Diagrammatic Verification of Security Protocols

Christian Skalka and Scott Smith

The Johns Hopkins University, {ces,scott}@cs.jhu.edu

Abstract

The Actor Specification Diagram Language is a rigorously defined graphical specification language, with features for modelling concurrency and asynchronous communication. These features make the language appropriate for the formal analysis of distributed security protocols. The graphical nature of the language makes protocol descriptions easy to read and understand, vital properties for appeal to a wider user population. Proofs of protocol correctness can be given via graphical transformation, with each step of transformation justified by rigorous argument if desired. To illustrate the utility of a graphical specification language we analyze the well-known Needham-Schroeder Pubic Key protocol.

1 Introduction

The purpose of this paper is to demonstrate the suitability of a formal diagrammatic specification language for the verification of security protocols. The particular language we use is the actor specification diagram language (referred to as *specdiags* for brevity), presented in [18, 19, 20]. In this paper we model and analyze the Needham-Schroeder Public Key ($NSPK$) protocol. Various authentication properties for $NSPK$ have been proven in a number of different formalisms, including [17, 6, 4, 7]. We analyze $NSPK$ to give a useful benchmark of comparison with other specification techniques such as the above.

While specification diagrams have a number of appealing characteristics, their unique strength in the domain of protocol verification is their graphical nature. Since these graphical diagrams are easily understood, and explicitly illustrate the full execution of a system, properties can be "read off" the diagram, rather than requiring assertions in some symbolic language. While the technical advantages of graphical reasoning methods have been explored previously [11], graphical approaches have particular benefits in the context of security protocol analysis; they allow a more intuitive reading of the specification, making the representation more accessible to non-specialists. This is particularly important, since non-specialists are usually the end-users of protocol specifications. In this paper, we demonstrate these features by defining a diagram which models the execution of $NSPK$, and then prove some of the usual authentication properties by making simple transformations and observations of the diagram.

Since specification diagrams have an actor-based semantics [2, 3, 21] which includes a concurrent, asynchronous communication model, they allow for a faithful modeling of distributed computing environments such as the internet. Message delivery delay, exceptional behavior handling, and inclusion of arbitrary mathematical datatypes such as integers, lists, and sets, are some of the particular features of specdiags that allows for complete and faithful specification. The formal semantics and associated proof principles of specification diagrams also allow for a rigorous analysis of security protocols, as will be demonstrated.

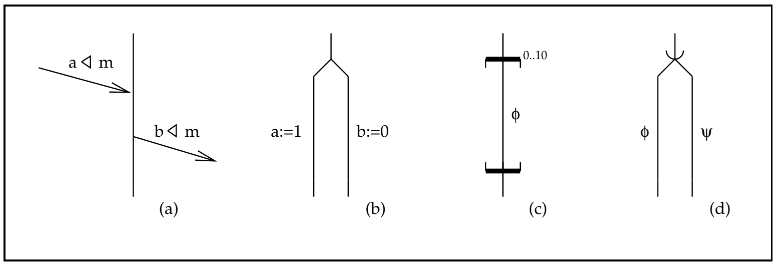

Figure 1: basic diagram operations

2 Specification Diagrams: background

Actor Specification Diagrams are diagrammatic representations of actor systems. They share surface similarities with UML sequence diagrams [5], but have a rigorous semantics and significantly greater expressivity. The semantics are based on the actor model of distributed computation [2, 3]. Actors are named entities that communicate via asynchronous message send and receive, with each actor having its own message queue. Actors in specdiags can be composed in parallel, and can branch nondeterministically on a given predicate. Specification diagrams are firmly rooted in the actor model of distributed computation, but share concepts with CCS/CSP [13, 8] (choice/parallel, traces) and axiomatic program calculi [14] (embedded logical assertions and assumptions). The primary reference for the Actor Specification Diagram Language is [20], which contains a full specification and operational semantics of the language.

A brief overview of specification diagram syntax is given in Figure 1, which contains diagram fragments illustrating the basic operations. Diagrams are read from top to bottom, with lower events occuring after higher events. Fragment (a) in diagram 1 shows the receipt of a message m addressed to a, and then the subsequent send of the same message to b. Fragment (b) shows the fork and parallel composition of two diagram fragments; in one component the assignment $a := 1$ occurs, while in the other $b := 0$ occurs. Fragment (c) shows a loop which is executed an arbitrary number of times between 0 and 10, where the predicate ϕ must hold during each iteration (if ϕ were to fail, that computation path would not be admissible–it never happened). Fragment (d) illustrates a nondeterministic choice between two possible computation paths; in one path the predicate ϕ holds, while ψ holds in the other. There are several other syntactic constructs in specdiags, including diagram recursion, which are not needed for the results of this paper.

Complete specification diagrams are built up from the above basic operations. These diagrams can be considered visual representations of concurrent state transition systems, where states and transitions are represented graphically. Figure 2 contains a simple example of a complete diagram, called *Router*. This diagram specifies an idealized packet router, which receives messages of the form $\mathrm{pkt}(p, m)$, where p is the recipient's address and m is the message body; these messages are passed on to the intended recipient p by the router. This behavior is informally observable via an examination of the diagram. The "$0 \ldots \infty$" notation denotes a loop of arbitrary iteration, possibly infinite for the case of an unbounded stream of inputs to the router.

Like other state transition systems, the semantics of a diagram is given in terms of its observable behavior– that is, not based on the internal structure of the diagram per se, but rather on how it might interact with external actors. The specdiag semantics is rigorously defined in [20], via an operational actor theory [21] which models diagram behavior. The result of this semantics is a set of *interaction paths* (*ip*'s) which fully characterizes the input/output behavior of a diagrammed system. Each ip is an observed input/output trace of the system, and the model

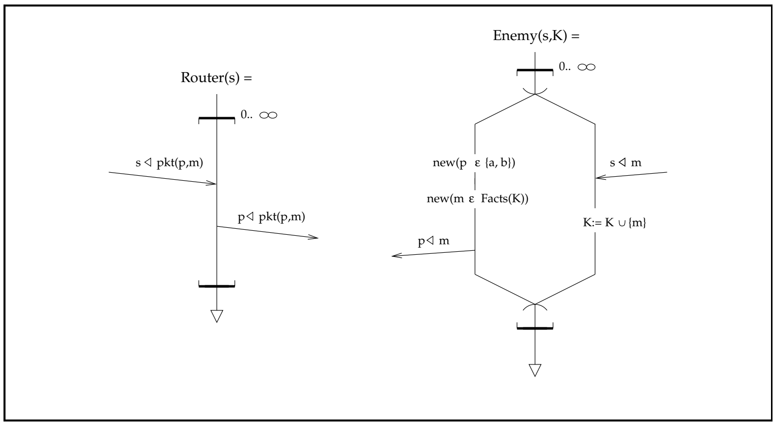

Figure 2: Router and compromised router (enemy) specifications

of the system as a whole is a set of ip's, which gives all possible ways that messages can go into and out of a system. For example, the ip semantics of *Router*, denoted $[\![\mathit{Router}(s)]\!]$, is a set including the following three interaction paths, amongst others:

$$
\begin{array}{l}
[\,] \\
[\,\mathbf{in}(s \lhd \mathrm{pkt}(p_1, m_1)),\ \mathbf{out}(p_1 \lhd \mathrm{pkt}(p_1, m_1))\,] \\
[\,\mathbf{in}(s \lhd \mathrm{pkt}(p_1, m_1)),\ \mathbf{out}(p_1 \lhd \mathrm{pkt}(p_1, m_1)), \\
\qquad\qquad\qquad\qquad \mathbf{in}(s \lhd \mathrm{pkt}(p_2, m_2)),\ \mathbf{out}(p_2 \lhd \mathrm{pkt}(p_2, m_2))\,]
\end{array}
$$

Note that this set includes the empty interaction path [ ], modeling the case for which no messages are ever received. Since *Router* can loop indefinitely, there are an infinite number of interaction paths in the complete ip set interpretation, corresponding to the fact that a router can forward arbitrarily many messages (even infinitely many) in its lifetime (which in theory could be forever). Thus, the semantics of any given specdiag is such a set of interaction paths.

For protocol analysis, it is advantageous to make a very minor extension to the specdiag language, by adding a notion of (meta-)*event* to the language and ip semantics:

Definition 2.1 The primitive statement **event**(m) has the semantics of emitting action **event**(m) on any interaction path.

The purpose of **event** is to allow certain information to be included in a diagram's *ip* semantics as meta-information that can be used to verify protocol behavior.

A low-level analogue of an *ip* is a *computation path* (cp), which is a trace of all computation steps in a system. In addition to **in** and **out** transitions, a cp will contain internal steps such as assignment, variable declaration, etc. An interaction path can be obtained from any computation path by extracting the input and output steps from it (via the function $\mathit{cp2ip}$ defined in [20]). Thus, two systems can have an equivalent semantics, but have different computation paths. In other words, diagrams are amenable to simulation results. These and other specdiag concepts will be clarified by examples and demonstrations in sections 4 and 5.

3 Modelling NSPK: preliminaries

The NSPK protocol has been described extensively in the literature, and the reader is referred to e.g. [9] for a complete description of the protocol. In this section, we make some prelimary definitions that will be essential to our model of the protocol execution. First, we define the following notions of message concatenation and message components:

Definition 3.1 Concatenation of messages m_1, m_2 is denoted as $m_1.m_2$. A message m' is, inductively, a *component* of a message m, denoted $m' \in m$, iff $m' = m$ or $m = m_1.m_2$ and $m' \in m_1$ or $m' \in m_2$.

