Secure Protocol Composition

(Extended Abstract)

Anupam Datta Ante Derek John C. Mitchell Dusko Pavlovic

Computer Science Department
Stanford University
Stanford, CA 94305-9045
{danupam,aderek,jcm}@cs.stanford.edu

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

ABSTRACT

Modular composition of security mechanisms is complicated by the way that one mechanism may reveal information that interferes with the security of another. We develop methods for modular reasoning about security protocols, using before-after assertions and protocol invariants. The before-after assertions allow us to prove properties of a sequential composition of protocol steps and therefore enable construction of complex protocols from smaller sub-protocols. Invariants provide a mechanism for ensuring that sub-protocols which are individually secure do not interact insecurely when they are composed to construct a bigger protocol. The application of the method is demonstrated by giving modular formal proofs involving two standard protocols.

Categories and Subject Descriptors

F.3 [**Logics and meanings of programs**]: Specifying and Verifying and Reasoning about Programs

General Terms

Security, Theory, Verification

Keywords

Security Protocols, Protocol Composition, Logic, Invariants

1. INTRODUCTION

Modularity is a central problem in computer security and a proven challenge to many investigators (including [20, 21, 22, 23, 24]). In this paper, we explore modular construction of network protocols and present a system for reasoning about compound protocols from their parts. In general terms, we address two basic problems in modular security. The first may be called *additive combination* – we wish to combine protocol components in a way that accumulates security properties. For example, we may wish to combine a basic key exchange protocol with an authentication mechanism to produce a protocol for authenticated key exchange. The second basic problem is ensuring *nondestructive combination.* If two mechanisms are combined, each serving a separate purpose, then it is important to be sure that neither one degrades the security properties of the other. For example, if we add an alternative mode of operation to a protocol, then some party may initiate a session in one mode and simultaneously respond to another session in another mode, using the same public key or long-term key in both. Unless the modes are designed not to interfere, there may be an attack on the multi-mode protocol that would not arise if only one mode were possible. An interesting illustration of the significants of nondestructive combination is the construction in [16] which shows that for every security protocol there is another protocol that interacts with it insecurely.

Recognizing that many common network protocols are built using an accepted set of standard concepts, we have identified a set of basic components, protocol composition operations, protocol refinements, and protocol transformations for authentication and key exchange protocols. In [5], we characterize the structure of a family of key exchange protocols that includes Station-To-Station (STS), ISO-9798-3, Just Fast Keying (JFK) and related protocols, showing how all the protocols in this family may be derived systematically. We have also constructed systematic derivations of other families of protocols, in each case showing how a simple starting protocol may be extended, incrementally adding properties or optimizing performance in each step. While the derivation system seems a useful tool for developing and understanding protocols, we have not yet been able to prove that each derivation step is sound for all protocols where it could be applied. In this paper, we show how to prove correctness of additive and nondestructive combinations of protocol components.

Intuitively, additive combination is captured by a before-after formalism for reasoning about steps in protocol execution. Suppose P is a sequence of protocol steps, and ϕ and ψ are formulas asserting secrecy of some data, past actions of other principals, or other facts about a run of a protocol. The triple $\phi[P]_A\psi$ means that if ϕ is true before principal A does actions P, then ψ will be true afterwards. For example, the precondition might assert that A knows B's public key, the actions P allow A to receive a signed message and verify B's signature, and the postcondition may say that B sent the signed message that A received. The importance

FMSE'03, October 30, 2003, Washington, DC, USA.

of before-after assertions is that we can combine assertions about individual protocol steps to derive properties of a sequence of steps: if $\phi[P]_A\psi$ and $\psi[P']_A\theta$, then $\phi[PP']_A\theta$. For example, an assertion assuming that keys have been successfully distributed can be combined with steps that do key distribution to prove properties of a protocol that distributes keys and uses them.

We ensure nondestructive combination, which is useful for reasoning about running older versions of a protocol concurrently with current versions (e.g., SSL 2.0 and SSL 3.0) and for verifying protocols like IKE [13] which contain a large number of sub-protocols, using invariance assertions. The central assertion in our reasoning system, $\Gamma \vdash \phi[P]_A\psi$, says that in any protocol satisfying the invariant Γ, the before-after assertion $\phi[P]_A\psi$ holds in any run (regardless of any actions by any dishonest attacker). Typically, our invariants are statements about principals that follow the rules of a protocol, as are the final conclusions. For example, an invariant may state that every honest principal maintains secrecy of its keys, where "honest" means simply that the principal only performs actions that are given by the protocol. A conclusion in such a protocol may be that if Bob is honest (so no one else knows his key), then after Alice sends and receives certain messages, Alice knows that she has communicated with Bob. Under the specific conditions described in this paper, nondestructive combination occurs when two protocols are combined and neither violates the invariants of the other.

As informally described, "additive combination" and "nondestructive combination" may seem like overlapping concepts, at least to the degree that additive combination assumes that the added steps do not destroy any security properties. In our logic, we factor the two concepts into two separate notions, one for adding steps to a protocol under some assumed invariants, and another for showing that a combination of protocol steps preserves a set of invariants. More specifically, if we want to add an authentication step to a protocol, we first show that the additional step preserves the same needed invariants. Then, under the assumption that invariants are preserved, we combine properties guaranteed by separate steps. There is some synergy in this approach, since the logical principles used to prove an invariant are the same as those used to prove protocol properties from a given set of invariants.

To show the utility of our logic for practical protocols, we present three examples. Example 6.1 is a formal correctness proof of *ISO-9798-3* , constructed by composing proofs of two parts, a standard signature-based challenge response protocol [28], and a Diffie-Hellman key exchange protocol [6]. Example 6.2 proves correctness of a protocol that exchanges certificates to establish public keys, and then uses public-key cryptography and nonces to establish a shared secret. Example 6.3 combines the two, showing that running any number of instances of *ISO-9798-3* and the Needham-Schroeder public-key protocol in parallel preserves the security properties of both. While the formal proofs are somewhat lengthy when written out in full detail, as in any formal proof system, the proof structure takes advantage of composition ideas and illustrates the power of a modular approach. Although the assertions we prove only mention steps of the protocol, the logic is sound in a stronger sense: each provable assertion about an action or sequence of actions holds in any run of the protocol that contains the given actions and arbitrary additional actions by any number of additional principals and malicious attacker(s). This "implicit attacker" method lets us prove security properties of protocols under attack, while reasoning only about the sequence of actions taken by honest parties in the protocol.

It is well known that many natural security properties (e.g., noninterference) are not preserved either under composition or under refinement. This has been extensively explored using trace-based modeling techniques [20, 21, 22, 23, 24], using properties that are not first-order predicates over traces, but second-order predicates over sets of traces that may not have closure properties corresponding to composition and refinement. In contrast, our security properties are safety properties over sets of traces that satisfy safety invariants, thus avoiding negative results about composability.

The rest of the paper is organized as follows. Section 2 discusses the process calculus that we use for defining the steps of a protocol. The syntax and semantics of the core protocol logic is presented in Section 3. The proof system is presented in Section 4. Section 5 describes the extensions to the core proof system used to reason about protocol composition. In Section 6, we illustrate applications of the logic. In Section 7, we describe previous work on protocol composition [4, 11, 12, 14, 16, 18, 27, 36] and discuss how our formalization can be used to explain some of those results. Finally, in Section 8, we present our conclusions and propose some themes for future work.

2. CORD CALCULUS

Cords [9] are the formalism we use to represent protocols and their parts. They form an action calculus [29, 30, 33], based on π-calculus [31], and related to spi-calculus [1]. The cords formalism is also similar to the approach of the Chemical Abstract Machine formalism [3], in that the communication actions can be viewed as reactions between "molecules". Cord calculus serves as a simple "protocol programming language" which supports our Floyd-Hoare style logical annotations, and verifications in an axiomatic semantics. Cord calculus is presented in [9]. In order to make this paper self-contained, a brief summary is included in Appendix A.

