!\!\}_A) \supset$ $\mathsf{CSent}(A, \{\!\!\{A, m\}\!\!\}_B) \wedge \mathsf{CSent}(A, \{\!\!\{n\}\!\!\}_B))$	(10)
9, 10, **G2**	$(B)[(nun)(\{\!\!\{A, m\}\!\!\}_B)\ldots]_B\ \mathsf{Honest}(A) \supset$ $(\mathsf{CSent}(A, \{\!\!\{A, m\}\!\!\}_B) \wedge \mathsf{CSent}(A, \{\!\!\{n\}\!\!\}_B))$	(11)

6.1. Failure of the original Needham–Schroeder protocol

Note that this proof fails for the original Needham–Schroeder protocol, which omitted the principal name from the second message.

For the original *NS*, we need to prove

$$\phi'(A) = \forall m, n, B.\mathsf{Decrypts}(A, \{\!\!\{m, n\}\!\!\}_A) \supset$$
$$(\mathsf{CSent}(A, \{\!\!\{A, m\}\!\!\}_B) \wedge \mathsf{CSent}(A, \{\!\!\{n\}\!\!\}_B))$$

But when we try to apply the trying the honesty rule on ϕ', in the case where $\mathsf{Decrypts}(A, \{\!\!\{m, n\}\!\!\}_A)$ is true you can come up with a counterexample, for example: $(\mathsf{CSent}(A, \{\!\!\{A, m\}\!\!\}_E) \wedge \mathsf{CSent}(A, \{\!\!\{n\}\!\!\}_E))$.

It is possible to prove

$$\phi''(A) = \forall m, n.\exists X.\mathsf{Decrypts}(A, \{\!\!\{m, n\}\!\!\}_A) \supset$$

$$(\mathsf{CSent}(A, \{\!|A, m|\!\}_X) \wedge \mathsf{CSent}(A, \{\!|n|\!\}_X))$$

(i.e. that A's messages were sent to somebody, but not necessarily to B). This is exactly Lowe's observation about *NS* (where E is a principal whose private key is known to the intruder) [13]. B thinks he was talking to A, while A thinks she was talking to E.

7. Related work

The cord notation and proof system presented here is strongly related to the Floyd–Hoare style logic used for making before-after assertions about imperative programs [9,11]. As in Floyd–Hoare logic, we have axioms that correspond to each action in the calculus, though we use only post-conditions, not pre-conditions.

As explained in Section 2, our process calculus developed from an effort to refine the strand space formalism [8] by variables and substitution. Strand spaces seemed attractive as a variant of the natural language of "arrows and messages"; unfortunately, they did not support the logical annotations that we wanted to add to protocols. In concrete analyses, each strand would actually represent a family of strands, parameterized by the possible values of all the data that occurred in it. Communicating a value from one agent to another was then modelled as spontaneous instantiation of the two parameters in the two corresponding strands to the same value. We started by replacing the parameter in the receiving strand by a variable, and modelling communication as substitution of the sent value for that variable. This allowed us to capture syntactically the data known to an agent.[1] Of course, this brought us to the well ploughed ground of process calculi.

Representing communication as reaction of a send-action and a receive-action, with the substitution as the effect of their reaction, is clearly very much in the spirit of π-calculus [20] and chemical abstract machines [2]. There has been a substantial amount of work on developing process calculi of this kind for specific applications in security. In particular, the Spi-calculus [1] was developed as an extension of the π-calculus specifically designed for analyzing security by coinductive methods of process calculus, i.e., in terms of bisimilarity of processes.

Our idea was, however, to proceed in a different direction: towards logical semantics of protocols. We needed a process calculus as a rudimentary programming language, to support logical annotations, rather than to directly analyze protocols as processes. The result is the cord calculus. Its core, presented here, is simplified wherever possible: we dropped channels, reducing all communication to broadcasting; pattern-matching was elevated to a generic destructor operator, capturing both

[1] The parameter mechanism of strand spaces does not allow a formal distinction between two values in message sent to Bob, one visible to him, and another contained in a token encrypted by Alice's key, that he should forward to her.

value comparison, and decryption. The main purpose of such reductions are the resulting simplifications, that more clearly display the essential logical issues. More or less straightforward extensions of the cord calculus allow representing point-to-point communication, as well as the various cryptographic concepts not considered here.

There are a large number of "belief" type logics such as BAN and its descendants [3,10,27]. Our approach differs from these logics in the following essential ways: (1) There is no "idealization" step – the protocol actions are included directly in the modal operators; (2) There is a close association between the actions and the logical statements we make to reason about them; and (3) The honesty rule is used to prove invariants of the protocol.

Several of the concepts presented here have appeared in other work on security protocols. Our use of the ν operator to represent new data is borrowed from process calculus, but is also reminiscent of the use of the existential quantifier used for the same purpose in Multiset Rewriting [4]. Events are a common idea from process calculus and Hoare logic. In protocol analysis, they have also been used in the NRL Protocol Analyzer [15], though our notion of event here is more basic and corresponds exactly to the actions in the cord calculus. The Source predicate is used to capture the notion of data origination and casuality. This is similar to the notions of originating and uniquely originating data in strands [28].