We formalize the notion of public/private key pairs in our model by defining a mathematical universe $\mathcal{R}$, over which the following axioms hold:

Definition 3.2 The functions (f, f^{-1}) comprise a (*public key*, *private key*) pair over $\mathcal{R}$ iff for all non-empty messages m, the following conditions hold:

1. $f^{-1}(f(m)) = m$
2. $\neg\exists\, f' \in \mathcal{R} \,.\, f' \neq f \wedge f'(m) = f(m)$
3. $\neg\exists f' \in \mathcal{R} \,.\, f' \neq f^{-1} \wedge f'(f(m)) \in m$

Throughout this paper, we work over an arbitrary, fixed domain $\mathcal{R}$. Clause 1 of the above definition ensures that private keys decode public key encodings. Clause 2 ensures that messages cannot be counterfeited. Clause 3 ensures that private keys cannot be forged, so that even some component of an encoded message cannot be retrieved without the appropriate private key. Therefore, this represents an idealized model of cryptography [1].

We will also use the concept of a *nonce* throughout this paper. For our purposes, a nonce is an identifier that is freshly generated upon declaration, and cannot be guessed; that is, if a fresh nonce is created, then for all other nonces n_b in the system, $n_b \neq n_a$.

4 Encoding $NSPK$ as a Specification Diagram

In this section we model the $NSPK$ protocol as a specification diagram. First, we define the $Initiator$ and $Responder$ specification diagrams. We then then specify an "enemy", an entity that can intercept transmissions, send spurious messages and generally attempt to disrupt the protocol, as a diagram *Enemy*. In order to verify $NSPK$, we consider the parallel composition of diagrams $Initiator$, $Responder$ and *Enemy*. This is specified as $Hostilnet1$ in figure 7, which specifies all possible behaviors of the system, including all possible enemy attacks, due to the general manner by which the enemy is specified.

In order to rigorously analyze $Hostilnet1$, we define an expansion of $Hostilnet1$, called $Hostilnet3$, also specified in figure 7, which explicitly details the cases of interaction between $Initiator$, $Responder$ and *Enemy*. $Hostilnet3$ can itself be viewed as a proof of correctness of the protocol, for the correctness of the protocol can easily be read off this diagram. Another benefit of having $Hostilnet3$ at hand is that the geberal behavior of $NSPK$ can be observed simply by looking at the diagram. The difficult task is to rigorously establish the equivalence of $Hostilnet1$ and $Hostilnet3$, a process that requires a careful case analysis on the possible behaviors of $Hostilnet1$. This proof is the subject of Section 5.

Since $NSPK$ executes in distributed environment such as the internet, we use a model of communication whereby principals communicate via addressed messages of the form $\mathrm{pkt}(p, m)$, which are relayed via a router, specified as *Router* in figure 2 and described in section 2. In the simplified model of routing used here, each packet travels through a single router en route to its destination. Communication is always asynchronous, a consequence of the actor-based semantics of specification diagrams [20, 21]. Also in keeping with the internet model, we assume a FIFO ordering on message receipt; while the specdiag semantics [20] does not assume a FIFO

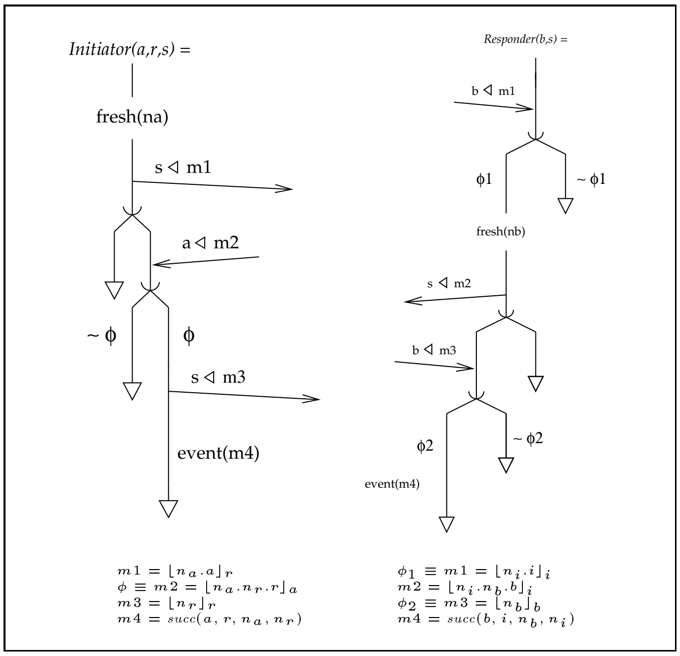

Figure 3: *NSPK* protocol initiator and responder

ordering, making the changes that would entail a FIFO ordering are relatively straightforward, and will not be presented here.

We assume the existence of a PGP-like public/private key system for encryption, where all public keys are assumed to be freely available; we accomplish this by defining a global injective function Ψ that maps actor names to distinct public keys. For brevity, we introduce a shorthand to denote packets with encrypted message bodies: $\lfloor m \rfloor_a \stackrel{\text{def}}{=} \text{pkt}(a, \Psi(a)(m))$. Given this basic notation, our modeling of the *NSPK* protocol is essentially a translation of the *NSPK* protocol into specification diagram form, with messages passed between principals via the router. To simplify matters in the context of this paper, we model a single run of the protocol between two principals.

4.1 Network Component Specifications

The protocol initiator is specified as *Initiator* in figure 3. This diagram has top-level variables a, r and s; where a is the name *Initiator* receives on, r is the name of the principal with whom it will initiate a run of *NSPK*, and s is the name of the router to which packets are sent. The diagram is robust, in that it contains paths for those cases in which invalid messages are received. In the initial diagram transition, the first message in the protocol is sent to r. If a valid response is received, the last message in the protocol is sent, and a "success event" is executed, advertising the value of r and r's nonce. The reason for including this last action is to allow successful protocol execution to be observable externally in the full network composition (see subsection 4.3). If an invalid response to the first sent message is recieved, or if no response is received at all, the diagram stops without executing a success event.

The protocol responder is specified as *Responder* in figure 3. This diagram has top level variables b and s, where b is the name which the diagram receives on and s is defined as in

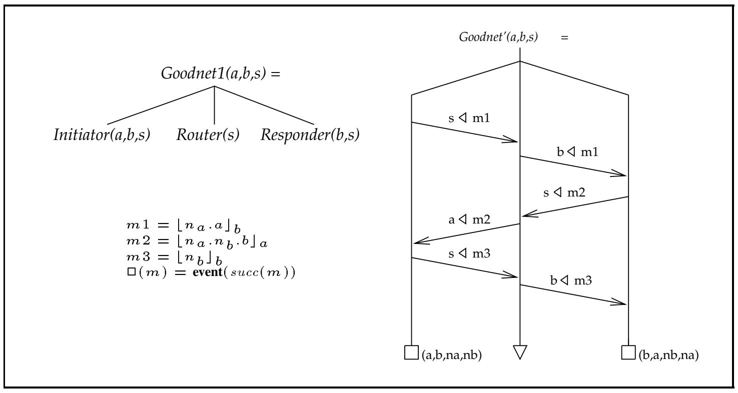

Figure 4: $Goodnet$ and its expansion, *Goodnet'*

$Initiator$. Note that *Responder* also executes an event action upon successful completion of the protocol, advertising the name and nonce of the authenticated party. And, as in the case of $Initiator$, the responder diagram contains explicit paths for those cases in which protocol execution fails.

Given the definitions of *Router*, $Initiator$ and $Responder$, an execution of the $NSPK$ protocol between the initiator and responder, in a completely secure environment, is obtained through the parallel composition of the principals and the router; this composition is specified as *Goodnet*(a, b, s) on the left of Figure 4. The safe and successful execution of the protocol in this environment can itself be specified diagrammatically, as *Goodnet'* on the right of Figure 4. The equivalence of these diagrams can be proved:

Proposition 4.1 $[\![Goodnet(a, b, s)]\!] = [\![Goodnet'(a, b, s)]\!]$.

The above proposition may first appear vacuous, but there is a significant difference between the two specifications: *Goodnet'*(a, b, s) constrains the sends of the initiator to be received by the router, and immediately forwarded to the responder. In *Goodnet*(a, b, s) on the other hand, there is no such constraint imposed on sending and receiving, and reasoning is required to show that *Goodnet'*(a, b, s) represents the only message-passing behavior of the system. One of the key features of the specdiag language is how cross-edges between parallel threads serves to constrain message-passing behavior, a feature which has no direct analogue in other specification languages.

4.2 The Enemy

While *Goodnet* illustrates a flawless execution of the protocol in a friendly environment, we must also consider possible failure of, or attacks on the system. To this end, we replace the *Router* with an *Enemy*, a potentially faulty or malicious router which can intercept, replay and fabricate messages in an attempt to break the protocol. In this presentation, we define the enemy as a router which receives messages sent to the address s, and generates *any* sort of message based on its current state of knowledge. This model captures all possible enemy behaviors—scenarios in which messages are replayed, or intercepted, or altered or fabricated can be simulated [16].