In this section, we illustrate with an example how protocols are represented in cord calculus. Figure 1 shows the *ISO-9798-3* protocol [15] in the informal arrows-and-messages notation which is commonly used to describe security protocols. The same protocol is written out in the language of cord calculus in Figure 2. The common point between the two is that a protocol is described by listing out the sequence of actions that honest parties would execute in a run. In this example, the protocol consists of two roles, the initiator role and the responder role. The sequence of actions in the initiator role are given by the cord **A** in Figure 2. In words, the actions are: generate a fresh random number; send a message with the Diffie-Hellman exponential of that number to the peer, $\hat{B}$; receive a message with source address $\hat{B}$; verify that the message contains $\hat{B}$'s signature over data in the expected format; and finally, send another message to $\hat{B}$ with the initiator's signature over the Diffie-Hellman exponential that she sent in the first message, the data she received from $\hat{B}$ (which should be a Diffie-Hellman exponential generated by $\hat{B}$) and $\hat{B}$'s identity. The notations (νx), $\langle t\rangle$, (x) refer respectively to the actions of nonce generation, sending a term and receiving a message. Here, a message is assumed to be of the form: (source, destination,

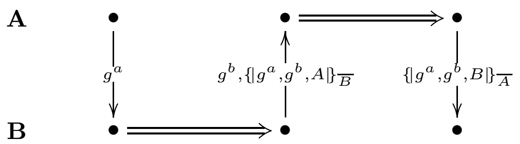

Figure 1: *ISO-9798-3* as arrows-and-messages

content). Detailed syntax of cord calculus is presented in Appendix A.

3. A PROTOCOL LOGIC

3.1 Syntax

The formulas of the logic are given by the grammar in Table 1, where ρ may be any role, written using the notation of cord calculus. Here, t and P denote a term and a process respectively. We use the word *process* to refer to a principal executing an instance of a role, e.g., Alice taking part in a session of a protocol in the initiator role. As a notational convention, we use X to refer to a process belonging to principal $\hat{X}$. We use ϕ and ψ to indicate predicate formulas, and m to indicate a generic term we call a "message". m is of the form (source, destination, protocol-identifier, content), i.e., each message has source and destination fields and a unique protocol identifier besides the contents. Note that the source field of a message may not be the same as the actual sender of the message since the intruder can spoof the source address. Also, the principal identified by the destination field may not receive the message since the intruder can intercept messages. The source and destination fields in the message are useful while proving authentication properties of protocols. When an honest principal sends out a message, the source field identifies her and the destination field identifies the intended recipient. Our formalization of authentication is based on the notion of matching records of runs [7] which requires that whenever $\hat{A}$ and $\hat{B}$ accept each other's identities at the end of a run, their records of the run should match, i.e., each message that $\hat{A}$ sent was received by $\hat{B}$ and vice versa, each send event happened before the corresponding receive event, and moreover the messages sent by each principal ($\hat{A}$ or $\hat{B}$) appear in the same order in both the records. Including the source and destination fields in the message allows us to match up send-receive actions. Since in this paper, we reason about correctness of a protocol in an environment in which other protocols may be executing concurrently, it is important that when $\hat{A}$ and $\hat{B}$ accept each other's identities, they also agree on which protocol they have successfully completed with the other. One way to extend the matching histories characterization to capture this requirement is by adding protocol identifiers to messages. Now if $\hat{A}$ and $\hat{B}$ have matching histories at the end of a run, not only do they agree on the source, destination and content of each message, but also on which protocol this run is an instance of.

Most protocol proofs use formulas of the form $\theta[P]_X\phi$, which means that after actions P are executed in process X, starting from a state where formula θ is true, formula ϕ is true about the resulting state of X. Here are the informal interpretations of the predicates, with the basis for defining precise semantics discussed in the next section.

The formula $\mathsf{Has}(X, x)$ means that principal $\hat{X}$ possesses information x in the process X. This is "possesses" in the limited sense of having either generated the data or received it in the clear or received it under encryption where the decryption key is known. The formula $\mathsf{Send}(X, m)$ means that the last action in a run of the protocol corresponds to principal $\hat{X}$ sending message m in in the process X. $\mathsf{Receive}(X, m)$, $\mathsf{New}(X, t)$, $\mathsf{Decrypt}(X, t)$, and $\mathsf{Verify}(X, t)$ are similarly associated with the receive, new, decrypt and signature verification actions of a protocol. $\mathsf{Fresh}(X, t)$ means that the term t generated in X is "fresh" in the sense that no one else has seen any term containing t as a subterm. Typically, a fresh term will be a nonce and freshness will be used to reason about the temporal ordering of actions in runs of a protocol. This form of reasoning is useful in proving authentication properties of protocols. The formula $\mathsf{Honest}(\hat{X})$ means that the actions of principal $\hat{X}$ in the current run are precisely an interleaving of initial segments of traces of a set of roles of the protocol. In other words, $\hat{X}$ assumes some set of roles and does exactly the actions prescribed by them. $\mathsf{Contains}(t_1, t_2)$ means that t_2 is a subterm of t_1. This predicate helps us identify the components of a message. The two temporal operators $\diamondminus$ and $\ominus$ have the same meaning as in Linear Temporal Logic [19]. Since we view a run as a linear sequence of states, $\diamondminus\ \phi$ means that in some state in the past ϕ holds, whereas $\ominus\ \phi$ means that in the previous state ϕ holds. The predicate $\mathsf{After}(\mathsf{a}_1,\ \mathsf{a}_2)$ means that the action a_2 happened after the action a_1 in a run. In this paper, we restrict attention to protocol roles in which all actions are unique. In particular, a principal executing a role of a protocol does not send the same message twice. This seems like a reasonable assumption since even if she did send two messages with the same content, she would probably distinguish the two by using message sequence numbers or a similar mechanism. The technical benefit of this assumption is that After becomes a transitive relation for actions executed by honest principals.

We note here that the temporal operator $\diamondminus$ and some of the predicates (Send, Receive) bear semblance to those used in NPATRL [35], the temporal requirements language for the NRL Protocol Analyzer [25, 26]. However, while NPATRL is used for specifying protocol requirements, our logic is also used to infer properties of protocols. This leads to essential semantical differences.

3.2 Semantics

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

$$\mathbf{A} = [\quad (\nu a) \quad \langle \hat{A}, \hat{B}, g^a \rangle (\hat{B}, \hat{A}, n, \{\!|g^a, n, \hat{A}|\!\}_{\overline{B}} / \hat{B}, \hat{A}, y, z) \quad (z / \{\!|g^a, y, \hat{A}|\!\}_{B}) \quad \langle \hat{A}, \hat{B}, \{\!|g^a, y, \hat{B}|\!\}_{\overline{A}} \rangle]$$

$$\mathbf{B} = [(\hat{X}, \hat{B}, m) \quad (\nu b) \quad \langle \hat{B}, \hat{X}, g^b, \{\!|m, g^b, \hat{X}|\!\}_{\overline{B}} \rangle \quad (\hat{X}, \hat{B}, \{\!|m, g^b, \hat{B}|\!\}_{\overline{X}})]$$

Figure 2: *ISO-9798-3* as a cord space

Action formulas

$\mathsf{a} \ ::= \ \mathsf{Send}(P, m) \,|\, \mathsf{Receive}(P, m) \,|\, \mathsf{New}(P, t) \,|\, \mathsf{Decrypt}(P, t) \,|\, \mathsf{Verify}(P, t)$

Formulas

$\phi \ ::= \ \mathsf{a} \,|\, \mathsf{Has}(P, t) \,|\, \mathsf{Fresh}(P, t) \,|\, \mathsf{Honest}(N) \,|\, \mathsf{Contains}(t_1, t_2) \,|\, \phi \wedge \phi \,|\, \neg\phi \,|\, \exists x.\phi \,|\, \Diamond\!\!\!\!-\, \phi \,|\, \ominus\, \phi$

Modal forms

$\Psi \ ::= \ \rho\, \phi \,|\, \phi\, \rho\, \phi$

Table 1: Syntax of the logic

---

of all initial configurations of protocol $\mathcal{Q}$, each including a possible intruder cord. Let $\mathsf{Runs}(\bar{\mathcal{Q}})$ be the set of all runs of protocol $\mathcal{Q}$ with intruder, each a sequence of reaction steps within a cord space. If ϕ has free variables, then $Q, R \models \phi$ if we have $Q, R \models \sigma\phi$ for all substitutions σ that eliminate all the free variables in ϕ. We write $\mathcal{Q} \models \phi$ if $\mathcal{Q}, R \models \phi$ for all $R \in \mathsf{Runs}(\bar{\mathcal{Q}})$.