In [26], a faithfulness assumption and causality criterion are given for protocols, and it was shown that protocols failing to satisfy them were flawed. The "honesty rule" is a generalization and formalization of the faithfulness assumption. By combining the honesty rule with our proof system, we can reason about a protocol based on assumptions about the honesty of its participants.

8. Conclusion

We propose a specialized protocol logic that is built around a process language for communicating cords describing the actions of a protocol. The logic contains axioms and inference rules for each of the main protocol actions and proofs are protocol-directed, meaning that the outline of a proof of correctness follows the sequence of actions in the protocol.

A central idea is that assertions associated with an action will hold in any protocol execution that contains this action. This gives us the power to reason about all possible runs of a protocol, without explicitly reasoning about steps that might be carried out by an attacker. At the same time, the semantics of our logic is based on sets of traces of protocol execution (possibly including an attacker), not the kind of abstract idealization found in some previous logics. This approach lets us prove properties of protocols that hold in all runs, without any explicit reasoning about the potential actions of an intruder.

Acknowledgements

Thanks to the anonymous referees for their many helpful comments.

Appendix A. Summary of syntax

For convenience, we present a summary in Table 14 of the BNF syntax for cords and the protocol logic, as presented throughout the paper.

Appendix B. Soundness of axioms and proof rules

Here we prove Theorem 5.1, the soundness of the axioms and proof rules. Section B.1 proves the soundness of the axioms, Section B.2 proves the soundness of the relationships between the predicates, and Section B.3 proves the soundness of the proof rules.

First we prove some lemmas that will be useful for the proofs that follow. In the following lemmas, R indicates a run of any protocol.

Lemma B.1. *EVENT*$(R, X, \langle m\rangle, \emptyset, \emptyset) \supset \exists Y.$*EVENT*$(R, Y, (x), m, x)$ *and* *EVENT*$(R, X, (x), m, x) \supset \exists Y.$*EVENT*$(R, Y, \langle m\rangle, \emptyset, \emptyset)$

Proof. This follows from the definitions of the basic cord calculus reactions.

Given a procotol $\mathcal{Q}$ with run R, where S and S' are arbitrary actions, recall that *EVENT*$(R, X, \langle m\rangle, \emptyset, \emptyset)$ means that the reduction

$$([S\langle m\rangle S']_X \otimes C) \ \ \triangleright\triangleright\ \ ([SS']_X \otimes C')$$

occurs in run R. Refering to the list of basic reaction steps in Table 3, we see that this is a reaction of type (2). Plugging in that rule, we get

$$([S\langle m\rangle S']_X \otimes [T(x)T']_Y \otimes D) \ \ \triangleright\triangleright\ \ ([SS']_X \otimes [TT'(m/x)]_Y \otimes D)$$

with side condition $FV(m) = \emptyset$. Note that these are the same reduction, if $C = [T(x)T']_Y \otimes D$ and $C' = [TT'(m/x)]_Y \otimes D$. But by taking $E = ([S\langle m\rangle S']_X \otimes D$ and $E' = [SS']_X \otimes D$, this reduction can also be written as

$$([T(x)T']_Y \otimes E) \ \ \triangleright\triangleright\ \ ([TT'(m/x)]_Y \otimes E')$$

which from the definition of *EVENT* means *EVENT*$(R, Y, (x), m, x)$.

Similarly for the other direction. □

Table 14

Summary of syntax

(names)	$N ::= X$	variable name
	A	constant name
(basic keys)	$K_0 ::= k$	constant key
	y	variable key
	N	name
(keys)	$K ::= K_0$	basic key
	$inv(K_0)$	inverse key
(terms)	$t ::= x$	variable term
	c	constant term
	N	name
	K	key
	t, t	tuple of terms
	$\{\|t\|\}_K$	term encrypted with key K
(actions)	$a ::= \epsilon$	the null action
	$\langle t \rangle$	send a term t
	(x)	receive term into variable x
	(νx)	generate new term x
	$(t/q(x_1, \ldots, x_n))$	match term t to pattern q
(basic terms)	$b ::= x \mid c \mid N \mid K$	basic terms allowed in patterns
(basic patterns)	$p ::= b, \ldots, b$	tuple pattern
(patterns)	$q ::= p$	basic pattern
	$\{\|p\|\}_{\overline{K}}$	decryption pattern
(strands)	$S ::= aS \mid a$	
(processes)	$r ::= (x \ldots x)C\langle x \ldots x \rangle$	
(formulas)	$\phi ::= \mathsf{Sent}(N, t)$	
	$\mathsf{Knows}(N, t)$	
	$\mathsf{Source}(t, N, t, N)$	
	$\mathsf{Decrypts}(N, t)$	
	$\mathsf{Honest}(N)$	
	$\phi \wedge \phi$	
	$\neg\phi$	
	$\forall x.\phi$	
(modal forms)	$\Psi ::= \rho\ \phi$	

Lemma B.2. *If EVENT*$(R, X, \sigma\langle \{\|x\|\}_Y \rangle, \emptyset, \emptyset)$ *for some protocol* $\mathcal{Q}$*, where* $\sigma x = m$ *and* $\sigma Y = K$*, then* $\mathcal{Q}, R \models \mathsf{Knows}(X, K) \wedge \mathsf{Knows}(X, m)$.