In order to formally describe what the enemy can do with its knowledge, we define the set $Facts(K)$, which is the set of messages that the enemy is able to generate given knowledge base K. The enemy is able to generate new messages by concatenating other messages, deconstructing

D1:

1. $m1 = \lfloor n_a.a \rfloor_s$
2. $m2 = \lfloor n_a.n_s.s \rfloor_a$
3. $m3 = \lfloor n_s \rfloor_s$

D3:

1. $m1 = \lfloor n_a.a \rfloor_\beta$
2. $\phi \equiv \neg \exists n_\beta \ . \ m2 = \lfloor n_a.n_\beta.\beta \rfloor_a$

D1–D4: $\Box(m) = \textbf{event}(succ(m))$

D2:

1. $\phi1 \equiv \exists i, n_i \ . \ m1 = \lfloor n_i.i \rfloor_b$
2. $m2 = \lfloor n_i.n_b.b \rfloor_i$
3. $\psi1 \equiv m3 = \lfloor n_b \rfloor_b$

D4:

1. $m1 = \lfloor n_a.a \rfloor_b$
2. $m2 = \lfloor n_a.n_b.b \rfloor_a$
3. $m3 = \lfloor n_b \rfloor_b$
4. $\phi1 \equiv m4 = \lfloor n_b \rfloor_b$

Figure 5: Symbol definitions for diagram components in Figure 6

other concatenated messages, and by applying functions to messages, including public keys, and accessible private keys. The only restriction we need to make on functions at the enemy's disposal, is that they cannot include the private keys of the principals (otherwise the protocol can always be broken):

Definition 4.1 Let $\mathcal{R}_e$ be a set of functions such that $\mathcal{R}_e \subset \mathcal{R}$, $\mathcal{R}_e$ is closed under composition, $\Psi(a)^{-1} \notin \mathcal{R}_e$ and $\Psi(b)^{-1} \notin \mathcal{R}_e$.

Then, the set of messages that can be derived from a given knowledge base K, called $\mathit{Facts}(K)$, is defined as follows:

Definition 4.2 Let K be some set of messages (knowledge), and let m be a message; then $m \in \mathit{Facts}(K)$ iff one of the following conditions holds:

1. $m \in K$
2. $\exists m', f \in \mathcal{R}_e \ . \ m' \in K \wedge m = f(m')$

A consequence of this definition, which will be relevant in later proofs, is the following:

Corollary 4.1 If $m \in \mathit{Facts}(K)$ and $K \subseteq K'$, then $m \in \mathit{Facts}(K')$.

Proof By definition 4.2, and case analysis of m. $\Box$

With this definition of the enemy's abilities, the enemy itself is abstractly specified by the diagram *Enemy* in figure 2.

This entity receives all message packets sent by either principal, removes the message body for inclusion in its knowledge base, and at any time is able to send messages to either principal. It is important to note that the enemy cannot violate the rules of encryption specified in definition 3.2. In particular, the enemy cannot forge messages containing unknown nonces:

Lemma 4.1 For all actor names i, sets of knowledge K, possibly empty messages m_1, m_2 and nonces v, if $v \notin \mathit{Facts}(K)$ then $\Psi(i)(m_1.v.m2) \notin \mathit{Facts}(K)$.

Proof By definitions 4.2 and 3.2. $\Box$

As an example of the sort of information which cannot be extracted from a particular knowledge base, in a form which will be useful in later proofs, the following lemma gives concrete examples of "unknowns" for the enemy with knowledge K:

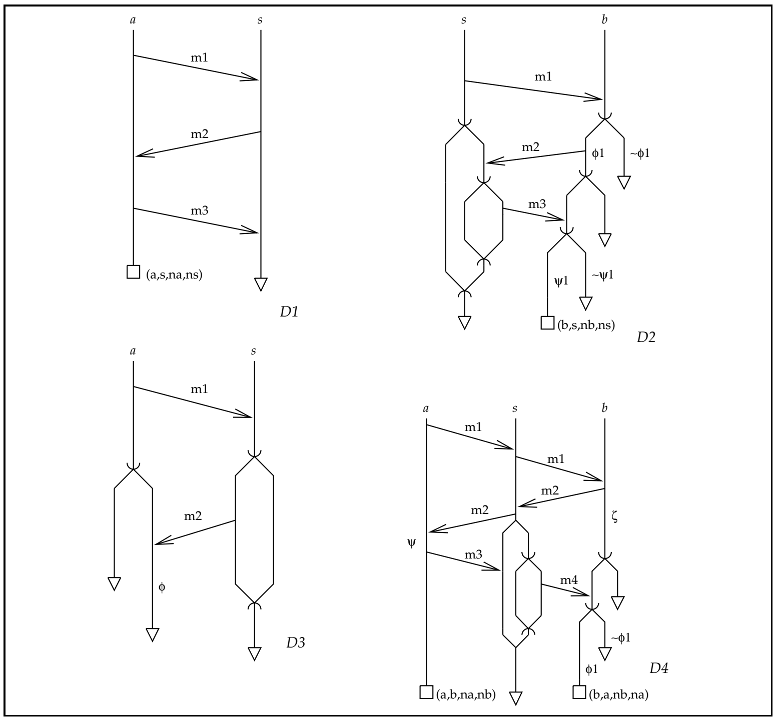

Figure 6: *Hostilnet3* Diagram Components

Lemma 4.2 For all K and nonces $v_a, v_b \notin \mathit{Facts}(K)$ and $v_s \in \mathit{Facts}(K)$, the following assertions hold:

1. $v_a, v_b \notin \mathit{Facts}(K \cup \{\lfloor v_a.a \rfloor_b, \lfloor v_b \rfloor_b, \lfloor v_a.v_b.b \rfloor_a\})$
2. $v_a \notin \mathit{Facts}(K \cup \{\lfloor v_a.a \rfloor_b, \lfloor v_s.v_b.b \rfloor_s\})$
3. $v_b \notin \mathit{Facts}(K \cup \{\lfloor v_a.a \rfloor_s, \lfloor v_s.v_b.b \rfloor_a\})$

Proof By definitions 4.2 and 3.2. □

4.3 The Complete Model

The initiator, responder, and enemy have now been defined, and the full model consists of the composition of the three entities. The standard composition is the case where the initiator wishes to authenticate with the responder– that is, with b, the receiving name of $\mathit{Responder}$. However, we also wish to analyze the case where initiator attempts to authenticate with the enemy, to show this does not lead to a faulty conclusion that authentication with the responder took place– the consequence of a well-known attack on an older version of NSPK, described in [9, 16]. Therefore, we will also model execution of the protocol with the initiator starting a run with the enemy itself– that is, with s, whose private key the enemy has access to (see definition 4.1).

The network model for an execution of the NSPK protocol is the parallel composition of the initiator, responder and enemy, which is specified as $\mathit{Hostilnet1}$ in figure 7. Note that in this composition, all messages are internal—that is, all senders and receivers of messages are contained within the system. This means that there are no receptionists (the set of actors for a

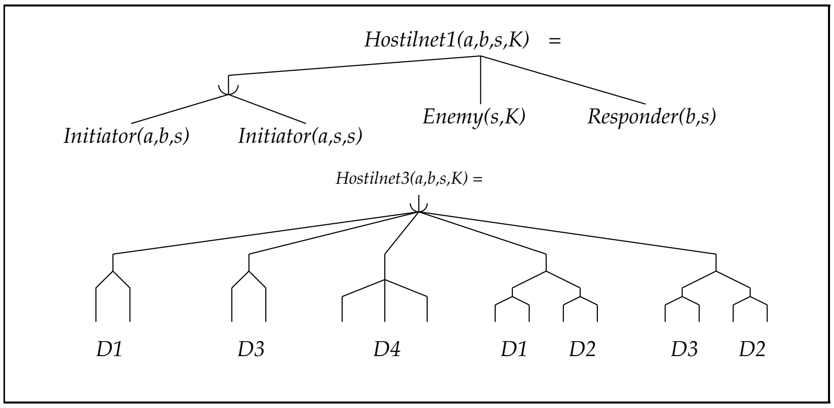

Figure 7: *Hostilnet1* and its expansion, *Hostilnet3*.

system that receive external messages) nor external actors (the set of external names known to the system) for *Hostilnet1* throughout computation– that is, the system remains *closed*. Additionally, all $NSPK$ networks that are constructed in this paper will have the same set of top-level actors $IActs$. Hence, we can at this point simplify notation by implicitly defining for any $NSPK$ network:

Definition 4.3 The receptionists ρ and external actors χ are both $\emptyset$ for this system; the internal actors set $IActs$ is $\{a, b, s, K\}$. In the following, the actor theory notation $[\![D \ : \ Th]\!]$ is taken to mean $[\![\langle D, IActs\rangle_\chi^\rho \ : \ Th]\!]$.

Since all message sends are internal, only **event** actions will show up in the interaction paths of the system. Each of these actions occurs upon successful execution of the protocol by either principal, and contains a tuple $succ(x_1, x_2, n_1, n_2)$, where x_1 is the name of the principal, x_2 is the name of the principal which x_1 believes itself to have authenticated, and n_1 and n_2 are the nonces used in the protocol run. Our method of analyzing the protocol is then to enumerate all possible interaction paths in $[\![Hostilnet1(a, b, s, K)]\!]$. We can then use the information in these *ip*s to analyze properties of any given run. For example, we expect that after an execution of $Hostilnet1$, if $Responder$ believes itself to have authenticated $Initiator$, then $Initiator$ believes itself to have authenticated $Responder$. Since $Initiator$ is identified by a and $Respond$ is identified by b in this system, we verify this property by showing that for all interaction paths in π $[\![Hostilnet1(a, b, s, K)]\!]$, if **event**$(succ(b, a, n_b, n_a)) \in \pi$ then **event**$(succ(a, b, n_a, n_b)) \in \pi$.

