A compositional logic for protocol correctness [*]

Nancy Durgin John Mitchell Dusko Pavlovic

Computer Science Dept.
Stanford University
Stanford, CA 94305-9045
{nad,jcm}@cs.stanford.edu

Kestrel Institute
Palo Alto, CA 94304
dusko@kestrel.edu

Abstract

We present a specialized protocol logic that is built around a process language for describing the actions of a protocol. In general terms, the relation between logic and protocol is like the relation between assertions in Floyd-Hoare logic and standard imperative programs. Like Floyd-Hoare logic, our 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. We prove that the protocol logic is sound, in a specific sense: each provable assertion about an action or sequence of actions holds in any run of the protocol, under attack, in which the given actions occur. This approach lets us prove properties of protocols that hold in all runs, while explicitly reasoning only about the sequence of actions needed to achieve this property. In particular, no explicit reasoning about the potential actions of an attacker is required.

1 Introduction

There has been considerable research on formal analysis of security protocols, ranging from BAN logic and related approaches [2, 6, 19] to finite-state analysis [18, 14] and proof methods based on higher-order logic [16]. 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. While automated trace-based tools can be used to find protocol errors within a few weeks work, it remains very 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 logic, we are able to prove correctness 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 Hoare logic for sequential imperative programs, 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 [2, 6, 19] and from explicit reasoning about protocol and intruder as in Paulson's inductive method [16]. 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 six predicates: Sent, $\mathsf{Created}$, $\mathsf{Decrypts}$, Knows, Source, and Honest. The first three 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. Our interpretation of Knows is also very mechanical, and much more elementary than in logics of knowledge. Specifically,

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

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 set of possible ways 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, as this arises in our model, 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 an honest principal. Using the honesty rule, it is possible to prove that if a principal A is honest, and A sends a message m of some form, then A must have previously received a message m' of some related form, for example. This form of reasoning allows us to prove that if one protocol participant completes the prescribed sequence of actions, and a principal named in one of the message is honest, then some secret is shared between the two principals.

Section 2 describes the process calculus used to define protocols and section 3 describes the formulas and semantics of our logic. The proof system is presented in section 4. A sample proof is discussed in section 5 (and in the appendix), with concluding remarks collected in section 6. The example in section 5 shows how we can prove correctness of Lowe's variant [8] of the Needham-Schroeder public key protocol [15]. A brief discussion in that section also shows how the same proof outline fails to produce a correct proof for the original Needham-Schroeder protocol, since Alice cannot correctly establish the identity of the responder unless the protocol is repaired according to Lowe's suggestion.

2 Communicating Cords

Cords are the formalism we use to represent protocols and their parts. They form an action calculus [9, 10, 17], based on π-calculus [13], and related to spi-calculus [1]. The basic idea of π-calculus is to represent communication by term reduction, so that the communication links can be created dynamically [12]. The idea of spi is to add to π the suitable constructors for encryption and decryption, and analyze secure communication in terms of bisimulations and process equivalences.[1] We treat the encryption in the same way, 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. As the present paper only addresses the latter, some features may not be apparent.

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 [8] of the Needham-Schroeder public key protocol [15], which we will refer to as NSL, might look something like Figure 1.

Strand spaces [5] 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 shows in Figure 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 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 gen-

[1] A notable earlier attempt at a similar approach was [7].

$$\begin{tikzcd}
A & \bullet \arrow[d, "\{\!|A,m|\!\}_B"] & \bullet \arrow[r, Rightarrow] & \bullet \arrow[d, "\{\!|n|\!\}_B"] \\
B & \bullet \arrow[r, Rightarrow] & \bullet \arrow[u, "\{\!|m,B,n|\!\}_A"] & \bullet
\end{tikzcd}$$

Figure 1. NSL as Arrows and Messages

$$\begin{tikzcd}
A = & +\{\!|A,m|\!\}_B \arrow[d] & -\{\!|m,B,n|\!\}_A \arrow[r, Rightarrow] & +\{\!|n|\!\}_B \arrow[d] \\
B = & -\{\!|A,m|\!\}_B \arrow[r, Rightarrow] & +\{\!|m,B,n|\!\}_A \arrow[u] & -\{\!|n|\!\}_B
\end{tikzcd}$$

Figure 2. NSL as a Strand Space

erally, 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 modeled by the substitution.

The cord space, corresponding to the above protocol is shown in Figure 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 $(x/\{\!|Y,z|\!\}_B)$ is now instantiated to $(\{\!|A,m|\!\}_B/\{\!|Y,z|\!\}_B)$. 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 $(u/\{\!|m,B,v|\!\}_A)$. 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 a. Moreover, the set of basic terms also contains the names N, which can be variable X, or constant A.

Upon these basic sets, the term language is then generated by some given constructors p, which always include tupling, and the public key encryption $\{\!|t|\!\}_N$, of the term t by the key named N. The decryption is not presented as a separate constructor.

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), and matching a term against a pattern $(t/p(x))$. In the present paper, we shall also consider nonce generation (νx) as a separate action, although other features of the calculus could cater for it. The extensions may allow other actions, such as reading time, or point-to-point communication.

$$\begin{tikzcd}
A = & {[} & (\nu m) & \langle\{\!|A,m|\!\}_B\rangle \arrow[dl] & & (u) & (u/\{\!|m,B,v|\!\}_A) & \langle\{\!|v|\!\}_B\rangle \arrow[dl] & {]} \\
B = & {[} & (x) & (x/\{\!|Y,z|\!\}_B) & (\nu n) & \langle\{\!|z,B,n|\!\}_Y\rangle \arrow[u] & (w) & (w/\{\!|n|\!\}_B) & {]}
\end{tikzcd}$$

Figure 3. NSL as a Cord Space

In summary, we have:

$$\begin{array}{llcl}
\text{(names)} & N & ::= & X \quad | \quad A \\
\text{(terms)} & t & ::= & x \quad | \quad a \quad | \quad N \quad | \\
 & & & t,\ldots,t \quad | \quad \{\!|t|\!\}_N \\
\text{(actions)} & a & ::= & \langle t\rangle \quad | \quad (x) \quad | \quad (\nu x) \quad | \quad (t/p(x))
\end{array}$$