Proof. This also follows from the definitions of the cord calculus – no term containing a free variable can be sent.

Let R' be the run containing all the events of run R up to, but not including, the $\langle \{\|m\|\}_K \rangle$ event.

Given a procotol $\mathcal{Q}$ with run R, where S and S' are arbitrary actions, recall that *EVENT*$(R, X, \langle \{\!|m|\!\}_K \rangle, \emptyset, \emptyset)$ means that the reduction

$$([S\langle \{\!|m|\!\}_K \rangle S']_X \otimes C) \quad \triangleright\triangleright \quad ([SS']_X \otimes C')$$

occurs in run R. Refering to the list of basic reaction steps in Table 3, we see that this is a reaction of type (2). Plugging in that rule, we get

$$([S\langle \{\!|m|\!\}_K \rangle S']_X \otimes [T(x)T']_Y \otimes D) \quad \triangleright\triangleright \quad ([SS']_X \otimes [TT'(\{\!|m|\!\}_K/x)]_Y \otimes D)$$

with side condition $FV(\{\!|m|\!\}_K) = \emptyset$. In order for this reaction to occur, we have the condition that $\{\!|m|\!\}_K$ can not contain free variables at the time of the reaction. That means one of the following must be true in the run R' for K

1. $(K) \in R'|_X$, i.e., *EVENT*$(R', X, (x), K, x)$
2. $(\nu K) \in R'|_X$, i.e., *EVENT*$(R', X, (\nu x), K, x)$
3. $K \in FV(R'|_X)$ (i.e., K appears in the static interface to the role)
4. $(p(t)/p(x)) \in R|_X$, for $K \subseteq t$ i.e., *EVENT*$(R, X, (p(t)/p(x)), t, x)$
5. $(\{\!|p(t)|\!\}_{K'}/\{\!|p(x)|\!\}_{\overline{K'}}) \in R|_X$ for $K \subseteq t$,
   i.e., *EVENT*$(R, X, (\{\!|p(t)|\!\}_{K'}/\{\!|p(x)|\!\}_{\overline{K'}}), t, x)$

In the first three cases, the semantics of Knows gives us $\mathsf{Know}_0(X, K)$, so $\mathsf{Knows}(X, K)$. In the fourth case, K occurred in some pattern match action in R', which means $\mathsf{Know}_i(X, p(K))$, so $\mathsf{Know}_{i+1}(X, K)$ and therefore $\mathsf{Knows}(X, K)$. In the last case, K occurred in some decryption action in R', which means $\mathsf{Know}_i(X, \{\!|p(t)|\!\}_{K'})$, so $\mathsf{Know}_{i+1}(X, K)$ and therefore $\mathsf{Knows}(X, K)$. If m is atomic (contains no subterms), then the same argument holds for m.

If m is not atomic, then m must be built by some constructors in the cord calculus. The possibilities are as follows

1. $m = m', m''$
2. $m = \{\!|m'|\!\}_{m''}$

In both of the above cases, m' and m'' can contain no free variables, so if m' and m'' are atomic, we have $\mathsf{Know}_i(X, m')$ and $\mathsf{Know}_i(X, m'')$ from the argument above. If m' or m'' is not atomic, then they must have been constructed and again they can contain no free variables, so we repeat the argument for each component until all components are atomic.

Therefore, $\mathsf{Knows}(X, m)$ and $\mathsf{Knows}(X, K)$, so the lemma is proved. □

B.1. Axioms

In the following subsections we prove the soundness of the axioms from Table 7. Each section starts with an informal statement of the axiom, and then proves its soundness.

B.1.1. **AN1** $[(\nu m)]_X\ \mathsf{Knows}(Y, m) \supset (Y = X)$

Informally, Axiom **AN1** says that if a principal X generates a new value m and takes no further actions, then the only principal who knows m is X.

By definition,

$$\mathcal{Q}, R \models [(\nu m)]_X\ \mathsf{Knows}(Y, m) \supset (Y = X)$$

if $\mathcal{Q}, R \models \sigma(\mathsf{Knows}(Y, m) \supset (Y = X))$ whenever (νm) matches $R|_X$, where σ is a matching substitution. By definition of *matches*, (νm) matches $R|_X$ with $\sigma(x) = m$ only if (νx) is the last action of one of the roles of X in R. But since this is the last action from that role in R, X cannot have sent m to any other principal, and from the side conditions for the ν reaction in Table 3 we know that m does not occur elsewhere in the cordspace. Since no events of form *EVENT*$(R, X, \langle m' \rangle, \emptyset, \emptyset)$, where $m \subseteq m'$, have occurred, we know from Lemma B.1 that there can be no events *EVENT*$(R, Y, (x), m', x)$. So from the semantics of Knows, $\mathsf{Knows}(Y, m)$ can't be true for any principal besides X. Therefore, from the semantics of Knows, only X knows m. This shows that axiom AN1 is sound.