The semantics of *Hostilnet1* are given by the specdiag operational semantics, and this semantics must be analyzed to characterize the interaction paths of *Hostilnet1* to determine if the above correctness property holds. We may diagrammatically analyze *Hostilnet1* by asserting its equivalence to another diagram, *Hostilnet3*, which explicitly enumerates the cases the protocol may perform. This diagram is given in Figure 7, with components of this figure in turn found in Figures 6 and 5. In these figures, all messages $p \lhd m$ are written as m for simplicity; message recipients are obvious from the diagram, since all communication is internal. Also, note the symbol $\Box_{(m)}$, which abbreviates the statement **event**$(succ(m))$.

We will demonstrate in the next Section that *Hostilnet3* is equivalent to *Hostilnet1*, so that the above correctness condition can directly be "read off" of *Hostilnet3*: observe the components of Figure 6 all have an event box labeled $\Box_{(b,a,n_b,n_a)}$ *only* for the case there is also a box labeled $\Box_{(a,b,n_a,n_b)}$.

5 Proof of correspondence of *Hostilnet1* and *Hostilnet3*.

The principal result of this paper is the following theorem, which establishes the semantic equivalence of *Hostilnet1* and *Hostilnet3*:

Theorem 5.1 $[\![\mathit{Hostilnet1}(a,b,s,K)]\!] = [\![\mathit{Hostilnet3}(a,b,s,K)]\!]$ for all a, b, s, K.

Our proof strategy is to enumerate the interaction paths of *Hostilnet1*, by collapsing its computation paths into a smaller, respresentative set. To accomplish this, we will use the Actor Theory (AT) framework, described in [20]. Intuitively, Actor Theories allow us to break down any Specification Diagram computation into atomic steps, which can then be manipulated and analyzed. In fact, once the atomic state transitions for the diagram are defined, we can enumerate the *ip*s of the system using an inductive method. In this way, our approach is similar to that used by Paulson in [15]. Any proofs not in the text appear in the appendix of this paper.

Our first step in the process of explicating $[\![\mathit{Hostilnet1}(a,b,s,K)]\!]$ is to generate the *canonical form* actor theory $NcATh$, which is semantically equivalent to *Hostilnet1*. Canonical form refers to an actor theory in which several atomic steps have been combined into a single *big-step*. $NcATh$ is then be transformed into an actor theory $NscATh$, which is in supercanonical form, containing a further simplified set of reaction rules for the network. These reaction rules are explicitly defined, allowing the interaction paths of $NscATh$ to be enumerated; this in turn will characterize $[\![\mathit{Hostilnet1}(a,b,s,K)]\!]$, since $NcATh$ and $NscATh$ are semantically equivalent by construction, as will be demonstrated in lemma 5.7.

5.1 Canonical Form AT Definitions

In this section, we define the component AT's of $NcATh$, then $NcATh$ itself. These definitions are in canonical form, meaning that each AT transition consists of a number of computation steps which, without loss of generality, can be collapsed into a single big-step. This reduces the number of semantically equivalent computation paths that must be considered, simplifying the proof.

Definition 5.1 The canonical form AT's $IcATh$, $RcATh$ and $EcATh$ are defined as follows:

1. $IcATh = \Delta(\langle \mathit{Initiator}(a,r,s), \{a\}\rangle)$
2. $RcATh = \Delta(\langle \mathit{Responder}(b,s), \{b\}\rangle)$
3. $EcATh = \Delta(\langle \mathit{Enemy}(s,K), \{s\}\rangle)$

As described in [20], the Δ operation generates a canonical form AT. $IcATh$, $RcATh$ and $EcATh$ are characterized with the following three lemmas:

Lemma 5.1 The reaction rules of $IcATh$ are I1 through I4 as defined in figure 8, where the state families associated with these rules are isomorphic to the following:

$\mathit{Initiat}(a,r,s) = \mathit{Initiator}(a,r,s)$

$\mathit{Init1}(a,r,s,v_a) = \{\ n_a \mapsto v_a :$
$\quad$ **receive**$(a \lhd m) \oplus$ **stop**;
$\quad$ **constrain**$(m = \lfloor n_a.n_r.r\rfloor_a) \oplus$ (**constrain**$(m \neq \lfloor n_a.n_r.r\rfloor_a)$; **stop**);
$\quad$ **send**$(s \lhd \lfloor n_r\rfloor_r)$;
$\quad$ **event**$(succ(a,r,n_a,n_r))$;
$\quad\}$

Proof By the specification of $\mathit{Initiator}$, and by the definition of Δ as given in [20]. $\square$

Lemma 5.2 The reaction rules of RcATh are R1 through R6 as defined in figure 8, where the state families associated with these rules are isomorphic to the following:

$$\mathit{Respond}(b,s) = \mathit{Responder}(b,s)$$

$$\begin{array}{l} \mathit{Resp1}(b,s,v_b,v_{iv},iv) = \{\ n_b \mapsto v_b,\ n_i \mapsto v_{iv},\ i \mapsto iv : \\ \qquad \mathbf{receive}(b \lhd m)\ \oplus\ \mathbf{stop}; \\ \qquad \mathbf{constrain}(m = \lfloor n_b \rfloor_b)\ \oplus\ (\mathbf{constrain}(m \neq \lfloor n_b \rfloor_b);\ \mathbf{stop}); \\ \qquad \mathbf{event}(\mathit{succ}(b,i,n_b,n_i)); \\ \qquad \} \end{array}$$

Proof By the specification of $\mathit{Responder}$, and by the definition of Δ as given in [20]. $\square$

Lemma 5.3 There is only one state in the specialized actor theory EcATh, which is isomorphic to the specification diagram enemy: $\mathit{Enmy}(s,K) = \mathit{Enemy}(s,K)$. The reaction rules of EcATh are E1 through E3 as defined in diagram 8.

Proof By the specification of *Enemy*, and by the definition of Δ as given in [20]. $\square$

Now, we use localization and the product operations ($\times$ and ,) defined in [20], to construct NcATh from the component AT's. We introduce the metavariable β which ranges over $\{b,s\}$ to simplify notation:

Definition 5.2 Let:

$$\mathit{Hostilnet2}(a,b,s,K) = \mathit{Initiat}(a,\beta,s), \mathit{Respond}(b,s), \mathit{Enmy}(s,K)$$

Then the actor theory NcATh is obtained from the localized product $\mathit{Loc}(\mathit{IcATh} \times \mathit{RcATh} \times \mathit{EcATh})$, with all states unreachable from the start state $\mathit{Hostilnet2}(a,b,s,K)$ removed.

Given results described in [20], it is straightforward to prove equivalence of this canonical network actor theory, and the network diagram:

Lemma 5.4 $[\![\mathit{Hostilnet1}(a,b,s,K) : \mathit{cSDTh}]\!] = [\![\mathit{Hostilnet2}(a,b,s,K) : \mathit{NcATh}]\!]$ for all a,b,s,K.

5.2 Bubbling and Supercanonical Form

With lemma 5.4, we take one step towards bridging the gap between $\mathit{Hostilnet1}$ and $\mathit{Hostilnet3}$. Now we can take another, by showing that a number of big-step transitions in a given canonical form cp can be rearranged without loss of generality; a transformed actor theory which then collapses these transitions into a single step is called *supercanonical*. In this particular case, we produce a supercanonical actor theory NscATh by *bubbling* the computation paths of NcATh. Bubbling is the process of synchronizing message sends and receives in computation paths:

Definition 5.3 For all computation paths π of $\mathit{Hostilnet2}$, $\mathit{sbubbled}(\pi)$ holds iff any transition in π containing a **send** by *Enmy* is immediately followed by a transition containing the corresponding **receive** by one of the principals. Conversely, $\mathit{rbubbled}(\pi)$ holds iff any transition in π containing a **send** by either $\mathit{Initiat}$ or $\mathit{Respond}$ is immediately followed by a transition containing the corresponding **receive** by *Enmy*, and $\mathit{bubbled}(\pi)$ holds iff $\mathit{sbubbled}(\pi)$ and $\mathit{rbubbled}(\pi)$.