The inductive definition of $Q, R \models \phi$ is presented in Appendix B. The main idea is to view a run as a sequence of reaction steps within a cord space. Each reaction step corresponds to a principal executing an action. It therefore becomes possible to assert whether a particular action occurred in a given run and also to make assertions about the temporal ordering of the actions. An alternative view, similar to the execution model used in defining Linear Temporal Logic (LTL) semantics, is to think of a run as a linear sequence of states. Transition from one state to the next is effected by an action carried out by some principal in some role. Associating that action with the state that the system ends up in as a consequence, allows us to use the well-understood terminology of LTL in our logic. A formula is true in a run if it is true in the last state of that run. An action formula a is therefore true in a run if it is the last action in that run. On the other hand, a past formula $\Diamond\!\!\!\!-\, \mathsf{a}$ is true if in the past the action formula a was true in some state, i.e., if the action had occurred in the past.

4. PROOF SYSTEM

4.1 Axioms and Inference Rules

A representative fragment of the axioms and inference rules in the proof system are collected in Table 2. For expositional convenience, we divide the axioms into four groups.

The axioms about protocol actions state properties that hold in the state reached by executing one of the actions in a state in which formula ϕ holds. Note that the a in axiom **AA1** is any one of the 5 actions and a is the corresponding predicate in the logic. **AA1** states that if a principal has executed an action in some role, then the corresponding predicate asserting that the action had occurred in the past is true. Also, in the previous state, ϕ is true. If process X generates a new value n and does no further actions, then **AN3** says that n is fresh in that state.

VER and **SEC** respectively refer to the unforgeability of signatures and the need to possess the private key in order to decrypt a message encrypted with the corresponding public key. The additional condition requiring principal $\hat{X}$ to be honest guarantees that the intruder is not in possession of the private keys. These axioms (together with a few more axioms not described in this extended abstract) provide an abstraction of the standard Dolev-Yao intruder model [8].

Axioms **P1**, **P2**, and **P3** capture the fact that most predicates are preserved by additional actions. For example, if in some state $\mathsf{Has}(X, n)$ holds, then it continues to hold, when X executes additional actions. Intuitively, if a process possesses some information at a point in a run, then she remembers it for the rest of the run. Note, however, that the Fresh predicate is not preserved if the freshly generated value n is sent out in a message (see **F**).

The PLTL axioms **T1**, **T2**, and **T3** allow reasoning about the temporal ordering of actions. This turns out to be useful in proving authentication properties of protocols.

The inference rules follow naturally from the semantics. **G2** is exactly of the same form as the rule of consequence in Hoare Logic. **G3** and **TGEN** are respectively the modal and temporal generalization rules.

4.2 The Honesty Rule

The honesty rule is essentially an invariance rule for proving properties of all roles of a protocol. It is similar to the basic invariance rule of LTL [19]. 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. In order to do so, Alice may assume that Bob is honest and derive consequences from this assumption. 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.

Axioms about Protocol Actions:

$$\begin{array}{ll} \textbf{AA1} & \phi[a]_X \ \diamondminus\, (\mathsf{a} \wedge \ominus\phi) \\ \textbf{AN3} & \phi[(\nu n)]_X \ \mathsf{Fresh}(X, n) \end{array}$$

Axioms Capturing Dolev-Yao Model:

$$\begin{array}{ll} \textbf{SEC} & \mathsf{Honest}(\hat{X}) \wedge \diamondminus\ \mathsf{Decrypt}(Y, \{\!|n|\!\}_X) \supset (\hat{Y} = \hat{X}) \\ \textbf{VER} & \mathsf{Honest}(\hat{X}) \wedge \diamondminus\ \mathsf{Verify}(Y, \{\!|n|\!\}_{\overline{X}}) \supset \\ & \exists X.\exists m.(\diamondminus\ \mathsf{Send}(X, m) \wedge \mathsf{Contains}(m, \{\!|n|\!\}_{\overline{X}})) \end{array}$$

Preservation Axioms: (For $\mathsf{Persist} \in \{\mathsf{Has}, \diamondminus\ \phi\},$)

$$\begin{array}{rl} \textbf{P1} & \mathsf{Persist}(X, t)[a]_X \mathsf{Persist}(X, t) \\ \textbf{P2} & \mathsf{Fresh}(X, t)[a]_X \mathsf{Fresh}(X, t), where (t \not\subseteq a) \vee (a \neq \langle m \rangle) \\ \textbf{P3} & \mathsf{HasAlone}(X, t)[a]_X \mathsf{HasAlone}(X, t), where (t \not\subseteq_v a) \vee (a \neq \langle m \rangle) \\ \textbf{F} & \mathsf{Fresh}(X, t)[\langle m \rangle]_X \neg \mathsf{Fresh}(X, t), where (t \subseteq m) \end{array}$$

PLTL Axioms:

$$\begin{array}{ll} \textbf{T1} & \diamondminus\ (\phi \wedge \psi) \supset (\diamondminus\ \phi \wedge \diamondminus\ \psi) \\ \textbf{T2} & \diamondminus\ (\phi \vee \psi) \supset (\diamondminus\ \phi \vee \diamondminus\ \psi) \\ \textbf{T3} & \ominus\neg\phi \leftrightarrow \neg\ominus\phi \end{array}$$

Inference Rules:

$$\frac{\theta[P]_X \phi \quad \theta[P]_X \psi}{\theta[P]_X \phi \wedge \psi}\ \textbf{G1} \qquad \frac{\theta[P]_X \phi \quad \theta' \supset \theta \quad \phi \supset \phi'}{\theta'[P]_X \phi'}\ \textbf{G2}$$

$$\frac{\phi}{\theta[P]_X \phi}\ \textbf{G3} \qquad \frac{\phi}{\neg \diamondminus \neg \phi}\ \textbf{TGEN}$$

Table 2: Fragment of the Proof System

Recall that a protocol $\mathcal{Q}$ is a set of roles, $\mathcal{Q} = \{\rho_1, \rho_2, \ldots, \rho_k\}$. If $\rho \in \mathcal{Q}$ is a role of protocol $\mathcal{Q}$, we write $P \epsilon BS(\rho)$ if P is a continuous segment of the actions of role ρ such that (a) P is the empty sequence; or (b) P starts at the beginning of ρ and goes upto the first receive ; or (c) P starts from a receive action and goes upto the next receive action; or (d) P starts from the last receive action and continues till the end of the role. We call such a P a *basic sequence* of role ρ. The reason for only considering segments starting from a read and continuing till the next read is that if a role contains a send, the send may be done asynchronously without waiting for another role to receive. Therefore, we can assume without loss of generality that the only "pausing" states of a principal are those where the role is waiting for input. If a role calls for a message to be sent, then we dictate that the principal following this role must complete the send before pausing.

Since the honesty rule depends on the protocol, we write $\mathcal{Q} \vdash \theta[P]\phi$ if $\theta[P]\phi$ 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 as follows.

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

In words, if ϕ holds at the beginning of every role of $\mathcal{Q}$ and is preserved by all its basic sequences, then every honest principal executing protocol $\mathcal{Q}$ must satisfy ϕ. The side condition prevents free variables in the conclusion $\mathsf{Honest}(\hat{X}) \supset \phi$ from becoming bound in any hypothesis. Intuitively, since ϕ holds in the initial state and is preserved by all basic sequences, it holds at all pausing states of any run.

5. FORMALIZING PROTOCOL COMPOSITION

The machinery developed till this point only allows protocols to be analyzed in isolation. In this section, we extend the deductive system to reason about protocol composition. In doing so, we address the two ways in which composition problems can arise in security protocol analysis. Both arise out of complexity. In one case, we wish to gain control of complexity by building up a proof of correctness of a complex protocol from proofs of correctness of its component sub-protocols. In the other, we want to avoid insecure interactions between different protocols or different versions of the same protocol that may be operating over the same network.