B.1.2. **AN2** $[(\nu m)]_X\ \mathsf{Knows}(X, m)$

Informally, Axiom **AN2** says that if a principal X generates a new value m and takes no further actions, then after that action the principal X knows m.

By definition,

$$\mathcal{Q}, R \models [(\nu m)]_X\ \mathsf{Knows}(X, m)$$

if $\mathcal{Q}, R \models \sigma(\mathsf{Knows}(X, m))$ whenever (νm) matches $R|_X$, where σ is a matching substitution. By definition of *matches*, (νm) matches $R|_X$ with $\sigma(x) = m$ only if (νx) is the last action of one of the roles of X in R.

In this case, we have *EVENT*$(R, X, (\nu x), m, x)$, which from the semantics for Knows, means $\mathsf{Know}_0(X, m)$.

B.1.3. **AN3** $[(\nu m)]_X\ \forall t.\forall K.\mathsf{Source}(m, X, t, K)$

Informally, Axiom **AN3** says that if a principal X generates a new value m and takes no further actions, then after that action any encrypted message could be the source from which a principal other than X could have learned m. This is a vacuous statement, since Axiom **AN1** tells us that only X could know m at this point anyway.

By definition,

$$\mathcal{Q}, R \models [(\nu m)]_X\ \forall t.\forall K.\mathsf{Source}(m, X, t, K)$$

if $\mathcal{Q}, R \models \sigma(\forall t.\forall K.\mathsf{Source}(m, X, t, K))$ whenever (νm) matches $R|_X$, where σ is a matching substitution. By definition of *matches*, (νm) matches $R|_X$ with $\sigma(x) = m$ only if (νx) is the last action of one of the roles of X in R.

In this case we have *EVENT*$(R, X, (\nu x), m, x)$, and we know from Axiom **AN1** that $\mathsf{Knows}(Y, m) \supset X = B$, so from the semantics of Source we have a false antecedent, so we can prove $\mathsf{Source}(m, X, t, K)$ for any term t and key K.

B.1.4. **AS1** $[\langle m\rangle]_X$ Sent(X, m)

Informally, Axiom **AS1** says that if a principal X sends a message m, then the principal has sent m.

By definition,

$$\mathcal{Q}, R \models [\langle m\rangle]_X \ \mathsf{Sent}(X, m)$$

if $\mathcal{Q}, R \models \sigma(\mathsf{Sent}(X, m))$ whenever $\langle m\rangle$ matches $R|_X$, where σ is a matching substitution. By definition of *matches*, $\langle m\rangle$ matches $R|_X$ only if $\langle x\rangle$ with $\sigma(x) = m$ is the last action of one of the roles of X in R.

In that case we have $EVENT(R, X, \sigma\langle x\rangle, \emptyset, \emptyset) = EVENT(R, X, \langle m\rangle, \emptyset, \emptyset)$. So, from the semantics of Sent, this means Sent(X, m).

B.1.5. **AR1** $[(m)]_X\ \exists Y.$Sent(Y, m)

Informally, Axiom **AR1** says that if a principal X receives a message m, then that message was sent by some principal Y.

By definition,

$$\mathcal{Q}, R \models [(m)]_X \ \exists Y.\mathsf{Sent}(Y, m)$$

if $\mathcal{Q}, R \models \sigma(\exists Y.\mathsf{Sent}(Y, m))$ whenever (m) matches $R|_X$, where σ is a matching substitution. By definition of *matches*, (m) matches $R|_X$ only if (x) with $\sigma(x) = m$ is the last action of one of the roles of X in R.

In this case, we have $EVENT(R, X, (x), m, x)$. From Lemma B.1, we know that $EVENT(R, X, (x), m, x) \supset \exists Y.EVENT(R, Y, \langle m\rangle, \emptyset, \emptyset)$. From the semantics of Sent, this means $\exists Y.$Sent(Y, m).

B.1.6. **AR2** $[(m)]_X$ Knows(X, m)

Informally, Axiom **AR2** says that if a principal X receives a message m, then the principal X knows m.

By definition,

$$\mathcal{Q}, R \models [(m)]_X \ \mathsf{Knows}(X, m)$$

if $\mathcal{Q}, R \models \sigma(\mathsf{Knows}(X, m))$ whenever (m) matches $R|_X$, where σ is a matching substitution. By definition of *matches*, (m) matches $R|_X$ only if (x) with $\sigma(x) = m$ is the last action of one of the roles of X in R.