Remark. For the sake of simplicity, this grammar is formalized as if only one variable at a time can be used to receive a message, and as if all constructors p must be unary. In a moment, when we introduce the reduction rules, it will become clear that receiving into an n-tuple of variables, and pattern matching with an n-ary constructor can be respectively implemented as

$$\begin{array}{rcl}
(x_1,\ldots,x_n) & = & (y)(y/x_1,\ldots,x_n) \\
(t/p(x_1,\ldots,x_n)) & = & (t/p(y))\,(y/x_1,\ldots,x_n)
\end{array}$$

2.2 Strands and cords

A strand is just a list of actions. The idea is that they should be the subsequent actions of a *single role* in a protocol. Since some actions of a role may be mutually completely independent, they can in principle be executed in any order: e.g., $\langle a\rangle$ and (x). Different strands can thus be semantically equivalent.

A cord is an equivalence class of behaviorally indistinguishable strands. The variable binding mechanism allows us to extract, in the basic calculus of cords, the minimal equivalence relation $\approx$. The more precise behavioral equivalence relations can be captured in the extended versions of the calculus. In any case, the list of syntactic categories now extends to

$$\begin{array}{llcl}
\text{(strands)} & S & ::= & aS \\
\text{(cords)} & C & ::= & S/_{\approx}
\end{array}$$

The second production here is something of an abuse of notation: cords are equivalence classes of strands, modulo a specific equivalence relation $\approx$ that allows renaming of bound variables and reordering of non-conflicting actions. For example, $\approx$ identifies strands like $(x)\langle a\rangle$ and $\langle a\rangle(x)$, 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. In general, the strands S and T will be considered independent if no values can be passed between them. Formally, we capture this by requiring that $\approx$ includes the relation $\sim$ defined

$$ST \sim TS \quad\Longleftrightarrow\quad \left\{\begin{array}{lcl} FV(S)\cap BV(T) & = & \emptyset \\ FV(T)\cap BV(S) & = & \emptyset \end{array}\right.$$

where the operators FV and BV, assigning to the strands, respectively, the sets of the free and of the bound variables, are inductively defined as follows:

$$\begin{array}{lcl}
FV\,(\langle t\rangle S) & = & FV(t)\cup FV(S) \\
FV\,((x)S) & = & FV(S)\setminus\{x\} \\
FV\,((t/p(x))S) & = & FV(t)\cup FV(S)\setminus\{x\} \\
FV\,((\nu x)S) & = & FV(S)\setminus\{x\} \\
 & & \\
BV\,(\langle t\rangle S) & = & BV(S) \\
BV\,((x)S) & = & BV(S)\cup\{x\} \\
BV\,((t/p(x))S) & = & BV(S)\cup\{x\} \\
BV\,((\nu x)S) & = & BV(S)\cup\{x\}
\end{array}$$

The set $FV(t)$ of the free variables occurring in a term t is defined as usually: 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") $(t/p(x))$ and "new" (νx) thus bind the variable x. The scope is always to the right, with the usual extrusion rules. 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 equivalence relation $\approx$ is generated as the transitive, reflexive closure of $\sim$, and then also closed under the α-conversion, *viz* renaming the bound variables. The strands S and T will thus be $\approx$-equivalent, and represent the same cord, 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*.

Remark about tupling again. Note that $(x)(y)$ is equivalent with $(y)(x)$, but very different from $(x, y) = (z)(z/x, y)$. The cord $[(x)(y)]$ will be used only when the messages to be received into x and y are truly unrelated, and can be received in any order.

2.3 Cord spaces and runs

A cord space is a multiset of cords. It is the space of all the roles that may be involved in a protocol, or in the chemical abstract machine speak, a "soup" in which the particles may react. For instance, one run of the NSL protocol arises from the cord space

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

With the binary operation $\otimes$ and the empty cord $[]$ as the unit, cord spaces form the free commutative monoid generated by cords.

The runs of a protocol are represented as reaction sequences in cord spaces. The basic reactions are shown in Table 1. The substitution (t/x) is assumed to act on the strand left of it, *viz* R'. As usually, it is assumed that no free variable becomes bound after substitution, which can always be achieved by renaming of the bound variables.

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 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.

The run of the NSL protocol, displayed in Figures 1-3 can now be completely formalized as a sequence of syntactic reaction steps, which are shown in Table 2. Steps (a), (c) and (e) are based on rule (1), steps (b), (d) and (f) on (2), and (d) and (f) also use rule (3). In a sense, these six reduction steps correspond to the five arrows in Figures 1 and 2, *plus* the final test of n, that B performs, which was omitted in the diagrams. On the other hand, the three arrows that appear in Figure 3 correspond to the applications of rule (1). The tests, based on rule (2), were represented in Figure 3 not by arrows, but by displaying the corresponding actions.

It is often said that the most ubiquitous errors in reasoning about security arise from hidden steps, and implicit assumptions that may or may not be satisfied. A solid foundation for protocol design and analysis should at least include a method for capturing and displaying all relevant steps, and their intended meanings. We believe that cord calculus caters for this need. While providing precision on one hand, it easily scales up, and extends by additional features, such as point-to-point communication, time stamps etc. While the increased precision and the added features necessarily extend and complicate notation, judicious choice of syntax has so far allowed us to keep the calculus, in all the extensions considered so far, as succinct as the competing informal notations. E.g., by introducing abbreviations, the code in the core calculus introduced above can be reduced to the size of the informal arrows-and-messages descriptions: the ubiquitous combination $(y)(y/p(x))$ can be abbreviated to $(p(x))$. For $p(x)$ a constant, this gives for receive-and-test

$$(m) \quad = \quad (x)(x/m)$$

Similarly, receiving and matching $p(x) = \{\!|x|\!\}_A$ becomes $(\{\!|x|\!\}_A)$.