$$\mathit{Initiat}(a,r,s) \xrightarrow{\textbf{fresh}(v_a);\textbf{send}(s \lhd \lfloor v_a.a \rfloor_r)} \mathit{Init1}(a,r,s,v_a) \tag{I1}$$

$$\mathit{Init1}(a,r,s,v_a) \longrightarrow \textbf{stop} \tag{I2}$$

$$\mathit{Init1}(a,r,s,v_a) \xrightarrow{\textbf{receive}(a \lhd m);\textbf{constrain}(m \neq \lfloor v_a.v_r.r \rfloor_a)} \textbf{stop} \tag{I3}$$

$$\mathit{Init1}(a,r,s,v_a) \xrightarrow{\textbf{receive}(a \lhd m);\textbf{constrain}(m = \lfloor v_a.v_r.r \rfloor_a);\textbf{send}(s \lhd \lfloor v_r \rfloor_r);\textbf{event}(succ(a,r,v_a,v_r));} \textbf{stop} \tag{I4}$$

$$\mathit{Respond}(b,s) \longrightarrow \textbf{stop} \tag{R1}$$

$$\mathit{Respond}(b,s) \xrightarrow{\textbf{receive}(b \lhd m);\textbf{constrain}(m \neq \lfloor v_{iv}.iv \rfloor_b)} \textbf{stop} \tag{R2}$$

$$\mathit{Respond}(b,s) \xrightarrow{\textbf{receive}(b \lhd m);\textbf{constrain}(m = \lfloor v_{iv}.iv \rfloor_b);\textbf{fresh}(v_b);\textbf{send}(s \lhd \lfloor v_{iv}.v_b.b \rfloor_{iv})} \mathit{Resp1}(b,s,v_b,v_{iv},iv) \tag{R3}$$

$$\mathit{Resp1}(b,s,v_b,v_{iv},iv) \longrightarrow \textbf{stop} \tag{R4}$$

$$\mathit{Resp1}(b,s,v_b,v_{iv},iv) \xrightarrow{\textbf{receive}(b \lhd m);\textbf{constrain}(m \neq \lfloor v_b \rfloor_b);} \textbf{stop} \tag{R5}$$

$$\mathit{Resp1}(b,s,v_b,v_{iv},iv) \xrightarrow{\textbf{receive}(b \lhd m);\textbf{constrain}(m = \lfloor v_b \rfloor_b);\textbf{event}(succ(b,i,v_b,v_i))} \textbf{stop} \tag{R6}$$

$$\mathit{Enmy}(s,K) \longrightarrow \textbf{stop} \tag{E1}$$

$$\mathit{Enmy}(s,K) \xrightarrow{\textbf{receive}(s \lhd \lfloor mv \rfloor_y)} \mathit{Enmy}(s, K \cup \{\lfloor mv \rfloor_y\}) \tag{E2}$$

$$\mathit{Enmy}(s,K) \xrightarrow{\textbf{new}(pv \in \{a,b\});\textbf{new}(mv \in Facts(K));\textbf{send}(pv \lhd \lfloor mv \rfloor_{pv})} \mathit{Enmy}(s,K) \tag{E3}$$

Figure 8: Reaction rules for $IcATh$, $RcATh$ and $EcATh$

By showing that message sends and receives can be synchronized without losing semantic information, we show that the simpler $NscATh$ is equivalent to $NcATh$. In our proof, we first consider the internal structure of the system in terms of its computation paths. Since computation paths π contain the actions that constitute interaction paths, we can trivially extract an interaction path from any π. The function $\mathit{cp2ip}$ formalizes this operation; as defined in [20], $\mathit{cp2ip}(\pi)$ is the interaction path obtained from π. Thus, we can say that if π_1 and π_2 are both computation paths, and $\mathit{cp2ip}(\pi_1) = \mathit{cp2ip}(\pi_2)$, then π_1 is semantically equivalent to π_2. This notion of equivalence is central to proving the sound transformation of $NcATh$ into supercanonical form.

Lemma 5.5 For all computation paths π of $\mathit{Hostilnet2}$, there exists a computation path π' of $\mathit{Hostilnet2}$ such that $sbubbled(\pi')$, and $\mathit{cp2ip}(\pi') = \mathit{cp2ip}(\pi)$.

Lemma 5.6 For all computation paths π of $\mathit{Hostilnet2}$, there exists a computation path π' of $\mathit{Hostilnet2}$ such that $rbubbled(\pi')$ and $\mathit{cp2ip}(\pi') = \mathit{cp2ip}(\pi)$; and also if $sbubbled(\pi)$, then $sbubbled(\pi')$.

Lemma 5.7 For all computation paths π of $\mathit{Hostilnet2}$, there exists a computation path π' of $\mathit{Hostilnet2}$ such that $bubbled(\pi')$, and $\mathit{cp2ip}(\pi') = \mathit{cp2ip}(\pi)$.

Proof The result follows immediately from lemmas 5.5 and 5.6. □

Now, we define the supercanonical actor theory $NscATh$:

Definition 5.4 The actor theory $NScATh$ contains the reaction rules of $NcATh$, except with **send** and **receive**s of a given message collapsed into a single step; the states of $NscATh$ are the states of $NcATh$, with unreachable states removed.

By the previous bubbling lemmas, the semantics of $NscATh$ are equivalent to those of $NcATh$:

Lemma 5.8 $[\![Hostilnet2(a,b,s,K) : NcATh]\!] = [\![Hostilnet2(a,b,s,K) : NscATh]\!]$ for all a, b, s and K.

Proof By definitions 5.3 and 5.8 and lemma 5.7:

$$[\![Hostilnet2(a,b,s,K) : NcATh]\!] \subseteq [\![Hostilnet2(a,b,s,K) : NscATh]\!]$$

And since $NscATh$ is constructed by reducing the number of computation paths in $NcATh$, it must be the case that:

$$[\![Hostilnet2(a,b,s,K) : NscATh]\!] \subseteq [\![Hostilnet2(a,b,s,K) : NcATh]\!]$$

hence the lemma holds. $\square$

Thus, bubbling allows us to synchronize message sends and receives, which simplifies the theory, allowing us to consider even fewer cases in the analysis. In the next section, we use $NscATh$ to make the final bridge between the initial protocol specification and figure 7, which illustrates the semantics of protocol execution.

5.3 Interaction Paths of $NscATh$, Analysis of $NSPK$

In this section, we enumerate $[\![Hostilnet2(a,b,s,K) : NscATh]\!]$. It is easy to show that this set is equivalent to $[\![Hostilnet3(a,b,s,K)]\!]$, by a simple observation of $Hostilnet3$. Using this result and lemmas 5.4 and 5.8, it will be possible to prove Theorem 5.1 in a straightforward manner.

Our method of enumerating $[\![Hostilnet2(a,b,s,K) : NscATh]\!]$ is to first enumerate the computation paths of the system. We accomplish this by definition 5.4 and induction on the length of computation sub-paths; see lemma 6.2 in the appendix. With this computation path enumeration, it is easy to enumerate the associated interaction paths in the following lemma:

Lemma 5.9 The interaction paths of $[\![Hostilnet2(a,b,s,K) : NscATh]\!]$ are as follows:

1. [ ]
2. [**event**$(succ(a,s,v_a,v_r))$]
3. [**event**$(succ(b,s,v_b,v_i))$]
4. [**event**$(succ(a,s,v_a,v_r))$, **event**$(succ(b,s,v_b,v_i))$]
5. [**event**$(succ(b,s,v_b,v_i))$, **event**$(succ(a,s,v_a,v_r))$]
6. [**event**$(succ(a,b,v_a,v_b))$]
7. [**event**$(succ(a,b,v_a,v_b))$, **event**$(succ(b,a,v_b,v_a))$]
8. [**event**$(succ(b,a,v_b,v_a))$, **event**$(succ(a,b,v_a,v_b))$]

Given this enumeration of $[\![Hostilnet2(a,b,s,K) : NscATh]\!]$, it is possible to show correspondence of the supercanonical theory and the full network specification in figure 7, by a straightforward observation of the diagram:

Lemma 5.10 $[\![\mathit{Hostilnet2}(a,b,s,K) : \mathit{NscATh}]\!] = [\![\mathit{Hostilnet3}(a,b,s,K) : \mathit{CSDth}]\!]$ for all a,b,s,K.

Proof Lemma 5.9 lists all interaction paths of $\mathit{Hostilnet2}$; by observing $\mathit{Hostilnet3}$ in figure 7, it is easy to see that these interaction paths are exactly the ones that occur amongst the possible computation paths of $\mathit{Hostilnet3}$. $\square$

Finally, the main Theorem of the paper can be demonstrated– that is, a proof of correspondence between the textual and graphical versions of the diagram– by drawing a correspondence between the interaction paths of NscATh and $\mathit{Hostilnet3}$, and by using Lemmas 5.4 and 5.8 to make the bridge between $\mathit{Hostilnet1}$ and NscATh:

Proof of Theorem 5.1 By lemma 5.4:

$$[\![\mathit{Hostilnet1}(a,b,s,K) : \mathit{cSDTh}]\!] = [\![\mathit{Hostilnet2}(a,b,s,K) : \mathit{NcATh}]\!]$$

By lemma 5.8:

$$[\![\mathit{Hostilnet2}(a,b,s,K) : \mathit{NcATh}]\!] = [\![\mathit{Hostilnet2}(a,b,s,K) : \mathit{NscATh}]\!]$$

And, by lemma 5.10:

$$[\![\mathit{Hostilnet2}(a,b,s,K) : \mathit{NscATh}]\!] = [\![\mathit{Hostilnet3}(a,b,s,K) : \mathit{CSDth}]\!]$$

Hence, the theorem holds. $\square$

Given this result, relevant verification properties for NSPK fall out as a corollaries, so that protocol analysis becomes a matter of simply observing the diagram and its interaction paths. For example:

Corollary 5.1 If **event**$(succ(b,a,n_b,n_a)) \in \pi$, then **event**$(succ(a,b,n_a,n_b)) \in \pi$ for all $\pi \in [\![\mathit{Hostilnet1}(a,b,s,K)]\!]$.

Proof By theorem 5.1 and lemma 5.9. $\square$

Thus, we have established correctness of the NSPK protocol in our model. Of course, corrolary 5.1 stresses only one particular property of the protocol; the strength of $\mathit{Hostilnet3}$ is that the full nature of the protocol execution is apparent, so that a complete understanding of its workings can be gained simply by an observation of the diagram.