The protocol composition rules are collected in Table 3. Γ denotes a set of formulas which we refer to as *environment invariants*. The idea is to capture, using these formulas, the constraints that the environment must satisfy in order to enable a specific protocol to retain its security property. Typically, these constraints will impose restrictions on the actions of the honest principals, i.e., the principals who are faithfully executing one or more of the protocols running in the environment. We write $\Gamma \vdash \phi$ if ϕ is provable using the formulas in Γ and the axioms and proof rules of the deductive system. The semantic entailment, $\Gamma \models \phi$, is defined in Appendix B. Essentially, it says, that in any run in which the invariants in Γ hold, the formula ϕ is true.

The weakening rule, **W**, states that a formula ϕ which is provable from a set of hypotheses, Γ, remains provable if additional formulas are added to the set of hypotheses. The protocol composition rule **C1** gives us a way of sequentially composing two roles P and P' when the logical formula guaranteed by the execution of P, i.e., the post-condition of P, matches the pre-condition required in order to ensure that P' achieves some property. As before, Γ denotes a set of hypotheses which are used in proving the properties of the protocols. This form of reasoning allows a proof of correctness of a protocol to be built up incrementally from a proof of its component sub-protocols. The other composition rule **C2** states that if the environment invariants hold for two protocols, $\mathcal{Q}$ and $\mathcal{Q}'$, then the invariants also hold for their composition. This rule is sound if the formulas in Γ capture trace-based invariants, which are proved using the honesty rule in our deductive system.

If $\mathcal{Q}$ and $\mathcal{Q}'$ are protocols, then we define $\mathcal{Q} \circ \mathcal{Q}'$ to be any protocol such that every role ρ in $\mathcal{Q} \circ \mathcal{Q}'$ is a concatenation of basic sequences of roles in $\mathcal{Q}$ or $\mathcal{Q}'$. Therefore, every $\rho \in \mathcal{Q} \circ \mathcal{Q}'$ can be written as $\rho = \rho_1 \rho_2 \ldots \rho_n$ where every ρ_i is a basic sequence of a role in $\mathcal{Q}$ or $\mathcal{Q}'$. Note that sequential and parallel composition arise as special cases of this general composition operation.

Our general methodology for proving protocol composition results involves the following steps:

1. Prove separately the security properties of protocols $\mathcal{Q}$ and $\mathcal{Q}'$.

2. Identify the set of environment invariants used in the two proofs, Γ and Γ'. The formulas included in these sets will typically be the formulas in the two proofs, which were proved using the honesty rule.

3. Apply the weakening rule so that $\Gamma \cup \Gamma'$ represents the set of environment invariants that will be used while applying the composition rules **C1** and **C2**. This step is required in case of sequential composition or if we want to prove that the properties of both $\mathcal{Q}$ and $\mathcal{Q}'$ are preserved by the composition process. However, if the goal is to just prove that the properties of $\mathcal{Q}$ are preserved, then the set of environment invariants that will be used while applying **C1** and **C2** will simply be Γ.

4. When the post-condition of a role of $\mathcal{Q}$ matches the pre-condition of the corresponding role of $\mathcal{Q}'$, sequentially compose the two roles by applying rule **C1**. This step is required only in the case of sequential composition.

5. Prove that the environment invariants used in proving the properties of the protocols, $\Gamma \cup \Gamma'$, hold for both the protocols. Since $\mathcal{Q} \vdash \Gamma$ was already proved in Step 1, in this step, it is sufficient to show that $\mathcal{Q} \vdash \Gamma'$ and similarly that $\mathcal{Q}' \vdash \Gamma$. If Step 3 was skipped, then it is sufficient to just show that $\mathcal{Q}' \vdash \Gamma$.

Note that in proving a composition result (whether sequential or parallel), we always prove that the two protocols under consideration respect each other's invariants (Step 5), i.e., that they do not interact insecurely. In addition, while proving that two protocols can be sequentially composed, we require that the post-condition of the first matches the pre-condition of the second (Step 4). Thus, in proving a sequential composition result, we address the two central

$$\frac{\Gamma \vdash \phi}{\Gamma \cup \Gamma' \vdash \phi}\,\mathbf{W} \qquad \frac{\Gamma \vdash \phi_1[P]_A\phi_2 \qquad \Gamma \vdash \phi_2[P']_A\phi_3}{\Gamma \vdash \phi_1[P;P']_A\phi_3}\,\mathbf{C1} \qquad \frac{\mathcal{Q} \vdash \Gamma \qquad \mathcal{Q}' \vdash \Gamma}{\mathcal{Q} \circ \mathcal{Q}' \vdash \Gamma}\,\mathbf{C2}$$

Table 3: Composition Rules

problems of compositional protocol analysis mentioned in the beginning of the section.

6. EXAMPLES OF PROTOCOL COMPOSITION

In this section, we illustrate the use of the methodology outlined in the previous section, by presenting modular proofs of two standard protocols, *ISO-9798-3* [15] and *NSL* [32, 17]. The parallel composition of these two protocols is also proved secure. Due to space constraints, we only discuss the the proof of *ISO-9798-3* in detail, and sketch an outline of the *NSL* proof and the proof of correctness of their parallel composition.

EXAMPLE 6.1. *The ISO-9798-3 Protocol*

The *ISO-9798-3* protocol is constructed by a sequential composition of a protocol based on the Diffie-Hellman key exchange protocol and the standard signature-based challenge-response protocol. Here, we prove the key secrecy property of the Diffie-Hellman protocol and the mutual authentication property of the challenge-response protocol. We then prove that the *ISO-9798-3* protocol can be used to establish an authenticated shared secret by composing the correctness proofs of these two protocols.

Challenge Response Protocol, CR*:.* Our formulation of authentication is based on the concept of *matching conversations* [2] and is similar to the idea of proving authentication using *correspondence assertions* [37]. The same basic idea is also presented in [7] where it is referred to as *matching records of runs*. Simply put, it requires that whenever $\hat{A}$ and $\hat{B}$ accept each other's identities at the end of a run, their records of the run *match*, i.e., each message that $\hat{A}$ sent was received by $\hat{B}$ and vice versa, each send event happened before the corresponding receive event, and moreover the messages sent by each principal ($\hat{A}$ or $\hat{B}$) appear in the same order in both the records.

The complete formal proof of the mutual authentication property guaranteed by executing the CR protocol is deferred to the full version of the paper. Here, we just describe the property proved and provide some intuition about the structure of the proof. The final property proved about the initiator role (referred as ϕ_{auth} henceforth) is of the form: *precondition [actions] postcondition*, where:

$$\begin{array}{lcl}
\mathit{precondition} & = & \mathsf{Fresh}(A, m) \\
\mathit{actions} & = & [\langle \hat{A}, \hat{B}, m\rangle(\hat{B}, \hat{A}, n, \{\!|m, n, \hat{A}|\!\}_{\overline{B}}/\hat{B}, \hat{A}, y, z) \\
 & & \quad (z/\{\!|m, y, \hat{A}|\!\}_{B})\langle \hat{A}, \hat{B}, \{\!|m, y, \hat{B}|\!\}_{\overline{A}}\rangle]_A \\
\mathit{postcondition} & = & \mathsf{Honest}(\hat{B}) \supset \mathsf{ActionsInOrder}(\\
 & & \quad \mathsf{Send}(A, \{\hat{A}, \hat{B}, m\}), \\
 & & \quad \mathsf{Receive}(B, \{\hat{A}, \hat{B}, m\}), \\
 & & \quad \mathsf{Send}(B, \{\hat{B}, \hat{A}, \{n, \{\!|m, n, \hat{A}|\!\}_{\overline{B}}\}\}), \\
 & & \quad \mathsf{Receive}(A, \{\hat{B}, \hat{A}, \{n, \{\!|m, n, \hat{A}|\!\}_{\overline{B}}\}\}))
\end{array}$$