2.4 Protocols

And finally, how do we represent a protocol? A protocol is a set of roles, such as initiator, responder and server, each representing the actions of a participant in a protocol session. Each cord captures one role, but a principal may consist of several cords, which form a cord space. To represent the initial state of a protocol, we form a cord space including the cords of all of its principals. To distinguish which cord belongs to which principal, we subscript them by their names. In the above example, each principal has just one role, but we shall in principle write

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

The protocol *attacker* is given by a set of actions, including receiving a message, decomposing into parts, decrypting 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 [15] and a model presented by Dolev and Yao [4]; see also [3].

A *run* of a protocol with attacker starts with selection of some number of honest principals and assignment of one or more roles to each honest principal. In addition, a number of compromised keys are generated and revealed to the attacker. Once this configuration is established, honest principals send and receive messages according to the roles assigned to them and the attacker may intercept and replace messages at will.

Since a principal may follow many roles, it could be possible for actions in one role to affect another role of the same principal. This gives rise to the following technical lemma, forbidding such "crosstalk", which is necessary for the soundness of the logic.

Lemma 2.1 (No Crosstalk) *If principal A follows role R, then all data sent be A as R is either generated by A as R or received by A as R.*

$$
\begin{array}{rcll}
[R(x)R'] \otimes [S\langle t\rangle S'] \ldots & \rhd_{(FV(t)=\emptyset)} \rhd & [RR'(t/x)] \otimes [SS'] \ldots & (1)\\
[R\,(p(t)/p(x))\,R'] \ldots & \rhd_{(FV(t)=\emptyset)} \rhd & [RR'(t/x)] \ldots & (2)\\
[R(\nu x)R'] \ldots & \rhd_{(x \notin FV(R'))} \rhd & [RR'] \ldots & (3)
\end{array}
$$

Table 1. Basic reaction steps

$$
\begin{array}{rcl}
A \otimes B & = & [(\nu m)\langle\{\!|A,m|\!\}_B\rangle(u)\ldots] \otimes [(x)(x/\{\!|Y,z|\!\}_B)\ldots]\\
& \rhd_{(a)} \rhd & [(\nu m)(u)\ldots] \otimes [(\{\!|A,m|\!\}_B/\{\!|Y,z|\!\}_B)(\nu n)\,\langle\{\!|z,B,n|\!\}_Y\rangle\ldots]\\
& \rhd_{(b)} \rhd & [(\nu m)(u)(u/\{\!|m,B,v|\!\}_A)\ldots] \otimes [(\nu n)\,\langle\{\!|m,B,n|\!\}_A\rangle\,(w)\ldots]\\
& \rhd_{(c)} \rhd & [(\nu m)\,(\{\!|m,B,n|\!\}_A/\{\!|m,B,v|\!\}_A)\,\langle\{\!|v|\!\}_B\rangle] \otimes [(\nu n)(w)\ldots]\\
& \rhd_{(d)} \rhd & [\langle\{\!|n|\!\}_B\rangle] \otimes [(\nu n)(w)(w/\{\!|n|\!\}_B)]\\
& \rhd_{(e)} \rhd & [] \otimes [(\nu n)(\{\!|n|\!\}_B/\{\!|n|\!\}_B)]\\
& \rhd_{(f)} \rhd & []
\end{array}
$$

Table 2. NSL example reaction

Note that this lemma normally holds true in real world scenarios – multiple processes running a protocol on the same machine don't generally share information. This is an assumption about the honest participants in a protocol, not about the attacker.

2.5 Static binding and cord category

Finally, as the reader familiar with action calculus may have noticed, cords, taken as particles, generate an action category [11, 17].[2] The idea is that a cord space C, displayed in the form

$$\gamma \quad = \quad (x_0 \ldots x_{i-1})C\langle y_0 \ldots y_{j-1}\rangle$$

can be viewed as an arrow $\gamma : i \longrightarrow j$, 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 occurences to the right. The variables $y_\bullet$ in the output interface may not be mutually different, nor different from $x_\bullet$. Of course, all expressions are up to α-conversion, i.e. up to variable renaming.

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

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

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

$$\gamma\,;\,\delta \quad = \quad (x_0 \ldots x_{i-1})\; CD(\vec{y}/\vec{u})\; \langle v_0 \ldots v_{k-1}\rangle$$

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

[2]Indeed, this is the source of the equivalence relation $\approx$.

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 role A above can be designed as the composite $A = A_0\,;\,A_1$, where

$$
\begin{array}{rcl}
A_0 & = & (X,Y)[(\nu m)\langle\{\!|X,m|\!\}_Y\rangle]\langle X,Y,m\rangle\\
A_1 & = & (\tilde{X},\tilde{Y},z)\left[(u)\left(u/\{\!|z,\tilde{Y},v|\!\}_{\tilde{X}}\right)\langle\{\!|v|\!\}_{\tilde{Y}}\rangle\right]\langle\rangle
\end{array}
$$

The constant values of A and B can be passed to it, at design time, by precomposing it with the morphism $()[]\langle A,B\rangle$.

Static binding plays, in the present paper, an essential role only in applications of rule **HON**. The details and the usage of the action calculus of cords must be left for a later occasion, but for the present paper we will display the interface only when it is relevant, and in other cases assume that all variables are bound.

3 A Protocol Logic

3.1 Syntax

The formulas of the logic are given by the following grammar:

$$
\begin{array}{rcl}
\phi & ::= & \mathsf{Sent}(t_1,t_2) \mid \mathsf{Knows}(t_1,t_2) \mid \mathsf{Source}(t_1,t_2,t_3)\\
& & \mid \mathsf{Created}(t_1,t_2) \mid \mathsf{Decrypts}(t_1,t_2) \mid \mathsf{Honest}(t_1)\\
& & \mid \phi \wedge \phi \mid \neg\phi \mid [P]_X\phi
\end{array}
$$

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:

$\mathsf{Knows}(Z, x)$: principal Z knows information x. This is "knows" in the precise 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 Z knows $\{|m|\}_K$ and does not know K^{-1}, then Z does *not* know m from this message.