6 Conclusion

The main contribution of this paper is a demonstration of the feasibility of a formal, diagrammatic language for the specification and analysis of security protocols. In particular, the diagram $\mathit{Hostilnet3}$ of Figures 7 and 6 gives an elegant graphical characterization of the possible behaviors of the Needham-Schroeder protocol under enemy attack, showing all the different cases that may arise during attack attempts. Specification diagrams have an operational semantics in an actor theory framework, which allows properties of diagrams to be rigorously established. From a purely formal standpoint, specification diagrams do not offer much over existing security protocol specification languages such as CSP [17] and I/O automata [10]—it is toward the goals of readible and expressive specifications that the language covers new ground.

For simplicity, we modeled execution of a single run of the Needham-Schroeder protocol in a closed system, with principals running the protocol once. This result should be extended to a more realistic model of a network, as an open system in which multiple runs of the protocol may occur. For future work we aim to analyze other security protocols to test the effectiveness of specification diagrams in a broader context.

References

[1] M. Abadi and A.D. Gordon. A calculus for cryptographic protocols: the spi calculus. In *Proceedings of the Fourth ACM Conference on Computer and Communications Security*, pages 36–47. ACM Press, April 1997.

[2] G. Agha. *Actors: A Model of Concurrent Computation in Distributed Systems*. MIT Press, Cambridge, Mass., 1986.

[3] G. Agha, I. A. Mason, S. F. Smith, and C. L. Talcott. A foundation for actor computation. *Journal of Functional Programming*, 7:1–72, 1997.

[4] I. Cervesato, N.A. Durgin, P.D. Lincoln, J.C. Mitchell, and A. Scedrov. A meta-notation for protocol analysis. In *Proceedings of the 12th IEEE Computer Security Foundations Workshop*, Mordana, Italy, June 28–30 1999.

[5] Rational Software Corporation. *UML Notation Guide, version 1.1*. September 1997. Obtained From `http://www.rational.com`.

[6] G. Denker, J. Meseguer, and C. Talcott. Protocol specification and analysis in Maude. In N. Heintze and J. Wing, editors, *Proc. of Workshop on Formal Methods and Security Protocols*, June 1998.

[7] F. Javier Thayer Fabrega and Jonathan C. Hertzog. Strand spaces: Why is a security protocol correct? In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 160–171, Oakland, California, May 8-10 1998.

[8] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, 1985.

[9] G. Lowe. An attack on the needham-schroeder public-key authentication protocol. *Information Processing Letters*, 56, 1995.

[10] Nancy Lynch. I/O automaton models and proofs for shared-key communication systems. In *Proceedings of the 12th IEEE Computer Security Foundations Workshop (CSFW'99)*, Mordana, Italy, June 28–30 1999.

[11] Zohar Manna and Henry Sipma. Verification of paramaterized systems by dynamic induction on diagrams. In *Proceedings of the 11th Conference on Computer Aided Verification (CAV99)*, volume 1633, pages 25–43. LNCS, 1999.

[12] Catherine Meadows. Formal verification of cryptographic protocols: A survey. In *Advances in Cryptology- Asiacrypt 94*, volume 917 of *LNCS*, pages 133–150. Springer-Verlag, 1995.

[13] Robin Milner. *A Calculus of Communicating Systems*, volume 92 of *LNCS*. Springer Verlag, 1980.

[14] Greg Nelson. A generalization of dijkstra's calculus. *TOPLAS*, 11:517–561, 1987.

[15] L.C. Paulson. The inductive approach to verifying cryptographic protocols. *J. Computer Security*, 6:85–128, 1998.

[16] Steve Schneider. Using csp for protocol analysis: The needham schroeder public-key protocol. Technical Report CSD-TR-96-14, Royal Holloway, University of London, Nevember 21 1996.

[17] Steve Schneider. Verifying authentication protocols in CSP. *IEEE Transactions on Software Engineering*, 24(9):741–758, September 1998.

[18] S. Smith. On specification diagrams for actor systems. In C. Talcott A. Gordon, A. Pitts, editor, *Proceedings of the Second Workshop on Higher-Order Techniques in Semantics*, Electronic Notes in Theoretical Computer Science. Elsevier, 1998. `http://www.elsevier.nl/locate/entcs/volume10.html`.

[19] S. Smith and C. Talcott. Specification diagrams for actor systems. In *Formal Methods in Object-Oriented Distributed Systems (FMOODS)*. Kluwer Academic Publishers, 1999.

[20] S. Smith and C. Talcott. Specification diagrams for actor systems. *HOSC, to appear*, 2002. http://www.cs.jhu.edu/ scott/specdiag/Papers/specdiag-11-98.ps.

[21] C. L. Talcott. Composable semantic models for actor theories. *Higher-Order and Symbolic Computation*, 11(3), 1998.

Appendix

Proof of Lemma 5.4 By definition 5.2 in this paper and lemma 5.14 in [20]:

$$[\![\mathit{Hostilnet1}(a,b,s,K) : cSDTh]\!] = [\![\mathit{Hostilnet2}(a,b,s,K) : cSDTh \times cSDTh \times cSDTh]\!]$$

By definitions 5.1-5.3 in this paper and lemmas 5.13 and 4.8 in [20]:

$$[\![\mathit{Hostilnet2}(a,b,s,K) : cSDTh \times cSDTh \times cSDTh]\!] = [\![\mathit{Hostilnet2}(a,b,s,K) : IcATh \times EcATh \times RcATh]\!]$$

And, by definition 5.2 in this paper and lemma 4.9 in [20]:

$$[\![\mathit{Hostilnet2}(a,b,s,K) : IcATh \times EcATh \times RcATh]\!] = [\![\mathit{Hostilnet2}(a,b,s,K) : NcATh]\!]$$

Therefore, the lemma holds. $\square$

The following definition and lemma are used in the proof of lemmas 5.5 and 5.6:

Definition 6.1 For all computation paths π of $\mathit{Hostilnet2}$, $pe(\pi)$ holds iff all **receive**'s in π occur before **stop**$_{\mathit{Enmy}}$ occurs.

Lemma 6.1 For all computation paths π of $\mathit{Hostilnet2}$ such that $\neg pe(\pi)$, there exists a computation path π' of $\mathit{Hostilnet2}$ such that $pe(\pi')$, and $cp2ip(\pi') = cp2ip(\pi)$.

Proof Suppose π is a computation path of $\mathit{Hostilnet2}$ such that $\neg pe(\pi)$, and let:

$$\pi = [\alpha \xrightarrow{r_{j-1}} C_j \xrightarrow{r_j} \beta \xrightarrow{r_{n-1}} C_n]$$

where α and β are sub-paths, and where C_j is the first configuration in which **stop**$_{\mathit{Enmy}}$ occurs in π. Let β' and C'_n be equivalent to β and C_n, except that all occurrences of **stop**$_{\mathit{Enmy}}$ are replaced with *Enmy*(s, K), where K is the knowledge of *Enmy* in configuration C_{j-1}, and let π' be defined as follows:

$$\pi' = [\alpha \xrightarrow{r_j} \beta' \xrightarrow{r_{n-1}} C'_n \xrightarrow{r_{j-1}} C_n]$$

Since C_j is the first configuration in π containing an occurrence of **stop**$_{\mathit{Enmy}}$ by assumption, therefore transition r_{j-1} must occur due to a firing of reaction rule E1. Removal of this transition means that *Enmy* stays "live", so that where $C_{\beta'}$ is the first configuration in β', the sequence $\alpha \xrightarrow{r_j} C_{\beta'}$ is a valid computation sub-path. And since *Enmy* is "dead" by assumption in β and C_n, therefore none of the transitions in the sequence $\beta \xrightarrow{r_{n-1}} C_n$ alter the state of *Enmy* in any way, so that $\beta' \xrightarrow{r_{n-1}} C'_n$, and hence $\alpha \xrightarrow{r_j} \beta' \xrightarrow{r_{n-1}} C'_n$, is a valid computation sub-path. The configuration C_n must contain the full **stop** state for the network, when the *Enmy* finally dies; and since C'_n contains only the *Enmy* in a live state, and r_{j-1} labels a firing of reaction rule E3, therefore the step $C'_n \xrightarrow{r_{j-1}} C_n$ is valid. Thus, π' is a valid computation path for $\mathit{Hostilnet2}$.

Finally, no transitions in π were eliminated in the construction of π'; and since all transitions which contain **event** steps are caused by reaction rules involving either $\mathit{Respond}$ or $\mathit{Initiat}$, which were left unchanged in the construction of π', therefore $cp2ip(\pi) = cp2ip(\pi')$. $\square$

Proof of Lemma 5.5 Let π be a computation path of $Hostilnet2$, and let π_{pe} be a computation path of $Hostilnet2$ such that $pe(\pi_{pe})$ and $cp2ip(\pi_{pe}) = cp2ip(\pi)$, which must exist by lemma 6.1. Suppose that $\neg sbubbled(\pi_{pe})$, and let:

$$\pi_{pe} = [\ \alpha \xrightarrow{r_{i-1}} C_i \xrightarrow{r_i} \beta \xrightarrow{r_{j-1}} C_j \xrightarrow{r_j} \gamma\ ]$$

where r_i contains the action **send**$(p \lhd m)$ occurring due to an enemy send, and r_j contains the corresponding receive by a principal, with $j - i > 1$ by assumption. Let:

$$\pi' = [\ \alpha \xrightarrow{r_{i-1}} \beta' \xrightarrow{r_{j-1}} C'_j \xrightarrow{r_i} C_j \xrightarrow{r_j} \gamma\ ]$$

where C'_j and β' are equivalent to C_j and β respectively, except with $p \lhd m$ removed from all message queues μ in C_j and β. Now, it is straightforward to observe that $\alpha \xrightarrow{r_{i-1}} \beta' \xrightarrow{r_{j-1}} C'_j$ is a valid computation sub-path, since it is equivalent to π_{pe} up to transition r_j, except with the **send**$(p \lhd m)$ event removed, along with all occurrences of $p \lhd m$ from the message queues. Also, $C_j \xrightarrow{r_j} \gamma$ must be a valid sub-path, since it occurs in π. Thus, in order to show that π' is a computation path, it remains to be shown that $C'_j \xrightarrow{r_i} C_j$ is a computation step.

