Available online at www.sciencedirect.com
ScienceDirect

Electronic Notes in Theoretical Computer Science 172 (2007) 311–358

Electronic Notes in Theoretical Computer Science

www.elsevier.com/locate/entcs

Protocol Composition Logic (PCL)

Anupam Datta [1] Ante Derek [2] John C. Mitchell [3]
Arnab Roy [4]

Computer Science Department
Stanford University

Abstract

Protocol Composition Logic (PCL) is a logic for proving security properties of network protocols that use public and symmetric key cryptography. The logic is designed around a process calculus with actions for possible protocol steps including generating new random numbers, sending and receiving messages, and performing decryption and digital signature verification actions. The proof system consists of axioms about individual protocol actions and inference rules that yield assertions about protocols composed of multiple steps. Although assertions are written only using the steps of the protocol, the logic is sound in a strong sense: each provable assertion involving a sequence of actions holds in any protocol run containing the given actions and arbitrary additional actions by a malicious adversary. This approach lets us prove security properties of protocols under attack while reasoning only about the actions of honest parties in the protocol. PCL supports compositional reasoning about complex security protocols and has been applied to a number of industry standards including SSL/TLS, IEEE 802.11i and Kerberos V5.

Keywords: Security protocol analysis, logic, composition

1 Introduction

Network security protocols, such as key-exchange and key-management protocols, are difficult to design and debug. For example, the 802.11 Wired Equivalent Privacy (WEP) protocol, used to protect link-layer communications from eavesdropping and other attacks, has several serious security flaws [11]. Anomalies and shortcomings have also been discovered in standards and proposed standards for Secure Sockets Layer [76,60], the later 802.11i wireless authentication protocols [37,38], Kerberos [43,7,19,14], and others. Although many of these protocols may seem relatively simple, in comparison with more complex distributed systems, security

[1] Email: `danupam@cs.stanford.edu`
[2] Email: `aderek@cs.stanford.edu`
[3] Email: `mitchell@cs.stanford.edu`
[4] Email: `arnab@cs.stanford.edu`

1571-0661/$ – see front matter © 2007 Elsevier B.V. All rights reserved.
doi:10.1016/j.entcs.2007.02.012

protocols must achieve certain goals when an arbitrary number of multiple sessions are executed concurrently and an attacker may use information acquired in one session to compromise the security of another. Since security protocols form the cornerstones of modern secure networked systems, it is important to develop informative, accurate, and deployable methods for finding errors and proving that protocols meet their security requirements. While model checking has proven useful for finding certain classes of errors in network security protocols [59,60,66], logical methods and proof procedures are needed to show that protocols are correct, with respect to precise models of protocol execution and precise models of the capabilities of malicious attackers. In this paper, we describe a specific logic, developed for the purpose of proving security properties of network protocols, and give some examples of its use.

Protocol Composition Logic (PCL) [20,21,22,23,24,25,26,29,30,38,67,68] is a formal logic for stating and proving security properties of network protocols. The logic codifies and supports direct reasoning about the consequences of individual protocol steps, in a way that allows properties of individual steps to be combined to prove properties of complex protocols. The basic assertions are similar to Hoare logic [40] and dynamic logic [35], with the formula $\theta[P]_X\varphi$ stating that after actions P are executed in thread X, starting from a state where formula θ is true, formula φ is true about the resulting state. While the formula only mentions the actions P of thread X, states reached after X does P may arise as the result of these actions and any additional actions performed by other threads, including arbitrary actions by an attacker. PCL includes a number of action predicates, such as $\mathsf{Send}(X,t)$, $\mathsf{Receive}(X,t)$, $\mathsf{New}(X,t)$, $\mathsf{Decrypt}(X,t)$, $\mathsf{Verify}(X,t)$, which assert that the named thread has performed the indicated actions. For example, $\mathsf{Send}(X,t)$ holds in a run if thread X sent the term t as a message. One class of secrecy properties can be specified using the predicate $\mathsf{Has}(X,t)$, which intuitively means that t is built from constituents that X either generated (using a `new` action) or received in a way that did not hide them under encryption by a key not had by X. One predicate that is novel to PCL is $\mathsf{Honest}(\hat{X})$, which asserts that all actions of $\hat{X}$ are actions prescribed by the protocol. Honest is used primarily to assume that one party has followed the prescribed steps of the protocol. For example, if Alice initiates a transaction with Bob, and wishes to conclude that only Bob knows the data she sends, she may be able to provably do so by explicitly assuming that Bob is honest. If Bob is not honest, then Bob may make his private key known to the attacker, allowing the attacker to decrypt intercepted messages.