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

$\mathsf{Decrypts}(Z, \{|m|\}_K)$ means principal Z received a message in a role that expected the message and was able to decrypt it. This implies that Z knows the message $\{|m|\}_K$, and the decrypted text, m.

$\mathsf{Created}(Z, m)$ means that principal Z created message m. This also means that Z knows some parts of the message m, particularly if it is an encrypted message $\{|m'|\}_K$, then Z also knows the encryption key K and the plaintext m'.

$\mathsf{Honest}(Z)$ means that the actions of principal Z in the current run are precisely an interleaving of the initial segments of a set of instantiated roles of the protocol. In other words, Z assumes some set of roles and does exactly the set of actions prescribed by them.

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

$$\mathsf{Source}(m, X, \mathbf{M})$$

means that the only way for a principal Y different from X to know m is if Y learned m from a message in the set $\mathbf{M}$, possibly by some indirect path.

3.2 Semantics

A formula may be true or false at a run of a protocol. More precisely, the main semantic relation, $Q, R \models \phi$, may be read, "formula ϕ holds at the end of run R of protocol 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 $Q, R \models \phi$, we make a few definitions regarding runs and the intruder. If Q is a protocol, then let

$$\bar{Q} = (Q \otimes I_Q)$$

where I_Q is the set of intruder actions for Q. (There is essentially only one set of intruder actions, enumerated in section 2.4. However, these actions must be adapted to the set of possible messages of Q.) Let $\mathsf{Runs}(\bar{Q})$ be the set of traces of protocol Q with intruder, as described in Section 2.4, each a sequence of states and actions.

A useful abbreviation is

$$\begin{aligned} &EVENT(R, B, P, \vec{n}, \vec{x}) \equiv \\ &\quad (([SPS']_B \otimes \dots) \;\; \triangleright\triangleright \;\; ([SS'(\vec{n}/\vec{x})]_B \otimes \dots) \in R) \end{aligned}$$

In words, $EVENT(R, B, P, \vec{n}, \vec{x})$ means that in run R, principal B executes actions P, receiving data $\vec{n}$ into variables $\vec{x}$. If P does not contain a receive action, then $\vec{n}$ and $\vec{x}$ will be empty.

Also we use $m \subseteq m'$ to indicate that m is a syntactic subterm of m' and $m \sqsubset m'$ to indicate that m is a syntactic subterm of m, not hidden by encryption. We use $\langle ., . \rangle$ to indicate pairing, i.e. $\langle x, y \rangle$ is the tuple containing x and y.

We define satisfaction $Q, R \models \phi$ by induction on ϕ as follows:

- $Q, R \models \mathsf{Sent}(B, m)$ if $EVENT(R, B, \langle m \rangle, \emptyset, \emptyset)$
- $Q, R \models \mathsf{Knows}(B, m)$ if $\exists i . \mathsf{Know}_i(B, m)$ where Know_j is defined by induction on j as follows:
  $(\mathsf{Know}_0(B, m)$ if $((m \in FV(R|_B)) \vee (m = B)$
  $\vee\, EVENT(R, B, (\nu m), \emptyset, \emptyset)$
  $\vee\, EVENT(R, B, (x), m, x)$
  and $\mathsf{Know}_{i+1}(B, m)$ if $\mathsf{Know}_i(B, m')$
  $\wedge\, (m' = m \vee m' = \langle m, m'' \rangle \vee m' = \langle m'', m \rangle$
  $\vee\, (m' = \{|m|\}_K$
  $\wedge\, EVENT(R, B, (m'/\{|y|\}_K), m, y))))$
- $Q, R \models \mathsf{Source}(m, B, \mathbf{M})$ if
  $EVENT(R, B, (\nu m), \emptyset, \emptyset) \wedge$
  $(\forall C . \forall m' . (EVENT(R, C, \langle m' \rangle, \emptyset, \emptyset) \wedge (m \subseteq m'))$
  $\supset (\exists m'' . m \subseteq m'' \subseteq m') \wedge m'' \in \mathbf{M})$
- $Q, R \models \mathsf{Created}(B, \{|m|\}_K)$ if
  $\mathsf{Knows}(B, K) \wedge \mathsf{Knows}(B, m)$
  $\wedge\, EVENT(R, B, \langle m' \rangle, \emptyset, \emptyset) \wedge (\{|m|\}_K \sqsubset m')$
- $Q, R \models \mathsf{Created}(B, \langle x, y \rangle)$ if
  $\mathsf{Created}(B, x) \vee \mathsf{Created}(B, y)$
- $Q, R \models \mathsf{Decrypts}(B, \{|m|\}_K)$ if
  $EVENT(R, B, (y)(y/\{|x|\}_K), m, x)$
- $Q, R \models \mathsf{Honest}(A)$ if $R|_A$ is an interleaving of instantiated roles of Q.
- $Q, R \models (\phi_1 \wedge \phi_2)$ if $Q, R \models \phi_1$ and $Q, R \models \phi_2$
- $Q, R \models \neg\phi$ if $Q, R \not\models \phi$
- $Q, R \models [P]_X\phi$ if P *matches* $R|_X$ implies $Q, R \models \phi$, where P matches $R|_X$ precisely if $R|_X$ is the interleaving of a set of initial segments of instantiated roles of Q and one of these instantiated roles is exactly P.

We write $Q \models \phi$ if $Q, R \models \phi$ for all $R \in \mathsf{Runs}(\bar{Q})$. Some consequences of the definitions are

$$\begin{array}{l} Q, R \models \mathsf{Honest}(A) \supset \phi \\ \quad \text{if } Q, R \models \mathsf{Honest}(A) \text{ implies } Q, R \models \phi \end{array}$$

where $\supset$ is regarded as an abbreviation for a combination of $\wedge$ and $\neg$, as usual, and

$$\begin{array}{l} Q \models [P]_X \phi \text{ if } \forall R \in \mathsf{Runs}(Q). \\ \quad (\text{if } P \text{ matches } R|_X \text{ then } Q, R \models \phi) \end{array}$$

4 Proof System

4.1 Axioms and rules about protocol actions

The axioms and inference rules about protocol actions are listed in Table 3. For the most part, these are relatively simple "translations" of actions into atomic formulas of the logic. For example, axiom **AN1** "says" that if principal X generates a new value m, then the only principal who knows m is X. By definition,

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

if $Q, R \models \mathsf{Knows}(Y, m) \supset (Y = X)$ whenever (νm) matches $R|_X$. By definition of *matches*, (νm) matches $R|_X$ only if (νm) is the last action of one of the roles of X in R. But in this case, X cannot have sent m to any other principal and therefore only X knows m. This shows that axiom **AN1** is sound.