In this case, we have $EVENT(R, X, (x), m, x)$, which from the semantics for Knows, means $\mathsf{Know}_0(X, m)$.

B.1.7. **AM1** $(x_1 \dots x_n)[\,]_X$ Knows$(X, x_1) \wedge \dots \wedge$ Knows(X, x_n)

Informally, Axiom **AM1** says a principal X always knows its own static parameters.

By definition,

$$\mathcal{Q}, R \models (x_1 \ldots x_n)[\ ]_X\ \mathsf{Knows}(X, x_1) \wedge \ldots \wedge \mathsf{Knows}(X, x_n)$$

if $\mathcal{Q}, R \models \sigma(\mathsf{Knows}(X, x_i))$ whenever $x_i \in FV(R|_X)$, where σ is a matching substitution. Since the x_i are static parameters of the cord, the substitution for x_i is the substitution cord σ_C of the initial configuration of the run. That is, if $\sigma_{\mathbf{C}}(x_i) = A$, then $\mathsf{Knows}(A, m)$.

From the semantics of Knows, this means $\mathsf{Know}_0(X, x_i)$.

B.1.8. **AD1** $[(\{\!|m|\!\}_K / \{\!|y|\!\}_{\overline{K}})]_X\ \mathsf{Decrypts}(X, \{\!|m|\!\}_K)$

Informally, Axiom **AD1** says that a principal X has decrypted the message $\{\!|m|\!\}_K$, if it knows a message $\{\!|m|\!\}_K$ and successfully pattern matched it against $\{\!|y|\!\}_{\overline{K}}$.

By definition,

$$\mathcal{Q}, R \models [(\{\!|m|\!\}_K / \{\!|y|\!\}_{\overline{K}})]_X \mathsf{Decrypts}(X, \{\!|m|\!\}_K)$$

if $\mathcal{Q}, R \models \sigma(\mathsf{Decrypts}(X, \{\!|m|\!\}_K))$ whenever $(\{\!|m|\!\}_K / \{\!|y|\!\}_{\overline{K}})$ matches $R|_X$, where σ is a matching substitution.

By definition of *matches*, $(\{\!|m|\!\}_K / \{\!|y|\!\}_{\overline{K}})$ matches $R|_X$ only if $(\{\!|x|\!\}_K / \{\!|y|\!\}_{\overline{K}})$ with $\sigma(x) = m$ is the last action of one of the roles of X in R.

In that case we have *EVENT*$(R, X, (\{\!|m|\!\}_K / \{\!|y|\!\}_{\overline{K}}), m, y)$, and in order for the pattern match action to take place, $FV(\{\!|m|\!\}_K) = \emptyset$, which means $\mathsf{Knows}(X, \{\!|m|\!\}_K)$. From the semantics of Decrypts, this means $\mathsf{Decrypts}(X, \{\!|m|\!\}_K)$.

B.2. Relationships between predicates

B.2.1. **DEC1** $\mathsf{Decrypts}(X, \{\!|m|\!\}_K) \supset \mathsf{Knows}(X, \{\!|m|\!\}_K)$

Informally, **DEC1** says that if a principal X decrypts a message $\{\!|m|\!\}_K$, then X knows the message $\{\!|m|\!\}_K$.

From the semantics of Decrypts, we have *EVENT*$(R, X, (\{\!|m|\!\}_K / \{\!|x|\!\}_{\overline{K}}), m, x)$ and $\mathsf{Knows}(X, \{\!|m|\!\}_K)$, so this follows trivially.

B.2.2. **DEC2** $\mathsf{Decrypts}(X, \{\!|m|\!\}_K) \supset \mathsf{Knows}(X, m)$

Informally, **DEC2** says that if a principal X decrypts a message $\{\!|m|\!\}_K$, then X knows the message m.

From the semantics of Decrypts, we have *EVENT*$(R, X, (\{\!|m|\!\}_K / \{\!|x|\!\}_{\overline{K}}), m, x)$ and $\mathsf{Knows}(X, \{\!|m|\!\}_K)$. From the decryption pattern match case of the semantics of Knows, with $p(m) = m$, that means $\mathsf{Knows}(X, m)$.

B.2.3. **SEC** $\mathsf{Honest}(X) \wedge \mathsf{Decrypts}(Y, \{\{\!|m|\!\}_X\}) \supset (Y = X)$

Informally, **SEC** says that if a principal X is honest, and some principal Y has decrypted a message $\{\!|m|\!\}_X$ (i.e., a message encrypted with X's public key), then Y must be X. In other words, if X is honest, then nobody but X knows $\overline{X}$ (X's private key).