The PCL axioms and inference rules fall into several categories. One simple but necessary class of axioms assert that after an action is performed, the indicated thread has performed that action. Another class of axioms are those that state properties of cryptographic operations. For example, an axiom reflecting the unforgeability property of digital signatures states that whenever an agent verifies the signature of an honest agent, then that agent must have generated a signature on that message and sent it out in an earlier message. PCL also uses a novel form of induction, currently referred to as the "honesty rule", in which induction over basic sequences of actions performed by honest agents can be used to derive conclusions about arbitrary runs in the presence of adversary actions. To see how this works, in simple form, suppose that in some protocol, whenever a principal receives a message of the form $ENC_K\{|a, b|\}$, representing the encryption of a pair (a, b) under key K, the principal then responds with $ENC_K\{|b|\}$. Assume further that this is the only situation in which the protocol specifies that a message consisting of a single encrypted datum is sent. Using the honesty rule, it is possible to prove that if a principal A is honest, and A sends a message of the form $ENC_K\{|b|\}$, then A must have previously received a message of the form $ENC_K\{|a, b|\}$. For certain protocols, this form of reasoning allows us to prove that if one protocol participant completes the prescribed sequence of actions, and another principal named in one of the messages is honest, then the two participants are guaranteed a form of authentication.

Like the previous generation of protocol logics exemplified by BAN and GNY logics [13,32], PCL was also initially designed as a logic of authentication, involves annotating programs with assertions, does not require explicit reasoning about the actions of an attacker, and uses formulas for freshness, sending and receiving messages, and to express that two agents have a shared secret. In contrast to BAN and related logics, PCL avoids the need for an "abstraction" phase because PCL formulas contain the protocol programs, and PCL addresses temporal concepts directly, both through modal formulas that refer specifically to particular points in the execution of a protocol, and through temporal operators in pre- and post-conditions. PCL is also formulated using standard logical concepts (predicate logic and modal operators), does not involve "jurisdiction" or "belief", and has a direct connection with the execution semantics of network protocols that is used in explicit reasoning about actions of a protocol and an attacker, such as with Paulson's inductive method [64] and Schneider's rank function method [71].

A distinctive goal of PCL is to support compositional reasoning about security protocols, including parallel composition of different protocols, and sequential composition of protocol steps. For example, many protocols assume that long-term cryptographic keys have been properly distributed to protocol agents. PCL allows proofs of key-distribution protocols to be combined with proofs for protocols that use these keys. Another aspect of PCL is a composition method based on protocol templates, which are "abstract" protocols containing function variables for some of the operations used to construct messages. In the template method, correctness of a protocol template may be established under certain assumptions about these function variables. Then, a proof for an actual protocol is obtained by replacing the function variables with combinations of operations that satisfy the proof assumptions. PCL appears to scale well to industrial protocols of five to twenty messages (or more), in part because PCL proofs appear to be relatively short (for formal proofs) and it has been successfully applied to a number of industry standards including SSL/TLS, IEEE 802.11i and Kerberos V5. The PCL composition theorems are particularly useful in carrying out these larger-scale case studies.

This paper collects results from previous papers, develops the basic concepts in

$$\begin{aligned} &A \rightarrow B : m \\ &B \rightarrow A : n, SIG_B\{\!|n, m, A|\!\} \\ &A \rightarrow B : SIG_A\{\!|n, m, B|\!\} \end{aligned}$$

Fig. 1. Challenge-response protocol as arrows-and-messages

a uniform notation and semantic setting, improves on some of the previous technical definitions and proofs, and completes some details omitted from previous papers. The core of PCL was formulated earlier in [30,24]. Subsequent work on proof methods for PCL [20,21,22,23,68] as well as case studies using PCL [5,38,68] led to extensions and modifications to the syntax, semantics and proof system. In this paper, we unify the results in the earlier papers by presenting a definition of the logic using a uniform notation. Specifically, this paper subsumes [30,24] in scope and draws on the programming language syntax and the treatment of temporal operators from [5].