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** states that if principal X generates a new value m and does no further actions in this role, then the set of messages that can contain m is empty.

Axiom **AM2** is about the binding of static variables of a protocol role. In this case the x in $(x \ \ldots)[]_X$, 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 m, then the set of messages that might allow another principal to obtain m is increased by one.

4.2 Axioms relating atomic predicates

Table 4 shows the relationships between the various properties, most of which follow naturally from the semantics outlined above. For example, if X decrypts $\{|m|\}_K$, then X knows m because that is the result of the decryption.

Axiom **SEC** is an assumption about protocols that we adopt in this paper for simplicity. This axiom says that principals do not reveal their private decryption keys. This is easily verified for many protocols by inspection: if no principal ever sends their key in any message, then it remains secret. While we could prove this formula for specific protocols, we assume this as an axiom and only consider protocols (in this paper) that obviously satisfy this formula. We repeat: axiom **SEC** is used here to shorten and simplify proofs; there is nothing inherent in this approach that requires us to restrict our attention to protocols that satisfy **SEC**.

Note that CSent is an abbreviation for "created and sent".

4.3 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 5.

The exception to preservation is Source. Since $\mathsf{Source}(m, X, \mathbf{M})$ means the only way for a principal other than X to know m is from messages in $\mathbf{M}$, this atomic formula may become false if X sends another message containing m. In this case, the relevant inference rule adds the message containing m to the set $\mathbf{M}$ of network messages containing m.

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 between the various instances if Alice is honest. This is the "No Crosstalk Lemma," Lemma 2.1.

4.4 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

AN1	$[(\nu m)]_X\ \mathsf{Knows}(Y, m) \supset (Y = X)$
AN2	$[(\nu m)]_X\ \mathsf{Knows}(X, m)$
AN3	$[(\nu m)]_X\ \mathsf{Source}(m, X, \{\ \})$
AS1	$[\langle m\rangle]_X\ \mathsf{Sent}(X, m)$
AR1	$[(\{\vert m\vert\}_K)]_X\ \exists Y.\mathsf{Created}(Y, \{\vert m\vert\}_K)$
AR2	$[(m)]_X\ \mathsf{Knows}(X, m)$
AM1	$[\ ]_X \mathsf{Knows}(X, \mathsf{key}(X))$
AM2	$(x \ldots)[\ ]_X\ \mathsf{Knows}(X, x)$
AM3	$[(y)(y/\{\vert m\vert\}_K)]_X \mathsf{Decrypts}(X, \{\vert m\vert\}_K)$
S1	$\dfrac{[P]_X \mathsf{Source}(m, Y, \mathbf{M})}{[P\langle m'\rangle]_X \mathsf{Source}(m, Y, \mathbf{M} \cup \{m'\})}\ \mathbf{m}\subseteq\mathbf{m'}$

Table 3. Axioms and rule for protocol actions

DEC1	$\mathsf{Decrypts}(X, \{\vert m\vert\}_K) \supset \mathsf{Knows}(X, \{\vert m\vert\}_K)$
DEC2	$\mathsf{Decrypts}(X, \{\vert m\vert\}_K) \supset \mathsf{Knows}(X, m)$
SEC	$\mathsf{Honest}(X) \wedge \mathsf{Decrypts}(Y, \{\{\vert m\vert\}_X\}) \supset (Y = X)$
SRC	$\mathsf{Source}(m, X, \{\{\vert m\vert\}_K\}) \wedge \mathsf{Knows}(Z, m) \wedge Z \neq X \supset$ $\exists Y.\mathsf{Decrypts}(Y, \{\vert m\vert\}_K)$
CR1	$\mathsf{Knows}(X, y) \wedge \mathsf{Knows}(X, K) \wedge \mathsf{Sent}(X, \{\vert y\vert\}_K) \supset$ $\mathsf{Created}(X, \{\vert y\vert\}_K)$
CR2	$\mathsf{Created}(X, \{\vert m\vert\}_K) \supset \mathsf{Knows}(X, m)$
CR3	$\mathsf{Created}(X, \{\vert m\vert\}_K) \supset \mathsf{Knows}(X, K)$
K1	$\mathsf{Knows}(X, \langle y, x\rangle) \supset \mathsf{Knows}(X, y) \wedge \mathsf{Knows}(X, x)$

$$\mathsf{CSent}(X, m) \quad \equiv \quad \mathsf{Created}(X, m) \wedge \mathsf{Sent}(X, m)$$

Table 4. Relationships between properties

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.