Here, the predicate $\mathsf{ActionsInOrder}(\mathsf{a}_1, \mathsf{a}_2, \ldots, \mathsf{a}_n)$ means that the actions $a_1, a_2, \ldots, a_n$ were executed in that order. Intuitively, this formula means that after executing the actions in the initiator role purportedly with $\hat{B}$, $\hat{A}$ is guaranteed that her record of the run matches that of $\hat{B}$, provided that $\hat{B}$ is honest. The set of environmental invariants used in this proof, Γ, contains only one formula,

$$\begin{array}{l}
\Gamma \; = \; \{ \; \mathsf{Honest}(\hat{B}) \supset (\\
\quad (\; \diamondminus \, \mathsf{Send}(B, m_0) \wedge \mathsf{Contains}(m_0, \{\!|m, n, \hat{A}|\!\}_{\overline{B}}) \wedge \\
\quad \neg \diamondminus \, \mathsf{Fresh}(B, m) \;) \supset \; (\\
\qquad m_0 = \{\hat{B}, \hat{A}, \{n, \{\!|m, n, \hat{A}|\!\}_{\overline{B}}\}\} \wedge \\
\qquad \diamondminus \, (\mathsf{Send}(B, \{\hat{B}, \hat{A}, \{n, \{\!|m, n, \hat{A}|\!\}_{\overline{B}}\}\}) \wedge \\
\qquad \ominus \mathsf{Fresh}(B, n)) \wedge \\
\qquad \mathsf{ActionsInOrder}(\mathsf{Receive}(B, \{\hat{A}, \hat{B}, m\}), \\
\qquad \quad \mathsf{Send}(B, \{\hat{B}, \hat{A}, \{n, \{\!|m, n, \hat{A}|\!\}_{\overline{B}}\}\})) \\
\quad) \;) \; \}
\end{array}$$

Intuitively, this invariant states that whenever honest $\hat{B}$ signs a term which is a triple with the third component $\hat{A}$, and the first component was not freshly generated by $\hat{B}$, then it is the case that this signature was sent as part of the second message of the CR protocol. (Note that each message sent and received has the protocol-id in it. We omit these to improve readability).

Base Diffie Hellman Protocol, DH_0*:.* The DH_0 protocol involves generating a fresh random number and computing its Diffie-Hellman exponential. It is therefore the initial part of the standard Diffie-Hellman key exchange protocol. In order to reason about the security property of this protocol, the term language and the protocol logic have to be enriched to allow reasoning about Diffie-Hellman computation. The terms $g(a)$ and $h(a, b)$, respectively representing the Diffie-Hellman exponential $g^a \, mod \, p$ and the Diffie-Hellman secret $g^{ab} \, mod \, p$, are added to the term language. To improve readability, we will use g^a and g^{ab} instead of $g(a)$ and $h(a, b)$. Table 4 presents the rules specific to the way that Diffie-Hellman secrets are computed. **DH1** captures the way that a Diffie-Hellman secret is computed from an exponent and

DH1	$\mathsf{Has}(X, a) \wedge \mathsf{Has}(X, g^b) \supset \mathsf{Has}(X, g^{ab})$
DH2	$\mathsf{Has}(X, g^{ab}) \supset \mathsf{Has}(X, g^{ba})$
DH3	$\mathsf{Has}(X, g^{ab}) \supset (\mathsf{Has}(X, a) \wedge \mathsf{Has}(X, g^b)) \vee$
	$(\mathsf{Has}(X, b) \wedge \mathsf{Has}(X, g^a))$
DH4	$\mathsf{Fresh}(X, a) \supset \mathsf{Fresh}(X, g^a)$

Table 4: Rules for Diffie-Hellman key exchange

an exponential. **DH2** captures the commutativity of exponentiation. **DH3** captures the hardness of the discrete log problem by stating that in order to compute the Diffie-Hellman secret, at least one exponent and the other exponential needs to be known. **DH4** captures the intuition that if a is fresh at some point of a run, then g^a is also fresh at that point.

The property of the initiator role of the DH_0 protocol is given by the formula below.

$$[(\nu a)]_A \, \mathsf{HasAlone}(A, a) \wedge \mathsf{Fresh}(A, g^a)$$

This formula follows easily from the axioms and rules of the logic. It states that after carrying out the initiator role of DH_0, A possesses a fresh Diffie-Hellman exponential g^a and is the only one who possesses the exponent a. This property will be useful in proving the secrecy condition of the *ISO-9798-3* protocol. The set of environmental invariants used in this proof, Γ', is empty.

Composing the Protocols:. We now prove the security properties of the *ISO-9798-3* protocol by composing the correctness proofs of DH_0 and CR. In doing so, we follow the general methodology for proving composition results outlined in Section 5. Let us go back and look at the form of the logical formulas characterizing the initiator roles of DH_0 and CR:

$$\begin{array}{ll} DH_0 & : \Gamma' \vdash [\mathbf{Init_{DH_0}}]_A \, \mathsf{Fresh}(A, g^a) \\ CR & : \Gamma \vdash \mathsf{Fresh}(A, m) \, [\mathbf{Init_{CR}}]_A \, \phi_{auth} \end{array}$$

At this point, steps 1 and 2 of the general methodology have already been carried out. We now apply the weakening rule to both the formulas above (step 3). Since Γ' is empty, $\Gamma \cup \Gamma'$ is simply Γ. Note that the post-condition of DH_0 matches the pre-condition of CR. We can therefore compose the two formulas by applying the composition rule **C1** (step 4). The resulting formula is:

$$\textit{ISO-9798-3 (auth.)} : \Gamma \vdash [\mathbf{Init_{DH_0}}; \mathbf{Init_{CR}}]_A \, \phi_{auth}$$

The result of composing the two roles is that the freshly generated Diffie-Hellman exponential is substituted for the nonce in the challenge-response cord. The resulting role is precisely the initiator role of the *ISO-9798-3* protocol. The formula above states that the mutual authentication property of CR is preserved by the composition process assuming that the environmental invariants in Γ are still satisfied. Finally, using the honesty rule, it is easily proved that DH_0 respects the environmental invariants in Γ (step 5). Therefore, by applying the composition rule **C2**, we conclude that the sequential composition of DH_0 and CR, which is *ISO-9798-3* , respects the invariants in Γ. This completes the composition proof for the mutual authentication property.

The other main step involves proving that the secrecy property of DH_0 is preserved by CR, since the CR protocol does not reveal the Diffie-Hellman exponents.

$$\begin{array}{ll} DH_0 & : \vdash [\mathbf{Init_{DH_0}}]_A \, \mathsf{HasAlone}(A, a) \\ CR' & : \vdash \mathsf{HasAlone}(A, a) \, [\mathbf{Init_{CR'}}]_A \, \mathsf{HasAlone}(A, a) \end{array}$$

Here, CR' is the same protocol as CR except that g^a is substituted for the nonce m. Therefore, by applying the composition rule **C1** again, we have the secrecy condition for the *ISO-9798-3* protocol:

$$\begin{array}{l} \textit{ISO-9798-3 (secrecy)} : \\ \quad \vdash [\mathbf{Init_{DH_0}}; \mathbf{Init_{CR}}]_A \, \mathsf{HasAlone}(A, a) \end{array}$$

Since the set of environment invariants is empty, steps 3 and 5 follow trivially. The rest of the proof uses properties of the Diffie-Hellman method of secret computation to prove the following logical formula:

$$\begin{array}{l} \textit{ISO-9798-3 (shared-secret)} : [\mathbf{Init_{DH_0}}; \mathbf{Init_{CR}}]_A \, \mathsf{Honest}(\hat{\mathsf{B}}) \supset \\ \quad (n = g^{b_0} \wedge \mathsf{Has}(A, g^{ab_0}) \wedge (\mathsf{Has}(X, g^{ab_0}) \supset \\ \quad\quad (X = A \vee X = B))) \end{array}$$

Intuitively, the property proved is that if $\hat{B}$ is honest, then $\hat{A}$ and $\hat{B}$ are the only people who know the Diffie-Hellman secret g^{ab}. In other words, the *ISO-9798-3* protocol can be used to compute an authenticated shared secret. The complete formal proof is omitted in this extended abstract.

EXAMPLE 6.2. *The NSL Protocol*

The 7 message *NSL* protocol can be proved correct by sequential composition of two protocols, which we refer to as *NSL-init* and *NSL-base* . By running *NSL-init* , a principal obtains the public key certificate of a peer from the server. If a principal possesses a peer's public key certificate, she can run *NSL-base* with him and set up an authenticated shared secret. In our formalism, the postcondition of *NSL-init* is that the principal knows a peer's public key certificate. Also, with a precondition capturing the same property, *NSL-base* has a postcondition stating that the two principals possess a shared secret. Thus, the two protocols can be composed using the composition rule **C1**. The resulting protocol is *NSL* . Moreover, it can be proved that the two protocols respect each other's invariants, allowing us to conclude that the *NSL* protocol can be used to set up a shared secret. A part of this proof appears in [9].