This axiom follows from Lemma 3.4, which says for any configuration **C** of protocol $\mathcal{Q}$, and any run R,

$$\mathit{EVENT}(R, X, (\{\!|t|\!\}_Y / \{\!|x|\!\}_{\overline{Y}}), t, x) \wedge X \in \mathit{HONEST}(\mathbf{C}) \supset X = Y.$$

Since $\mathsf{Honest}(X)$ means $X \in \mathit{HONEST}(\mathbf{C})$, and $\mathit{EVENT}(R, X, (\{\!|t|\!\}_Y / \{\!|x|\!\}_{\overline{Y}}), t, x)$ means $\mathsf{Decrypts}(X, \{\!|t|\!\}_Y)$, this follows directly.

B.2.4. **SRC** $\mathsf{Source}(m, X, t, K) \wedge \mathsf{Knows}(Z, m) \wedge Z \neq X \supset \exists Y.\mathsf{Decrypts}(Y, \{\!|t|\!\}_K)$

Informally, **SRC** says that if the only source of message m is the message $\{\!|t|\!\}_K$, then if anybody (besides principal X who created m) knows m, then somebody must have decrypted the message $\{\!|t|\!\}_K$.

From the semantics of $\mathsf{Source}(m, X, t, K)$ we have

$$\forall Z.(Z \neq X \wedge \mathsf{Knows}(Z, m)) \supset$$
$$\mathsf{Decrypts}(Z, \{\!|t|\!\}_K) \vee (\exists Y.\mathsf{Decrypts}(Y, \{\!|t|\!\}_K) \wedge \mathsf{Sent}(Y, t'))$$

And since $\mathsf{Knows}(Z, m) \wedge Z \neq X$, that means the antecedent is statisfied, so $\exists Y.\mathsf{Decrypts}(Y, \{\!|t|\!\}_K)$.

B.2.5. **K1** $\mathsf{Knows}(X, x, y) \supset \mathsf{Knows}(X, y) \wedge \mathsf{Knows}(X, x)$

Informally, **K1** says that if a principal X knows the tuple x, y, then the principal knows x and y.

This follows from the semantics of Knows. If $\mathsf{Knows}(X, x, y)$, then $\mathsf{Know}_i(X, x, y)$ for some i, and therefore $\mathsf{Know}_{i+1}(X, x)$ and $\mathsf{Know}_{i+1}(X, y)$.

B.3. Proof rules

B.3.1. **S1** – *The* Source *rule*

$$\mathbf{S1}\frac{[P]_X \mathsf{Source}(m, X, t, K)}{[P\langle\{\!|t|\!\}_K\rangle]_X \mathsf{Source}(m, X, t, K)} m \subseteq t$$

Informally, Axiom **S1** says that when principal X sends a message containing m, that message becomes the possible source from which another principal might obtain m.

Let R' be the run in the premise, and R be the continuation of R' in the conclusion. First note that the premise and the semantics of Source give us $\mathit{EVENT}(R', B, (\nu x), m, x)$ for P matching $R'|_X$, so we know $\mathit{EVENT}(R, B, (\nu x), m, x)$ for $P\langle\{\!|t|\!\}_K\rangle$ matching $R|_X$.

Assume $\mathsf{Knows}(X, m) \wedge X \neq B$.

We know from the semantics of Knows that $\mathsf{Knows}(X, m)$ can be true in a run R in the following circumstances:

1. $(\nu m) \in R|_X$, i.e., $EVENT(R, X, (\nu x), m, x)$
2. $m \in FV(R|_X)$ (i.e., m appears in the static interface to the role)
3. $(m) \in R|_X$, i.e., $EVENT(R, X, (x), m, x)$
4. $(p(t')/p(x)) \in R|_X$, for $m \subseteq t'$ i.e., $EVENT(R, X, (p(t')/p(x)), t', x)$
5. $(\{\!|p(t')|\!\}_{K'}/\{\!|p(x)|\!\}_{\overline{K'}}) \in R|_X$ for $m \subseteq t'$,
   i.e., $EVENT(R, X, (\{\!|p(t')|\!\}_{K'}/\{\!|p(x)|\!\}_{\overline{K'}}), t', x)$

If we assume $X \neq B$, then the first possibility is ruled out, because we know $EVENT(R, B, (\nu x), m, x)$, which means from reaction rule (5) that m couldn't appear anywhere else in the cordspace at the time of the (νx) action. If $EVENT(R, B, (\nu x), m, x)$ occured in run R first, then $EVENT(R, X, (\nu x), m, x)$ for $X \neq B$ could not occur later, and vice versa.

Similarly, the second possibility is ruled out, because again we know $EVENT(R, B, (\nu x), m, x)$, which means from reaction rule (5) that m couldn't appear anywhere else in the cordspace, in particular not in the static interface of any of the roles.

For the last case, if $t = t'$ and $K = K'$, then $\mathsf{Decrypts}(X, \{\!|t|\!\}_K)$, and we are done.