Generic Rules:

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

$$\frac{\phi}{[P]\phi}\ \mathbf{G3}$$

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

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

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

$$\frac{[P]_A\mathsf{Source}(m,Y,\mathbf{M})}{[P\langle m\rangle]_A\mathsf{Source}(m,Y,\mathbf{M})}\ (\mathbf{m}\not\subseteq\mathbf{N})\ \mathbf{P4} \qquad \frac{[P]_A\mathsf{Source}(m,Y,\mathbf{M})}{[P(x)]_A\mathsf{Source}(m,Y,\mathbf{M})}\ \mathbf{P5}$$

$$\frac{[P]_A\mathsf{Source}(m,Y,\mathbf{M})}{[P(\nu x)]_A\mathsf{Source}(m,Y,\mathbf{M})}\ \mathbf{P6}$$

Table 5. Inference Rules

$$\frac{\begin{array}{l}\mathcal{P} \vdash (X\ Y)[(\nu x)\langle\{\!|X,x|\!\}_Y\rangle]_X\ \phi(X)\\ \mathcal{P} \vdash (X\ Y)[(\nu x)\langle\{\!|X,x|\!\}_Y\rangle(\{\!|x,Y,y|\!\}_X)\langle\{\!|y|\!\}_Y\rangle]_X\ \phi(X)\\ \mathcal{P} \vdash (X)[\,]_X\ \phi(X)\\ \mathcal{P} \vdash (X)[(\{\!|Y,x|\!\}_X)(\nu y)\langle\{\!|x,X,y|\!\}_Y\rangle]_X\ \phi(X)\\ \mathcal{P} \vdash (X)[(\{\!|Y,x|\!\}_X)(\nu y)\langle\{\!|x,X,y|\!\}_Y\rangle(\{\!|y|\!\}_X)]_X\ \phi(X)\end{array}}{\mathcal{P} \vdash \mathsf{Honest}(X) \supset \phi(X)}\ \mathbf{HON}$$

Table 6. Honesty rule for NSL

Recall that a protocol $\mathcal{Q}$ is a set of roles, $\mathcal{Q} = \{R_1, R_2, \ldots, R_k\}$. Typically the roles may be initiator, responder and possibly a server, or the protocol may have some other form. If $R \in \mathcal{Q}$ is a role of protocol $\mathcal{Q}$, we write $P \subseteq R$ if P is an initial segment of the actions of role R such that the next action of R 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 if a role contains a send, for example, 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.

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.

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

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

Where $(\vec{Z})[P]_X$ is the initial steps P of role R 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 is shown in Table 6. 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 Appendix A. 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

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

holds for honest participant A. This formula is proved using the honesty rule in line 9 of B's deduction in Table 9.

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 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, $\mathsf{Decrypts}(A, \{\!|m, B, n|\!\}_A)$ is never true, so ϕ holds. For the case where the initiator receives a reply, we look at the $\mathcal{R}(A)$ deduction shown in Table 7, which shows that $\mathsf{Decrypts}(A, \{\!|m, B, n|\!\}_A)$, $\mathsf{CSent}(A, \{\!|a, m|\!\}_B)$, and $\mathsf{CSent}(A, \{\!|n|\!\}_B)$ are all true, so ϕ holds. Since ϕ holds for any valid sequence of steps that an honest participant A would make, then $\mathsf{Honest}(A) \supset \phi(A)$.

4.5 Soundness

Theorem 4.1 (Soundness) *If $Q \vdash \phi$ then $Q \models \phi$.*

The proof is an induction on the structure of proofs. Several cases are sketched in Section 4. An additional example, the proof of $\mathbf{AN3}$ is as follows:

Informally, Axiom $\mathbf{AN3}$ says that if a principal X generates a new value m and takes no further actions, then after that action the set of messages from which a principal other than X could have learned m is empty.

By definition,

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

if $Q, R \models \mathsf{Source}(m, X, \{\ \})$ whenever (νm) matches $R|_X$. By definition of *matches*, (νm) matches $R|_X$ only if (νm) is the last action of one of the roles of X in R. But in this case X cannot have sent any messages containing m.

From the semantics of Source, there can be no events of form

$$\begin{gathered}(EVENT(R, C, \langle m' \rangle, \emptyset, \emptyset) \wedge (m \subseteq m')) \supset \\ (\exists m''.m \subseteq m'' \subseteq m') \wedge m'' \in \mathbf{M})\end{gathered}$$

for any principal $C \neq X$, since we know X didn't send any messages containing m, and from $\mathbf{AN1}$ no other principal knows m, so the set $\mathbf{M}$ must be empty.

5 Sample correctness proof

We have constructed correctness proofs for a few different protocols, including public-key and private key protocols (with appropriate modification of the decryption action). In addition, our attempt to prove the original Needham-Schroeder protocol correct fails 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. Although we defer more elaborate discussion to the full paper, a proof for Lowe's variant of the Needham-Schroeder public-key protocol appears in three tables in Appendix A.

6 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.

References

[1] M. Abadi and A. Gordon. A calculus for cryptographic protocols: the spi calculus. *Information and Computation,*

148(1):1–70, 1999. Expanded version available as SRC Research Report 149 (January 1998).

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

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

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

[5] 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*, pages 160–171, Oakland, CA, May 1998. IEEE Computer Society Press.

[6] L. Gong, R. Needham, and R. Yahalom. Reasoning About Belief in Cryptographic Protocols. In D. Cooper and T. Lunt, editors, *Proceedings 1990 IEEE Symposium on Research in Security and Privacy*, pages 234–248. IEEE Computer Society, 1990.

[7] K. G. Larsen and R. Milner. A compositional protocol verifi - cation using relativized bisimulation. *Information and Computation*, 99, 1992.

[8] G. Lowe. Breaking and fi xing 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.

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

[10] R. Milner. Action calculi and the pi-calculus. In *NATO Summer School on Logic and Computation*, Marktoberdorf, November 1993.

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

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

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

[14] J. Mitchell, M. Mitchell, and U. Stern. Automated analysis of cryptographic protocols using Murφ. In *Proc. IEEE Symp. Security and Privacy*, pages 141–151, 1997.

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

[16] L. Paulson. Proving properties of security protocols by induction. In *10th IEEE Computer Security Foundations Workshop*, pages 70–83, 1997.

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

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

[19] P. Syverson and P. van Oorschot. On unifying some cryptographic protocol logics. In *Proc. 1994 IEEE Computer Security Foundations Workshop VII*, pages 14–29, 1994.

A Sample proof

Tables 7 through 9 contain a formal proof of the correctness of the Needham-Schroeder-Lowe protocol, for the responder, in our system.

In an informal sense, Table 7 shows Alice's understanding of what has happened at the end of a successful run of the protocol. The final formula of Table 7 is a formula $[P]_A\,\phi$ where P is the set of actions of the initiator's role in the protocol, and ϕ is a formula collecting together a set of Alice's observations.