Example 6.3. *Parallel Composition of ISO-9798-3 and NSL*

Since *ISO-9798-3* and *NSL* respect each other's invariants, their parallel composition can also be proved secure using our formalism. The main insight from this proof is that if the authenticators of two protocols (which are individually secure) cannot be confused with each other, then their composition is secure. Disjoint encryption, which has been suggested as a design principle to avoid insecure interaction between protocols [4, 12], appears to be a special case of this more general principle.

7. RELATED WORK

Early work on the protocol composition problem concentrated on designing protocols that would be guaranteed to compose with any other protocol. This led to rather stringent constraints on protocols: in essence, they required the fail-stop property [11] or something very similar to it [14]. Since real-world protocols are not designed in this manner, these approaches did not have much practical application. More recent work has therefore focussed on reducing the amount of work that is required to show that protocols are composable. Meadows, in her analysis of the IKE protocol suite using the NRL Protocol Analyzer [27], proved that the different sub-protocols did not interact insecurely with each other by restricting attention to only those parts of the sub-protocols, which had a chance of subverting each other's security goals. Independently, Thayer, Herzog and Guttman used a similar insight to develop a technique for proving composition results using their strand space model [36]. Their technique consisted in showing that a set of terms generated by one protocol can never be accepted by principals executing the other protocol. The techniques used for choosing the set of terms, however, is specific to the protocols in [10]. A somewhat different approach is used by Lynch [18] to prove that the composition of a simple shared key communication protocol and the Diffie-Hellman key distribution protocol is secure. Her model uses I/O automata and the protocols are shown to compose if adversaries are only passive eavesdroppers.

In a recent paper [4], Canetti, Meadows and Syverson, revisit the protocol composition problem. They show how the interaction between a protocol and its environment can have a major effect on the security properties of the protocol. In particular, they demonstrate a number of attacks on published and widely used protocols that are not feasible against the protocol running in isolation but become feasible in some environments. The main question that this study leaves open is: how should the environment be constrained so that it does not subvert the security goals of a protocol? The authors put forward some rules of thumb that could be useful in answering this question. Of these, at least two can be justified using our formalization. The first of these states that the environment should not use keys or other secrets in unaltered form. Specifically, the protocol under consideration should not encrypt messages with a key used to encrypt messages by any protocol in its environment. The reason this makes sense is that if two protocols use a particular form of encrypted message as a test to authenticate a peer, then the attacker might be able to make a principal running the first protocol accept a message which actually originated in a run of the second protocol. In our formalism, the environmental invariant for the protocol under consideration would fail to hold in such an environment, and the composition proof would therefore not go through. We note that this principle has been followed in the design of real-world protocols like IKE [13]. Also, Guttman and Fábrega have proved a theoretical result to the same effect in their strand space model [12]. The other rule of thumb (also recommended by Kelsey, Schneier and Wagner in [16]), is the use of unique protocol identifiers to prevent a message intended for use in one protocol to be mistaken for use in another protocol. This rule can also be similarly justified.

8. CONCLUSIONS

A modular approach towards construction and analysis of systems, which is often seen in other areas of computer science, does not seem to work very easily in computer security. The main problem is that systems which are individually secure might lose their security when they are put together because of the way they interact with each other. In this paper, we have presented a methodology for modular reasoning about security protocols. While doing so, we have addressed two basic problems: (a) how do you construct a protocol from smaller sub-protocols? (b) how do you prove that two protocols which are individually secure are also secure while running concurrently? In our formalism, we use before-after assertions to address the first problem and protocol invariants to address the second. The use of the methodology is illustrated by presenting modular proofs involving practical protocols, *ISO-9798-3* and *NSL* . This formalism also justifies some design principles which have been used by protocol designers in the construction of real-world protocols (e.g. IKE) and submuses some previous work in the formal methods community on the protocol composition problem. Future work would include a deeper investigation of the limits and applicability of this method and its connection with other approaches for reasoning about correctness of protocols. Also, it would be an interesting challenge to automate the proof system.

Acknowledgements. This work was partially supported by NSF CCR-0121403, Computational Logic Tools for Research and Education, the DoD University Research Initiative (URI) program administered by the Office of Naval Research (ONR) under Grant N00014-01-1-0795, and ONR Grants N00014-01-C-0454 and N00014-03-C-0237.

9. 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. Bellare and P. Rogaway. Entity authentication and key distribution. In *Advances in Cryprtology - Crypto '93 Proceedings*. Springer-Verlag, 1994.

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

[4] R. Canetti, C. Meadows, and P. Syverson. Environmental requirements for authentication protocols. In *Proceedings of Software Security - Theories and Systems, Mext-NSF-JSPS International*

Symposium, ISSS, LNCS 2609, pages 339–355. Springer-Verlag, 2003.

[5] A. Datta, A. Derek, J. C. Mitchell, and D. Pavlovic. A derivation system for security protocols and its logical formalization. In *Proceedings of 16th IEEE Computer Security Foundations Workshop*, pages 109–125. IEEE, 2003.

[6] W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654, 1976.

[7] W. Diffie, P. C. Van Oorschot, and M. J. Wiener. Authentication and authenticated key exchanges. *Designs, Codes and Cryptography*, 2:107–125, 1992.

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

[9] N. Durgin, J. C. Mitchell, and D. Pavlovic. A compositional logic for protocol correctness. In *Proceedings of 14th IEEE Computer Security Foundations Workshop*, pages 241–255. IEEE, 2001.

[10] 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.

[11] L. Gong and P. Syverson. Fail-stop protocols: An approach to designing secure protocols. *Dependable Computing for Critical Applications*, 5:79–100, 1998.

[12] J. Guttman and F. J. T. Fábrega. Protocol independence through disjoint encryption. In *Proceedings of 13th IEEE Computer Security Foundations Workshop*, pages 24–34. IEEE, 2000.

[13] D. Harkins and D. Carrel. The Internet Key Exchange (IKE), 1998. RFC 2409.

[14] N. Heintze and J. D. Tygar. A model for secure protocols and their composition. *IEEE Transactions on Software Engineering*, 22(1):16–30, January 1996.

[15] IEEE. Entity authentication mechanisms – part 3: Entity authentication using asymmetric techniques. Technical report ISO/IEC IS 9798-3, ISO/IEC, 1993.

[16] J. Kelsey, B. Schneier, and D. Wagner. Protocol interactions and the chosen protocol attack. In *Proceedings of the International Workshop on Security Protocols*, April 1997.

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

[18] N. Lynch. I/O automata models and proofs for shared-key communication systems. In *Proceedings of 12th IEEE Computer Security Foundations Workshop*, pages 14–29. IEEE, 1999.

[19] Z. Manna and A. Pnueli. *Temporal Verification of Reactive Systems: Safety*. Springer-Verlag, 1995.

[20] H. Mantel. On the Composition of Secure Systems. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 88–101, Oakland, CA, USA, May 12–15 2002. IEEE Computer Society.

[21] D. McCullough. Noninterference and the composability of security properties. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 177–186, Oakland, CA, USA, May 1988. IEEE Computer Society.

[22] D. McCullough. A hookup theorem for multilevel security. *IEEE Transactions on Software Engineering*, 16(6):563–568, 1990.

[23] J. McLean. Security models and information flow. In *Proceedings of the IEEE Symposium on Security and Privacy*, Oakland, CA, USA, May 1990. IEEE Computer Society.

[24] J. McLean. A general theory of composition for a class of "possibilistic" properties. *IEEE Transactions on Software Engineering*, 22(1):53–67, 1996.

[25] C. Meadows. A model of computation for the NRL protocol analyzer. In *Proceedings of 7th IEEE Computer Security Foundations Workshop*, pages 84–89. IEEE, 1994.

[26] C. Meadows. The NRL protocol analyzer: An overview. *Journal of Logic Programming*, 26(2):113–131, 1996.