Now, since $pe(\pi_{pe})$ by assumption, therefore the enemy must be "live" in C_j, and hence in C'_j by construction. And since the enemy is live in C'_j, it is capable of sending messages. Further, the label r_i must contain the steps defined in reaction rule E3 by definition of $NcATh$. Thus, since C_j is identical to C'_j except for an occurrence of $p \lhd m$ in the message queue, the step $C'_j \xrightarrow{r_i} C_j$ is valid if $m \in Facts(K')$, where K' is the enemy's knowledge base in C'_j. Noting that it must be the case that $m \in Facts(K)$, where K is the enemy's state of knowledge in C_i, we also observe that the enemy's knowledge base grows monotonically in any computation path by the definition of *Enmy*; thus, $K' \subseteq K$, so that by corollary 4.1, $m \in Facts(K')$. Hence, $C'_j \xrightarrow{r_i} C_j$ is valid, so π' is a computation path.

Additionally, no transitions in π_{pe} were eliminated in the construction of π'; and since all transitions which contain **event** steps are caused by reaction rules involving either $Respond$ or $Initiat$, which were left unchanged in the construction of π', therefore $cp2ip(\pi') = cp2ip(\pi_{pe}) = cp2ip(\pi)$. $\square$

Proof of Lemma 5.6 Let π be a computation path of $Hostilnet2$, and suppose that $\neg rbubbled(\pi)$. Let:

$$\pi = [\ \alpha \xrightarrow{r_{i-1}} C_i \xrightarrow{r_i} \beta \xrightarrow{r_{j-1}} C_j \xrightarrow{r_j} \gamma\ ]$$

where r_{i-1} contains the action **send**$(s \lhd \mathrm{pkt}(r, m))$, occurring due to a send by one of the principals, and r_j contains the corresponding receive by the enemy, with $j - i \geq 1$ by assumption. Let:

$$\pi' = [\ \alpha \xrightarrow{r_{i-1}} C_i \xrightarrow{r_j} C'_i \xrightarrow{r_i} \beta' \xrightarrow{r_{j-1}} \gamma\ ]$$

where C'_j and β' are equivalent to C_j and β respectively, except with $s \lhd \mathrm{pkt}(r, m)$ removed from all message queues μ in C_j and β, and all enemy knowledge bases K replaced with $K \cup \{m\}$. Then, the result in this case follows similarly as in lemma 5.5; the important point to note here is that if the enemy can send a message m' with knowledge base K, then it can also send the message m' with knowledge base $K \cup \{m\}$, by corollary 4.1. Thus, all steps in $C'_i \xrightarrow{r_i} \beta'$ are valid, since $C_i \xrightarrow{r_i} \beta$ is valid.

Additionally, we note that the ordering of enemy sends and principal receives was left unchanged in the construction of π', so that if $sbubbled(\pi)$, then $sbubbled(\pi')$. $\square$

Lemma 6.2 Let $\overrightarrow{R}$ denote the sequence of atomic computation steps in any reaction rule R appearing in figure 8, and let:

$$\begin{array}{llll} N_1 = \overrightarrow{\mathrm{I1}}; \overrightarrow{\mathrm{E2}} & N_2 = \overrightarrow{\mathrm{E3}}; \overrightarrow{\mathrm{I3}} & N_3 = \overrightarrow{\mathrm{E3}}; \overrightarrow{\mathrm{I4}}; \overrightarrow{\mathrm{E2}} & N_4 = \overrightarrow{\mathrm{E3}}; \overrightarrow{\mathrm{R2}} \\ N_5 = \overrightarrow{\mathrm{E3}}; \overrightarrow{\mathrm{R3}}; \overrightarrow{\mathrm{E2}} & N_6 = \overrightarrow{\mathrm{E3}}; \overrightarrow{\mathrm{R5}} & N_7 = \overrightarrow{\mathrm{E3}}; \overrightarrow{\mathrm{R6}} & \end{array}$$

Then, the states and reaction rules of $\mathit{Hostilnet2}(a, b, s, K) : \mathit{NscATh}$ are contained in the states and transitions obtained by making the substitutions specified in Figure 11, where $\emptyset$ is the empty transition, K_ι is an arbitrary initial knowledge base containing some number of nonces, and for brevity $p_x = \Psi(x)$; in any given rule $\Theta_j(S_a) \xrightarrow{N_1, \ldots, N_n} \Theta_k(S_b)$, the substitution Θ_k is taken to apply to each optional transition label N_i.

Proof The result follows by definition 5.4, and induction on the length of an arbitrary computation sub-path π of $\mathit{Hostilnet2}(a, b, s, K) : \mathit{NscATh}$. The most important point to note in the induction is that $v_a, v_b \notin \mathit{Facts}(K_\iota)$ by definition 4.2, since v_a and v_b are freshly created during the course of computation; and, at each subsequent point in the induction, the enemy can never forge a message containing one principal's nonce encrypted with the other principal's public key, by lemma 4.2. Thus, the protocol proceeds "safely", regardless of any attempts to the contrary on the part of the enemy. □

Proof of Lemma 5.9 Let IP be the set containing the seven interaction paths defined above. In order to show that $IP \subseteq [\![\mathit{Hostilnet2}(a, b, s, K) : \mathit{NscATh}]\!]$, by lemma 6.2 it is straightforward to construct computation paths Π from the reaction rules of NscATh, such that for all $p \in IP$, there exists $\pi \in \Pi$ such that $\mathit{cp2ip}(\pi) = p$.

In order to show that $[\![\mathit{Hostilnet2}(a, b, s, K) : \mathit{NscATh}]\!] \subseteq IP$, we observe from the reaction rules of NscATh that the only individual event actions that can occur are the four that are contained in the paths of IP, and each principal can execute at most one event action in any given computation path. Thus, the only permutations of possible event actions which are not in IP are as follows:

1. [**event**$(succ(b, a, v_b, v_a))$]
2. [**event**$(succ(a, s, v_a, v_r))$, **event**$(succ(b, a, v_b, v_a))$]
3. [**event**$(succ(b, a, v_b, v_a))$, **event**$(succ(a, s, v_a, v_r))$]
4. [**event**$(succ(a, b, v_a, v_b))$, **event**$(succ(b, s, v_b, v_s))$]
5. [**event**$(succ(b, s, v_b, v_s))$, **event**$(succ(a, b, v_a, v_b))$]

Let $\overline{IP}$ be the set containing these permutations. Note that by lemma 6.2, the only reaction rule which causes the action **event**$(succ(b, a, v_b, v_a))$ is number 52 in figure 11. However, in this reaction rule $p_b(v_b)$ is in the enemy's knowledge base K; but also by observation of figure 11, $p_b(v_b) \in K$ iff the action **event**$(succ(a, b, v_a, v_b))$ occurs. Hence, permutations 1-3 in $\overline{IP}$ are not in $[\![\mathit{Hostilnet2}(a, b, s, K) : \mathit{NscATh}]\!]$. By the same token, permutations 4-5 in $\overline{IP}$ are not in $[\![\mathit{Hostilnet2}(a, b, s, K) : \mathit{NscATh}]\!]$, since if the action **event**$(succ(b, s, v_b, v_s))$ occurs, $p_b(v_b)$ is never in the enemy's knowledge base, therefore the action **event**$(succ(a, b, v_a, v_b))$ cannot occur before or after **event**$(succ(b, s, v_b, v_s))$. Therefore, $[\![\mathit{Hostilnet2}(a, b, s, K) : \mathit{NscATh}]\!] \subseteq IP$. □