Otherwise, X received either m directly, or in some pattern containing m. But in order for $EVENT(R, X, (x), t', x)$ with $m \subseteq t'$, we know from Lemma B.1 that $\exists Y.EVENT(R, Y, \langle t' \rangle, \emptyset, \emptyset)$, and we know from Lemma B.2 that all values in the term t', in particular m, must have been known. So in order for $\mathsf{Knows}(Y, m)$, either $\mathsf{Decrypts}(Y, \{\!|t|\!\}_K)$ or $EVENT(R, Y, (x), t'', x)$ with $m \subseteq t''$. We repeat this argument until we find the Y that actually decrypted $\{\!|t|\!\}_K$.

Thus we have

$$
\begin{aligned}
&EVENT(R, B, (\nu x), m, x) \wedge \\
&\quad ((X \neq B \wedge \mathsf{Knows}(X, m)) \supset \mathsf{Decrypts}(X, \{\!|t|\!\}_K) \vee \\
&\quad (\exists Y.\mathsf{Decrypts}(Y, \{\!|t|\!\}_K) \wedge \mathsf{Sent}(Y, t'))) \text{where } m \subseteq t'
\end{aligned}
$$

which from the semantics of Source means $\mathsf{Source}(m, B, m', K)$.

B.3.2. **HON** – *the honesty rule*

For protocol $\mathcal{Q}$, the honesty rule is

$$
\frac{\forall \rho \in \mathcal{Q}.\forall P \subseteq \rho.\mathcal{Q} \vdash (\vec{Z})[P]_X\ \phi}{\mathcal{Q} \vdash \mathsf{Honest}(X) \supset \phi}\,\textbf{HON} \qquad \begin{array}{l}\text{no free variable in } \phi \\ \text{except } X \text{ bound in} \\ (\vec{Z})[P]_X\end{array}
$$

Where $(\vec{Z})[P]_X$ is the initial steps P of role ρ with static variables $(\vec{Z})$, and no free variable in ϕ other than X is bound by $(\vec{Z})[P]_X$. Assume $\mathcal{Q} \vdash (\vec{Z})[P]_X\ \phi$ for all $P \subseteq \rho, \rho \in Q$. By soundness for shorter proofs, we assume $\mathcal{Q}, R \models (\vec{Z})[P]_X\ \phi$ for all runs R of protocol $\mathcal{Q}$.

We must show that $\mathcal{Q}, R \models \mathsf{Honest}(X) \supset \phi$, for any run R. Let σ be any substitution such that $\sigma(\mathsf{Honest}(X) \supset \phi)$ has no free variables, and let $A = \sigma(X)$. Then we must show that if $\mathcal{Q}, R \models \mathsf{Honest}(A)$ then $\mathcal{Q}, R \models \sigma\phi$.

Assume $\mathcal{Q}, R \models \mathsf{Honest}(A)$. Then from the semantics of Honest and Lemma 3.1, $R|_A$ is an interleaving of traces of roles of $\mathcal{Q}$ carried out by A. Consider any standard trace in $R|_A$. By definition of honesty, this trace is matched by some prefix P of a role ρ in $\mathcal{Q}$.

Let τ be the matching substitution. Because $\mathcal{Q}, R \models (\vec{Z})[P]_A\ \sigma\phi$, and P matches $R|_A$ by τ, we conclude that $\mathcal{Q}, R \models \tau(\sigma\phi)$. But since $(\vec{Z})[P]_X$ does not bind any variables in ϕ except X, $\tau(\sigma\phi)$ is just $\sigma\phi$, and therefore $\mathcal{Q}, R \models \sigma\phi$.

B.3.3. Generic rules

G1 follows from the semantics of "$\wedge$", i.e.,

- $\mathcal{Q}, R \models (\phi_1 \wedge \phi_2)$ if $\mathcal{Q}, R \models \phi_1$ and $\mathcal{Q}, R \models \phi_2$
- $\mathcal{Q}, R \models [P]_X\phi$ if P matches $R|_X$ implies $\mathcal{Q}, R \models \phi$.

Similarly, **G2** follows from the meaning of the logical connectives.

G3 is valid because if ϕ is true after any run, then ϕ is true after a specific run that contains actions P.

B.3.4. Preservation rules

Since the semantics of Knows, Sent, and Decrypts are all based on the existence of a certain event in a run, adding additional events to the run cannot make these predicates false, so they are always preserved.

$\mathsf{Source}(m, X, t, k)$ is not preserved if additional messages other than $\{\!|t|\!\}_K$ are sent that contain the nonce m, because it is now possible for $\mathsf{Knows}(Y, m) \wedge X \neq Y$ to be true without the message $\{\!|t|\!\}_K$ ever being decrypted. So Source can become false if additional messages are sent that contain m.

References

[1] M. Abadi and A. Gordon, A calculus for cryptographic protocols: the spi calculus, *Information and Computation* **148**(1) (1999), 1–70. Expanded version available as SRC Research Report 149 (January 1998).

[2] G. Berry and G. Boudol, The chemical abstract machine, *Theoretical Computer Science* **96** (1992), 217–248.