[27] C. Meadows. Analysis of the Internet Key Exchange protocol using the NRL protocol analyzer. In *Proceedings of the IEEE Symposium on Security and Privacy*. IEEE, 1998.

[28] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1996.

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

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

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

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

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

[34] D. Peled. *Software Reliability Methods*. Springer-Verlag, 2001.

[35] P. Syverson and C. Meadows. A formal language for cryptographic protocol requirements. *Designs, Codes and Cryptography*, 7(1-2):27–59, 1996.

[36] F. J. Thayer, J. C. Herzog, and J. D. Guttman. Mixed strand spaces. In *Proceedings of 12th IEEE Computer Security Foundations Workshop*. IEEE, 1999.

[37] T. Y. C. Woo and S. C. Lam. A semantic model for authentication protocols. In *Proceedings IEEE Symposium on Research in Security and Privacy*, 1993.

APPENDIX

A. CORD CALCULUS

Cord calculus is the formalism we use to represent protocols. The main concepts are collected in this section.

A.1 Terms, Actions, Strands and Cords

The *terms* t are built starting from the variables x and the constants c. Moreover, the set of basic terms also contains the names N, which can be variables $\hat{X}$, $\hat{Y}$, $\hat{Z}$, or constants $\hat{A}$, $\hat{B}$, $\hat{C}$, and keys K which can be variables y and constants k. Upon these basic sets, the term language is then generated by some given constructors p, which always include tupling, the public key encryption $\{|t|\}_K$ of the term t by the key K, and the signature $\{|t|\}_{\overline{K}}$ over the term t with the private key $\bar{K}$. The language of *actions* is then built upon the terms by further constructors. They include sending a term $\langle t\rangle$, receiving into a variable (x), matching a term against a pattern $(t/q(x))$, and creating a new value (νx). A *strand* is a list of actions. The idea is that they should be the subsequent actions of a single role in a protocol. For example, the strand $[(\nu x)\langle x\rangle]$ represents a role in which a principal generates x and then sends out a message containing the freshly generated value. Since some actions of a role may be mutually independent, they can in principle be executed in any order. Different strands can thus be semantically equivalent. A *cord* is an equivalence class of behaviorally indistinguishable strands. We use the word *process* to refer to a principal executing an instance of a role. Table 5 summarizes the formal definition of cords. In addition to the sequence of actions, a cord has an *input interface* and an *output interface*. As the name suggests, the output interface represents the output of that cord. The input interface is used to provide initial data to a cord. These input parameters (called static parameters) can represent data known apriori (e.g. signing key) or data that becomes known by executing another cord via its output interface.

A.2 Cord Spaces and Runs

A *cord space* is a multiset of cords that may interact via communication. We use $\otimes$ for multiset union and [ ] for the empty multiset. The *runs* of a protocol arise as reaction sequences of cord spaces. The basic reactions within a cord space are shown in Table 6, with the required side conditions for each reaction shown below them. The substitution (t/x) is assumed to act on the strand left of it, *viz* S'. As usual, it is assumed that no free variable becomes bound after substitution, which can always be achieved by renaming the bound variables. Reaction (1) is a send and receive interaction, showing the simultaneous sending of term t by the first cord, with the receiving of t into variable x by the second cord. We call this an *external action* because it involves an interaction between two cords. The other reactions all take place within a single cord. We call these *internal actions*. Reaction (2) is a basic pattern match action, where the cord matches the pattern $p(t)$ with the expected pattern $p(x)$, and substitutes t for x. Reaction (3) is a decryption pattern match action, where the cord matches the pattern $\{|p(t)|\}_y$ with the decryption pattern $\{|p(x)|\}_{\overline{y}}$ and substitutes t for x. Reaction (4) is a signature verification pattern match action. Finally, reaction (5) shows the binding action where the cord creates a new value that doesn't appear elsewhere in the cordspace, and substitutes that value for x in the cord to the right. The intuitive motive for the condition $FV(t) = \emptyset$ should be clear: a term cannot be sent, or tested, until all of its free variables are instantiated.

A.3 Protocols

A protocol is defined by a finite set of roles, such as initiator, responder and server, each representing the actions of a participant in a protocol session. In representing protocol roles by cords, it is useful to identify the principal who carries out the role. Also, since the same principal might engage in multiple sessions in the same role (e.g., principal $\hat{A}$ might be the initiator in two sessions at the same time), associating a *role-id* with the cord allows us to distinguish between the actions carried out in the different sessions. A principal executing an instance of a role is referred to as a process.

The protocol intruder is capable of taking any of several possible actions, including receiving a message, decomposing it into parts, decrypting the parts if the key is known, remembering parts of messages, and generating and sending new messages. This is the standard "Dolev-Yao model", which appears to have developed from positions taken by Needham and Schroeder [32] and a model presented by Dolev and Yao [8]. A *run of a protocol* is a sequence of reaction steps from an *initial configuration*. An *initial configuration* is determined by a set of principals, a subset of which are designated as honest, a cord space constructed by assigning one or more roles to each honest principal, and an intruder cord that may use only the secret keys of dishonest principals. A particular initial configuration may give rise to many possible runs. Intuitively, a protocol has a property if in all runs of the protocol arising from all possible initial configurations, that property is preserved.

B. SEMANTICS OF PROTOCOL LOGIC

The formulas of the logic are interpreted over *runs*, which are finite sequences of reaction steps from an initial configuration. An equivalent view, consistent with the execution model used in defining Linear Temporal Logic (LTL) semantics, is to think of a run as a linear sequence of states. Transition from one state to the next is effected by an action carried out by some principal in some role. A formula is true in a run if it is true in the last state of that run.

The main semantic relation, $\mathcal{Q}, R \models \phi$, may be read, "formula ϕ holds for run R of protocol $\mathcal{Q}$." If $\mathcal{Q}$ is a protocol, then let $\bar{\mathcal{Q}}$ be the set of all initial configurations of protocol $\mathcal{Q}$, each including a possible intruder cord. Let $\mathsf{Runs}(\bar{\mathcal{Q}})$ be the set of all runs of protocol $\mathcal{Q}$ with intruder, each a sequence of reaction steps within a cord space. If ϕ has free variables, then $Q, R \models \phi$ if we have $Q, R \models \sigma\phi$ for all substitutions σ that eliminate all the free variables in ϕ. We write $\mathcal{Q} \models \phi$ if $\mathcal{Q}, R \models \phi$ for all $R \in \mathsf{Runs}(\bar{\mathcal{Q}})$.

In presenting the inductive definition of $Q, R \models \phi$ below, for ϕ without free variables, we use the following notation to describe a reaction step of cord calculus:

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

In words, $EVENT(R, X, P, \vec{n}, \vec{x})$ means that in run R, process X executes actions P, receiving data $\vec{n}$ into variables $\vec{x}$, where $\vec{n}$ and $\vec{x}$ are the same length. We use the notation $LAST(R, X, P, \vec{n}, \vec{x})$ to denote that the last event of run R is $EVENT(R, X, P, \vec{n}, \vec{x})$.

Action Formulas:

- $\mathcal{Q}, R \models \mathsf{Send}(A, m)$ if $LAST(R, A, \langle m\rangle, \emptyset, \emptyset)$.
- $\mathcal{Q}, R \models \mathsf{Receive}(A, m)$ if $LAST(R, A, (x), m, x)$.