$$
\begin{array}{ll}
\mathit{Initiat}(a,\beta,s)\ ,\ \mathit{Respond}(b,s)\ ,\ \mathit{Enmy}(s,K_\iota) & \text{(S1)} \\
\mathit{Init1}(a,\beta,s,v_a)\ ,\ \mathit{Respond}(b,s)\ ,\ \mathit{Enmy}(s,K) & \text{(S2)} \\
\mathit{Initiat}(a,\beta,s)\ ,\ \textbf{stop}\ ,\ \mathit{Enmy}(s,K) & \text{(S3)} \\
\mathit{Initiat}(a,\beta,s)\ ,\ \mathit{Resp1}(b,s,v_b,v_i,i)\ ,\ \mathit{Enmy}(s,K) & \text{(S4)} \\
\textbf{stop}\ ,\ \mathit{Respond}(b,s)\ ,\ \mathit{Enmy}(s,K) & \text{(S5)} \\
\mathit{Init1}(a,\beta,s,v_a)\ ,\ \textbf{stop}\ ,\ \mathit{Enmy}(s,K) & \text{(S6)} \\
\mathit{Init1}(a,\beta,s,v_a)\ ,\ \mathit{Resp1}(b,s,v_b,v_i,i)\ ,\ \mathit{Enmy}(s,K) & \text{(S7)} \\
\mathit{Init1}(a,\beta,s,v_a)\ ,\ \mathit{Respond}(b,s)\ ,\ \textbf{stop} & \text{(S8)} \\
\textbf{stop}\ ,\ \mathit{Respond}(b,s)\ ,\ \textbf{stop} & \text{(S9)} \\
\textbf{stop}\ ,\ \mathit{Resp1}(b,s,v_b,v_i,i)\ ,\ \mathit{Enmy}(s,K) & \text{(S10)} \\
\mathit{Init1}(a,\beta,s,v_a)\ ,\ \mathit{Resp1}(b,s,v_b,v_i,i)\ ,\ \textbf{stop} & \text{(S11)} \\
\mathit{Init1}(a,\beta,s,v_a)\ ,\ \textbf{stop}\ ,\ \textbf{stop} & \text{(S12)} \\
\textbf{stop}\ ,\ \mathit{Resp1}(b,s,v_b,v_i,i)\ ,\ \textbf{stop} & \text{(S13)} \\
\textbf{stop}\ ,\ \textbf{stop}\ ,\ \mathit{Enmy}(s,K) & \text{(S14)} \\
\textbf{stop}\ ,\ \textbf{stop}\ ,\ \textbf{stop} & \text{(S15)}
\end{array}
$$

Figure 9: State definitions for Figure 11

$$
\begin{array}{ll}
\Theta_1 = \left[\ K_\iota \cup \left\{ \lfloor v_a.a \rfloor_\beta \right\} / K\ \right] & \\
\Theta_2 = \left[\ K_\iota \cup \{ \lfloor v_k.v_b.b \rfloor_k \} / K\ \right] & k \in \{s,a\} \\
\Theta_3 = \left[\ K_\iota \cup \{ \lfloor v_a.a \rfloor_s \} / K,\ s/\beta\ \right] & \\
\Theta_4 = \left[\ K_\iota \cup \{ \lfloor v_a.a \rfloor_s, \lfloor v_s \rfloor_s \} / K,\ s/\beta\ \right] & \\
\Theta_5 = \left[\ K_\iota \cup \left\{ \lfloor v_a.a \rfloor_\beta, \lfloor v_k.v_b.b \rfloor_k \right\}\ \right] & k \in \{s,a\} \\
\Theta_6 = \left[\ K_\iota \cup \{ \lfloor v_a.a \rfloor_b \} / K,\ b/\beta\ \right] & \\
\Theta_7 = \left[\ K_\iota \cup \{ \lfloor v_a.a \rfloor_b, \lfloor v_a.v_b.b \rfloor_a \} / K,\ b/\beta,\ a/i\ \right] & \\
\Theta_8 = \left[\ K_\iota \cup \{ \lfloor v_s.v_b.b \rfloor_s \} / K, s/i\ \right] & \\
\Theta_9 = \left[\ K_\iota \cup \left\{ \lfloor v_a.a \rfloor_s, \lfloor v_\beta \rfloor_s, \lfloor v_k.v_b.b \rfloor_k \right\} / K,\ s/\beta\ \right] & k \in \{s,a\} \\
\Theta_{10} = \left[\ K_\iota \cup \{ \lfloor v_a.a \rfloor_s, \lfloor v_k.v_b.b \rfloor_k \} / K,\ s/\beta\ \right] & k \in \{s,a\} \\
\Theta_{11} = \left[\ K_\iota \cup \left\{ \lfloor v_a.a \rfloor_\beta, \lfloor v_k.v_b.b \rfloor_s \right\} / K,\ s/i\ \right] & k \in \{s,a\} \\
\Theta_{12} = \left[\ K_\iota \cup \{ \lfloor v_a.a \rfloor_b, \lfloor v_b \rfloor_b, \lfloor v_a.v_b.n \rfloor_a \} / K, a/i\ \right] & \\
\Theta_{13} = \left[\ K_\iota \cup \left\{ \lfloor v_a.a \rfloor_s, \lfloor v_\beta \rfloor_s, \lfloor v_k.v_b.b \rfloor_s \right\} / K,\ s/\beta,\ s/i\ \right] & k \in \{s,a\}
\end{array}
$$

Figure 10: Substitution definitions for Figure 11

1. $S1 \xrightarrow{N1} \Theta_1(S2)$
2. $S1 \xrightarrow{N4} S3$
3. $S1 \xrightarrow{N5} \Theta_2(S4)$
4. $S1 \xrightarrow{\emptyset} S3$
5. $\Theta_1(S2) \xrightarrow{N2,\emptyset} \Theta_1(S5)$
6. $\Theta_3(S2) \xrightarrow{N3} \Theta_4(S5)$
7. $\Theta_1(S2) \xrightarrow{N4,\emptyset} \Theta_1(S6)$
8. $\Theta_1(S2) \xrightarrow{N5} \Theta_5(S7)$
9. $\Theta_6(S2) \xrightarrow{N5} \Theta_7(S7)$
10. $\Theta_1(S2) \xrightarrow{\emptyset} S8$
11. $S3 \xrightarrow{N1} \Theta_1(S6)$
12. $\Theta_2(S4) \xrightarrow{N1} \Theta_5(S7)$
13. $\Theta_2(S4) \xrightarrow{N6,\emptyset} \Theta_2(S3)$
14. $\Theta_8(S4) \xrightarrow{N7} \Theta_8(S3)$
15. $\Theta_2(S3) \xrightarrow{N1} \Theta_5(S6)$
16. $\Theta_1(S5) \xrightarrow{N4,\emptyset} \Theta_1(S14)$
17. $\Theta_1(S5) \xrightarrow{N5} \Theta_5(S10)$
18. $\Theta_6(S5) \xrightarrow{N5} \Theta_7(S14)$
19. $\Theta_1(S5) \xrightarrow{\emptyset} S9$
20. $\Theta_4(S5) \xrightarrow{N4,\emptyset} \Theta_4(S14)$
21. $\Theta_4(S5) \xrightarrow{N5} \Theta_9(S13)$
22. $\Theta_4(S5) \xrightarrow{\emptyset} S9$
23. $\Theta_1(S6) \xrightarrow{N2,\emptyset} \Theta_1(S14)$
24. $\Theta_3(S6) \xrightarrow{N3} \Theta_4(S14)$
25. $\Theta_1(S6) \xrightarrow{\emptyset} S12$
26. $\Theta_5(S7) \xrightarrow{N2,\emptyset} \Theta_5(S10)$
27. $\Theta_{10}(S7) \xrightarrow{N3} \Theta_{10}(S10)$
28. $\Theta_5(S7) \xrightarrow{N6,\emptyset} \Theta_5(S6)$
29. $\Theta_{11}(S7) \xrightarrow{N7} \Theta_{11}(S6)$
30. $\Theta_5(S7) \xrightarrow{\emptyset} S11$
31. $\Theta_7(S7) \xrightarrow{N2,\emptyset} \Theta_7(S10)$
32. $\Theta_7(S7) \xrightarrow{N3} \Theta_{12}(S10)$
33. $\Theta_7(S7) \xrightarrow{N6,\emptyset} \Theta_7(S6)$
34. $\Theta_7(S7) \xrightarrow{\emptyset} \Theta_7(S11)$
35. $S8 \xrightarrow{\emptyset} S9$
36. $S8 \xrightarrow{\emptyset} S12$
37. $\Theta_5(S6) \xrightarrow{N2,\emptyset} \Theta_5(S14)$
38. $\Theta_{10}(S6) \xrightarrow{N3} \Theta_{10}(S14)$
39. $\Theta_5(S6) \xrightarrow{\emptyset} S12$
40. $\Theta_7(S6) \xrightarrow{N2,\emptyset} \Theta_7(S14)$
41. $\Theta_7(S6) \xrightarrow{N3} \Theta_{12}(S14)$
42. $\Theta_7(S6) \xrightarrow{\emptyset} \Theta_7(S12)$
43. $S9 \xrightarrow{\emptyset} S15$
44. $\Theta_5(S10) \xrightarrow{N6,\emptyset} \Theta_5(S14)$
45. $\Theta_{11}(S10) \xrightarrow{N7} \Theta_{11}(S14)$
46. $\Theta_5(S10) \xrightarrow{\emptyset} S13$
47. $\Theta_9(S10) \xrightarrow{N6,\emptyset} \Theta_9(S14)$
48. $\Theta_{13}(S10) \xrightarrow{N7} \Theta_{13}(S14)$
49. $\Theta_9(S10) \xrightarrow{\emptyset} S13$
50. $\Theta_7(S10) \xrightarrow{N6} \Theta_7(S14)$
51. $\Theta_7(S10) \xrightarrow{\emptyset} S13$
52. $\Theta_{12}(S10) \xrightarrow{N6,N7,\emptyset} \Theta_{12}(S14)$
53. $\Theta_{12}(S10) \xrightarrow{\emptyset} \Theta_{12}(S13)$
54. $S11 \xrightarrow{\emptyset} S13$
55. $S11 \xrightarrow{\emptyset} S12$
56. $S12 \xrightarrow{\emptyset} S15$
57. $S13 \xrightarrow{\emptyset} S15$
58. $S14 \xrightarrow{\emptyset} S15$

Figure 11: States and Reaction rules of $NscATh$