[3] M. Burrows, M. Abadi and R. Needham, A logic of authentication, in: *Proceedings of the Royal Society, Series A* **426**(1871) (1989), 233–271. Also appeared as SRC Research Report 39 and, in a shortened form, in *ACM Transactions on Computer Systems* **8**(1) (1990), 18–36.

[4] I. Cervesato, N. Durgin, P. Lincoln, J. Mitchell and A. Scedrov, A meta-notation for protocol analysis, in: *12-th IEEE Computer Security Foundations Workshop*, P. Syverson, ed., IEEE Computer Society Press, 1999.

[5] F. Crazzolara and G. Winskel, Events in security protocols, in: *ACM Conference on Computer and Communications Security*, 2001, pp. 96–105.

[6] F. Crazzolara and G. Winskel, Composing strand spaces, in: *FST TCS 2002: Foundations of Software Technology and Theoretical Computer Science, 22nd Conference Kanpur, India*, M. Agrawal and A. Seth, eds, *Proceedings*, Volume 2556 of *Lecture Notes in Computer Science*, Springer, 2002.

[7] D. Dolev and A. Yao, On the security of public-key protocols, *IEEE Transactions on Information Theory* **2**(29) (1983).

[8] F.J.T. Fábrega, J.C. Herzog and J.D. Guttman, Strand spaces: Why is a security protocol correct? in: *Proceedings of the 1998 IEEE Symposium on Security and Privacy*, Oakland, CA, IEEE Computer Society Press, 1998, pp. 160–171.

[9] R.W. Floyd, Assigning meaning to programs, in: *Mathematical aspects of computer science: Proc. American Mathematics Soc. Symposia*, J.T. Schwartz, ed., Volume 19, Providence RI, American Mathematical Society, 1967, pp. 19–31.

[10] L. Gong, R. Needham and R. Yahalom, Reasoning about belief in cryptographic protocols, in: *Proceedings 1990 IEEE Symposium on Research in Security and Privacy*, D. Cooper and T. Lunt, eds, IEEE Computer Society, 1990, pp. 234–248.

[11] C.A.R. Hoare, An axiomatic basis for computer programming, *Communications of the ACM* **12**(10) (1969), 576–580.

[12] K.G. Larsen and R. Milner, A compositional protocol verification using relativized bisimulation, *Information and Computation* **99** (1992).

[13] G. Lowe, An attack on the Needham–Schroeder public-key protocol, *Info. Proc. Letters* **56** (1995), 131–133.

[14] G. Lowe, Breaking and fixing the Needham–Schroeder public-key protocol using CSP and FDR, in: *2nd International Workshop on Tools and Algorithms for the Construction and Analysis of Systems*, Springer-Verlag, 1996.

[15] C. Meadows, The NRL protocol analyzer: an overview, *J. Logic Programming* **26**(2) (1996), 113–131.

[16] R. Milner, Action structures, LFCS report ECS-LFCS-92-249, Department of Computer Science, University of Edinburgh, JCMB, The Kings Buildings, Mayfield Road, Edinburgh, December 1992.

[17] R. Milner, Action calculi and the pi-calculus, in: *NATO Summer School on Logic and Computation*, Marktoberdorf, 1993.

[18] R. Milner, Action calculi, or syntactic action structures, in: *Mathematical Foundations of Computer Science 1993, 18th International Symposium, MFCS '93*, Springer, 1993, pp. 105–121.

[19] R. Milner, *Communicating and Mobile Systems: The π-Calculus*, Cambridge University Press, Cambridge, UK, 1999.

[20] R. Milner, J. Parrow and D. Walker, A calculus of mobile processes, part i, *Information and Computation* **100**(1) (1992), 1–40.

[21] J. Mitchell, M. Mitchell and U. Stern, Automated analysis of cryptographic protocols using Murφ, in: *Proc. IEEE Symp. Security and Privacy*, 1997, pp. 141–151.

[22] R. Needham and M. Schroeder, Using encryption for authentication in large networks of computers, *Communications of the ACM* **21**(12) (1978), 993–999.

[23] L. Paulson, Proving properties of security protocols by induction, in: *10th IEEE Computer Security Foundations Workshop*, 1997, pp. 70–83.

[24] D. Pavlovic, Categorical logic of names and abstraction in action calculi, *Math. Structures in Comp. Sci.* **7**(6) (1997), 619–637.

[25] A.W. Roscoe, Modelling and verifying key-exchange protocols using CSP and FDR, in: *8th IEEE Computer Security Foundations Workshop*, IEEE Computer Soc. Press, 1995, pp. 98–107.

[26] P. Syverson, Adding time to a logic of authentication, in: *ACM Conference on Computer and Communications Security*, 1993, pp. 97–101.

[27] P. Syverson and P. van Oorschot, On unifying some cryptographic protocol logics, in: *Proc. 1994 IEEE Computer Security Foundations Workshop VII*, 1994, pp. 14–29.

[28] J. Thayer, J. Herzog and J. Guttman, Strand spaces: Proving security protocols correct, 1999, *Journal of Computer Security* **15** (1999).