(names)	N	$::=$	$\hat{X}$	variable name
			$\hat{A}$	constant name
(basic keys)	K_0	$::=$	k	constant key
			y	variable key
			N	name
(keys)	K	$::=$	K_0	basic key
			$\overline{K_0}$	inverse key
(role id)	η	$::=$	s	variable role-id
			$\bar{c}$	constant role-id
(process)	P	$::=$	N, η	
(terms)	t	$::=$	x	variable term
			c	constant term
			N	name
			K	key
			η	session id
			t, t	tuple of terms
			$\{\|t\|\}_K$	term encrypted with key K
			$\{\|t\|\}_{\overline{K}}$	term signed with key $\overline{K}$
(actions)	a	$::=$	ϵ	the null action
			$\langle t \rangle$	send a term t
			(x)	receive term into variable x
			(νx)	generate new term x
			$(t/q(x_1, \ldots, x_n))$	match term t to pattern $q(x_1, \ldots, x_n)$
(basic terms)	b	$::=$	$x \mid c \mid N \mid K$	basic terms allowed in patterns
(basic patterns)	p	$::=$	$b, \ldots, b$	tuple pattern
(patterns)	q	$::=$	p	basic pattern
			$\{\|p\|\}_{\overline{K}}$	decryption pattern
			$\{\|p\|\}_K$	signature verification pattern
(strands)	S	$::=$	$aS \mid a$	

Table 5: Syntax of terms, actions and strands

$$[S(x)S'] \otimes [T\langle t \rangle T'] \otimes C \quad \rhd\!\rhd \quad [SS'(t/x)] \otimes [TT'] \otimes C \tag{1}$$

$$[S\,(p(t)/p(x))\,S'] \otimes C \quad \rhd\!\rhd \quad [SS'(t/x)] \otimes C \tag{2}$$

$$[S\,(\{\|p(t)\|\}_y/\{\|p(x)\|\}_{\overline{y}})\,S'] \otimes C \quad \rhd\!\rhd \quad [SS'(t/x)] \otimes C \tag{3}$$

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

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

Where the following conditions must be satisfied:

(1) $FV(t) = \emptyset$
(2) $FV(t) = \emptyset$
(3) $FV(t) = \emptyset$ and y bound
(4) $FV(t) = \emptyset$
(5) $x \notin FV(S)$ and $m \notin FV(C) \cup FV(S) \cup FV(S')$

Table 6: Basic reaction steps

- $\mathcal{Q}, R \models \mathsf{New}(A, m)$ if $LAST(R, A, (\nu x), m, x)$.
- $\mathcal{Q}, R \models \mathsf{Decrypt}(A, \{\!|m|\!\}_K)$ if $\mathcal{Q}, R \models \mathsf{Has}(A, \{\!|m|\!\}_K)$
  $\wedge\, LAST(R, A, (\{\!|m|\!\}_K/\{\!|x|\!\}_{\overline{K}}), m, x)$
  Note: $\mathsf{Decrypt}(A, n)$ is *false* if $n \neq \{\!|m|\!\}_K$ for some m and K.
- $\mathcal{Q}, R \models \mathsf{Verify}(A, \{\!|m|\!\}_{\overline{K}})$ if $\mathcal{Q}, R \models \mathsf{Has}(A, \{\!|m|\!\}_{\overline{K}})$
  $\wedge\, \mathcal{Q}, R \models \mathsf{Has}(A, m)$
  $\wedge\, \mathcal{Q}, R \models \mathsf{Has}(A, K)$
  $\wedge\, LAST(R, A, (\{\!|m|\!\}_{\overline{K}}/\{\!|m|\!\}_K), \emptyset, \emptyset)$
  Note: $\mathsf{Verify}(A, n)$ is *false* if $n \neq \{\!|m|\!\}_{\overline{K}}$ for some m and K.

Other Formulas:

- $\mathcal{Q}, R \models \mathsf{Has}(A, m)$ if there exists an i such that $\mathsf{Has}_i(A, m)$ where Has_j is defined by induction on j as follows:
  $(\mathsf{Has}_0(A, m)$ if $((m \in FV(R|_A))$
  $\vee\, EVENT(R, A, (\nu x), m, x)$
  $\vee\, EVENT(R, A, (x), m, x)$
  and $\mathsf{Has}_{i+1}(A, m)$ if $(\mathsf{Has}_i(A, m')$
  $\wedge\, ((m' = \{\!|p(t)|\!\}_K \wedge m = t$
  $\wedge\, EVENT(R, A, (m'/\{\!|p(y)|\!\}_{\overline{K}}), t, y))$
  $\vee\, (m' = p(t) \wedge m = t$
  $\wedge\, EVENT(R, A, (m'/p(y)), t, y))))$
  $\vee(\mathsf{Has}_i(A, m') \wedge \mathsf{Has}_i(A, m'')$
  $\wedge\, ((m = m', m'') \vee (m = m'', m')))$
  $\vee(\mathsf{Has}_i(A, m') \wedge \mathsf{Has}_i(A, K)$
  $\wedge\, m = \{\!|m'|\!\}_K)$
  $\vee(\mathsf{Has}_i(A, a) \wedge \mathsf{Has}_i(A, g^b)$
  $\wedge\, m = g^{ab})$
  $\vee(\mathsf{Has}_i(A, g^{ab}) \wedge m = g^{ba})$
  Intuitively, Has_0 holds for terms that are known directly, either as a free variable of the role, or as the direct result of receiving or generating the term. Has_{i+1} holds for terms that are known by applying i operations (decomposing via pattern matching, composing via encryption or tupling, or by computing a Diffie-Hellman secret) to terms known directly.
- $\mathcal{Q}, R \models \mathsf{Fresh}(A, m)$ if $\mathcal{Q}, R \models (\diamondminus\, \mathsf{New}(A, m) \vee (\diamondminus\, \mathsf{New}(A, n) \wedge m = g(n))) \wedge \neg(\diamondminus\, \mathsf{Send}(A, m') \wedge m \subseteq m')$.
- $\mathcal{Q}, R \models \mathsf{Honest}(A)$ if $A \in HONEST(\mathbf{C})$ in initial configuration $\mathbf{C}$ for R.
- $\mathcal{Q}, R \models \mathsf{Contains}(t_1, t_2)$ if $t_2 \subseteq_v t_1$. t_2 is a visible subterm of t_1, $t_2 \subseteq_v t_1$, if $t_2 \subseteq t_1$ and it is not the case that all occurrences of t_2 in t_1 are as parameters of one-way functions. For example, $n \not\subseteq_v g(n)$. The only one-way function that we consider here is the Diffie-Hellman exponentiation function, $g(x)$.
- $\mathcal{Q}, R \models (\phi_1 \wedge \phi_2)$ if $\mathcal{Q}, R \models \phi_1$ and $\mathcal{Q}, R \models \phi_2$
- $\mathcal{Q}, R \models \neg\phi$ if $\mathcal{Q}, R \not\models \phi$
- $\mathcal{Q}, R \models \exists x.\phi$ if $\mathcal{Q}, R \models (d/x)\phi$, for some d, where $(d/x)\phi$ denotes the formula obtained by substituting d for x in ϕ.
- $\mathcal{Q}, R \models \diamondminus\, \phi$ if $\mathcal{Q}, R' \models \phi$, where R' is a (not necessarily proper) prefix of R. Intuitively, this formula means that in some state in the past, formula ϕ is true.
- $\mathcal{Q}, R \models \ominus\, \phi$ if $\mathcal{Q}, R' \models \phi$, where $R = R'e$, for some event e. Intuitively, this formula means that $\ominus\, \phi$ is true in a state if ϕ is true in the previous state.

Modal Formulas:

- $\mathcal{Q}, R \models \phi_1\, [P]_A\, \phi_2$ if $R = R_0 R_1 R_2$, for some R_0, R_1 and R_2, and either P does not match $R_1|_A$ or P matches $R_1|_A$ and $\mathcal{Q}, R_0 \models \sigma\phi_1$ implies $\mathcal{Q}, R_0 R_1 \models \sigma\phi_2$, where σ is the substitution matching P to $R_1|_A$.
- $\mathcal{Q}, R \models [P]_A\, \phi$ if $R = R_1 R_2$, for some R_1 and R_2, and either P does not match $R_1|_A$ or P matches $R_1|_A$ and $\mathcal{Q}, R_1 \models \sigma\phi$, where σ is the substitution matching P to $R_1|_A$.
  Note: The semantics of $\mathcal{Q}, R \models [P]_A\, \phi$ can be expressed in terms of the semantics of $\mathcal{Q}, R \models \phi_1\, [P]_A\, \phi_2$ by setting ϕ_1 to $true$ and requiring that R_0 be empty.

Semantic Entailment:

- $\Gamma \models \phi$ if $\mathcal{Q} \models \Gamma$ implies $\mathcal{Q} \models \phi$. Γ denotes a set of formulas. Intuitively, if in every run of $\mathcal{Q}$ all the formulas in Γ are true, then in every run of $\mathcal{Q}$, formula ϕ is also true.