In the same informal sense, Table 8 shows what Bob can observe by the end of a run of the responder's role of the protocol.

Informally, Table 9 shows how Bob can combine reasoning about Alice's role with his own observations to conclude that if Alice is honest (i.e., the decryption key A^{-1} is not known to the attacker, and Alice has followed her role under NSL) then he has communicated with Alice.

$\mathbf{AM2}, \mathbf{AM1}$	$(A\ B)[\ ]_A \mathsf{Knows}(A, A) \wedge \mathsf{Knows}(A, B)$	(1)
$\mathbf{AN2}, \mathbf{AN3}$	$[(\nu m)]_A \mathsf{Knows}(A, m) \wedge \mathsf{Source}(m, A, \{\ \})$	(2)
$\mathbf{AS1},$	$[\langle \{\!\|A, m\|\!\}_B \rangle]_A \mathsf{Sent}(A, \{\!\|A, m\|\!\}_B)$	(3)
$\mathbf{AR1}, \mathbf{AR2}$	$[(\{\!\|m, B, n\|\!\}_A)]_A \mathsf{Knows}(A, \{\!\|m, B, n\|\!\}_A) \wedge$ $\exists X.\mathsf{Created}(X, \{\!\|m, B, n\|\!\}_A)$	(4)
$\mathbf{AS1}$	$[\langle \{\!\|n\|\!\}_B \rangle]_A \mathsf{Sent}(A, \{\!\|n\|\!\}_B)$	(5)
$1, 2, \mathbf{P3}$	$(A\ B)[(\nu m)]_A \mathsf{Knows}(A, A) \wedge \mathsf{Knows}(A, B) \wedge$ $\mathsf{Knows}(A, m) \wedge \mathsf{Source}(m, A, \{\ \})$	(6)
$3, 6, \mathbf{P1}, \mathbf{S1}$	$(A\ B)[(\nu m)\langle \{\!\|A, m\|\!\}_B \rangle]_A \mathsf{Knows}(A, A) \wedge \mathsf{Knows}(A, B) \wedge$ $\mathsf{Knows}(A, m) \wedge \mathsf{Sent}(A, \{\!\|A, m\|\!\}_B) \wedge$ $\mathsf{Source}(m, A, \{\{\!\|A, m\|\!\}_B\})$	(7)
$7, \mathbf{CR1}$	$(A\ B)[(\nu m)\langle \{\!\|A, m\|\!\}_B \rangle]_A \mathsf{CSent}(A, \{\!\|A, m\|\!\}_B) \wedge$ $\mathsf{Source}(m, A, \{\{\!\|A, m\|\!\}_B\})$	(8)
$4, 8, \mathbf{P2}, \mathbf{P5}$	$(A\ B)[(\nu m)\langle \{\!\|A, m\|\!\}_B \rangle(\{\!\|m, B, n\|\!\}_A)]_A \mathsf{CSent}(A, \{\!\|A, m\|\!\}_B) \wedge$ $\mathsf{Source}(m, A, \{\{\!\|A, m\|\!\}_B\}) \wedge$ $\mathsf{Knows}(A, \{\!\|m, B, n\|\!\}_A) \wedge \exists X.\mathsf{Created}(X, \{\!\|m, B, n\|\!\}_A)$	(9)
$9, \mathbf{AM3}$	$(A\ B)[(\nu m)\langle \{\!\|A, m\|\!\}_B \rangle(\{\!\|m, B, n\|\!\}_A)]_A \mathsf{CSent}(A, \{\!\|A, m\|\!\}_B) \wedge$ $\mathsf{Source}(m, A, \{\{\!\|A, m\|\!\}_B\}) \wedge$ $\mathsf{Decrypts}(A, \{\!\|m, B, n\|\!\}_A) \wedge \exists X.\mathsf{Created}(X, \{\!\|m, B, n\|\!\}_A)$	(10)
$10, \mathbf{DEC2}$	$(A\ B)[(\nu m)\langle \{\!\|A, m\|\!\}_B \rangle(\{\!\|m, B, n\|\!\}_A)]_A \mathsf{CSent}(A, \{\!\|A, m\|\!\}_B) \wedge$ $\mathsf{Source}(m, A, \{\{\!\|A, m\|\!\}_B\}) \wedge \mathsf{Knows}(A, n) \wedge$ $\mathsf{Decrypts}(A, \{\!\|m, B, n\|\!\}_A) \wedge \exists X.\mathsf{Created}(X, \{\!\|m, B, n\|\!\}_A)$	(11)
$5, 11, \mathbf{P1}, \mathbf{P4}$	$(A\ B)[(\nu m)\langle \{\!\|A, m\|\!\}_B \rangle(\{\!\|m, B, n\|\!\}_A)\langle \{\!\|n\|\!\}_B \rangle]_A$ $\mathsf{CSent}(A, \{\!\|A, m\|\!\}_B) \wedge \mathsf{Source}(m, A, \{\{\!\|A, m\|\!\}_B\}) \wedge$ $\mathsf{Decrypts}(A, \{\!\|m, B, n\|\!\}_A) \wedge \exists X.\mathsf{Created}(X, \{\!\|m, B, n\|\!\}_A) \wedge$ $\mathsf{Knows}(A, n) \wedge \mathsf{Sent}(A, \{\!\|n\|\!\}_B)$	(12)
$12, \mathbf{CR1}$	$(A\ B)[(\nu m)\langle \{\!\|A, m\|\!\}_B \rangle(\{\!\|m, B, n\|\!\}_A)\langle \{\!\|n\|\!\}_B \rangle]_A$ $\mathsf{CSent}(A, \{\!\|A, m\|\!\}_B) \wedge \mathsf{CSent}(A, \{\!\|n\|\!\}_B) \wedge$ $\mathsf{Source}(m, A, \{\{\!\|A, m\|\!\}_B\}) \wedge$ $\mathsf{Decrypts}(A, \{\!\|m, B, n\|\!\}_A) \wedge \exists X.\mathsf{Created}(X, \{\!\|m, B, n\|\!\}_A)$	(13)

Table 7. $\mathcal{R}$(A) – A's view at end of run

$\mathbf{AR1}$	$(B)[(\{\!\|A, m\|\!\}_B)]_B \exists X.\mathsf{Created}(X, \{\!\|A, m\|\!\}_B)$	(1)
$\mathbf{AN3}$	$[(\nu n)]_B \mathsf{Source}(n, B, \{\ \})$	(2)
$2, \mathbf{S1}$	$[(\nu n)\langle\{\!\|m, B, n\|\!\}_A\rangle]_B \mathsf{Source}(n, B, \{\{\!\|m, B, n\|\!\}_A\})$	(3)
$\mathbf{AR1}$	$[(\{\!\|n\|\!\}_B)]_B \exists X.\mathsf{Created}(X, \{\!\|n\|\!\}_B)$	(4)
$1, 3, \mathbf{P1}, \mathbf{P3}$	$(B)[(\{\!\|A, m\|\!\}_B)(\nu n)\langle\{\!\|m, B, n\|\!\}_A\rangle]_B$ $\exists X.\mathsf{Created}(X, \{\!\|A, m\|\!\}_B) \wedge$ $\mathsf{Source}(n, B, \{\{\!\|m, B, n\|\!\}_A\})$	(5)
$4, 5, \mathbf{P2}, \mathbf{P5}$	$(B)[(\{\!\|A, m\|\!\}_B)(\nu n)\langle\{\!\|m, B, n\|\!\}_A\rangle(\{\!\|n\|\!\}_B)]_B$ $\exists X.\mathsf{Created}(X, \{\!\|A, m\|\!\}_B) \wedge$ $\mathsf{Source}(n, B, \{\{\!\|m, B, n\|\!\}_A\}) \wedge$ $\exists Y.(\mathsf{Created}(Y, \{\!\|n\|\!\}_B)$	(6)

Table 8. $\mathcal{R}$(B) – B's view at end of run

$\mathcal{R}(B), \mathbf{G2}$	$(B)[(\{\!\|A, m\|\!\}_B)(\nu n)\langle\{\!\|m, B, n\|\!\}_A\rangle(\{\!\|n\|\!\}_B)]_B$ $\mathsf{Source}(n, B, \{\{\!\|m, B, n\|\!\}_A\}) \wedge$ $\exists X.(\mathsf{Created}(X, \{\!\|n\|\!\}_B)$	(1)
$\mathcal{R}(B), \mathbf{HON}$	$(B)[(\{\!\|A, m\|\!\}_B)(\nu n)\langle\{\!\|m, B, n\|\!\}_A\rangle(\{\!\|n\|\!\}_B)]_B$ $\mathsf{Honest}(B) \supset \neg\mathsf{Created}(B, \{\!\|n\|\!\}_B)$	(2)
$1, 2, \mathbf{G1}$	$(B)[(\{\!\|A, m\|\!\}_B)(\nu n)\langle\{\!\|m, B, n\|\!\}_A\rangle(\{\!\|n\|\!\}_B)]_B$ $\mathsf{Source}(n, B, \{\{\!\|m, B, n\|\!\}_A\}) \wedge$ $\exists X.(\mathsf{Created}(X, \{\!\|n\|\!\}_B) \wedge (X \neq B))$	(3)
$3, \mathbf{CR2}$	$(B)[(\{\!\|A, m\|\!\}_B)(\nu n)\langle\{\!\|m, B, n\|\!\}_A\rangle(\{\!\|n\|\!\}_B)]_B$ $\mathsf{Source}(n, B, \{\{\!\|m, B, n\|\!\}_A\}) \wedge$ $\exists X.(\mathsf{Created}(X, \{\!\|n\|\!\}_B) \wedge (X \neq B) \wedge \mathsf{Knows}(X, n))$	(4)
$\mathbf{SRC}$	$\mathsf{Source}(n, B, \{\{\!\|m, B, n\|\!\}_A\}) \wedge$ $\exists X.(X \neq B) \wedge \mathsf{Knows}(X, n) \supset$ $\exists Y.\mathsf{Decrypts}(Y, \{\!\|m, B, n\|\!\}_A)$	(5)
$\mathbf{DEC1}$	$\mathsf{Honest}(A) \wedge \mathsf{Decrypts}(Y, \{\!\|m, B, n\|\!\}_A) \supset (Y = A)$	(6)
$5, 6$	$\mathsf{Honest}(A) \wedge \mathsf{Source}(n, B, \{\{\!\|m, B, n\|\!\}_A\}) \wedge$ $\exists X.(X \neq B) \wedge \mathsf{Knows}(X, n) \supset$ $\mathsf{Decrypts}(A, \{\!\|m, B, n\|\!\}_A)$	(7)
$4, 7, \mathbf{G1-3}$	$(B)[(\{\!\|A, m\|\!\}_B)(\nu n)\langle\{\!\|m, B, n\|\!\}_A\rangle(\{\!\|n\|\!\}_B)]_B$ $\mathsf{Honest}(A) \supset \mathsf{Decrypts}(A, \{\!\|m, B, n\|\!\}_A)$	(8)
$\mathcal{R}(A), \mathbf{HON}$	$\mathsf{Honest}(A) \supset (\mathsf{Decrypts}(A, \{\!\|m, B, n\|\!\}_A) \supset$ $\mathsf{CSent}(A, \{\!\|A, m\|\!\}_B) \wedge \mathsf{CSent}(A, \{\!\|n\|\!\}_B))$	(9)
$8, 9, \mathbf{G2}$	$(B)[(\{\!\|A, m\|\!\}_B)(\nu n)\langle\{\!\|m, B, n\|\!\}_A\rangle(\{\!\|n\|\!\}_B)]_B$ $\mathsf{Honest}(A) \supset (\mathsf{CSent}(A, \{\!\|A, m\|\!\}_B) \wedge \mathsf{CSent}(A, \{\!\|n\|\!\}_B))$	(10)

Table 9. B**'s completed deduction with honesty rule**