The rest of the paper is organized as follows. Section 2 describes the syntax and operational semantics for the protocol programming language. Section 3 presents the syntax and semantics for PCL, with the proof system and soundness theorem in Section 4. An application of the formal system to an authentication protocol is presented in Section 5. Section 6 presents theorems about sequential and parallel composition of protocols and illustrates their use through application to a key exchange protocol. Section 7 summarizes other results associated with PCL, which are not elaborated in this paper. Related work is discussed in Section 8 with conclusions in Section 9.

2 Modelling Protocols

In order to formally state and prove properties of security protocols we first need to represent protocols parts as mathematical objects and define how they execute. The common informal arrows-and-messages notation (used, for example, in [69,12]) is generally insufficient, since it only presents the executions of the protocol that occur when there is no attack. One important part of security analysis involves understanding the way honest principals running a protocol will respond to messages from a malicious attacker. In addition, our protocol logic requires more information about a protocol than the set of protocol executions obtained from honest and malicious parties; we need a high-level description of the program executed by each principal performing each protocol role so that we know not only which actions occur in a run (and which do not), but why they occur.

Now, we show how protocols are represented with an example. Figure 1 shows the standard three-way signature based challenge-response protocol (CR) in the informal arrows-and-messages notation. The goal of the protocol – mutual authentication of two parties, is achieved by exchanging two fresh nonces m and n, and the signature over both nonces and the identity of the other party.

The roles of the same protocol are written out in our notation in Figure 2, writing $\hat{X}$ and $\hat{Y}$ for the principals executing roles $\mathbf{Init_{CR}}$ and $\mathbf{Resp_{CR}}$, respectively. We differentiate between *principals* (denoted by $\hat{X}$, $\hat{Y}$, ...) which correspond to protocol participants and may be involved in more than one execution of the protocol at any point and *threads* (denoted by X, Y, ...) which refer to a principal executing one particular session of the protocol. In this example, the protocol consists of two roles, the initiator role and the responder role. The sequence of actions in the initiator role is given by the cord $\mathbf{Init_{CR}}$ in Figure 2. In words, the actions of a principal executing the role $\mathbf{Init_{CR}}$ are: generate a fresh random number; send a message with the random number to the peer $\hat{Y}$; receive a message with source address $\hat{Y}$; verify that the message contains $\hat{Y}$'s signature over the data in the expected format; and finally, send another message to $\hat{Y}$ with the initiator's signature over the nonce sent in the first message, the nonce received from $\hat{Y}$ and $\hat{Y}$'s identity. Formally, a *protocol* will be given by a finite set of roles, one for each role of the protocol. In addition to the sequence of actions, a cord has static input and output parameters used when sequentially composing roles.

```
$\mathbf{Init_{CR}} \equiv (\hat{Y})[$                    $\mathbf{Resp_{CR}} \equiv ()[$
        new $m$;                                  receive $\hat{X}, \hat{Y}, x$;
        send $\hat{X}, \hat{Y}, m$;                          new $n$;
        receive $\hat{Y}, \hat{X}, y, s$;                    $r :=$ sign $(n, x, \hat{X}), \hat{Y}$;
        verify $s, (y, m, \hat{X}), \hat{Y}$;                send $\hat{Y}, \hat{X}, n, r$;
        $r :=$ sign $(y, m, \hat{Y}), \hat{X}$;              receive $\hat{X}, \hat{Y}, t$;
        send $\hat{X}, \hat{Y}, r$;                          verify $t, (n, x, \hat{Y}), \hat{X}$;
  $]_X()$                                   $]_Y()$
```

Fig. 2. Roles of the Challenge-response protocol

2.1 Protocol Programming Language

Our protocol programming language is a conventional process calculus in the same vein as CCS, CSP, and their variants and descendants [41,57]. However, since the protocols we consider in this paper are a concurrent composition of sequential roles, the process calculus is tailored to this form. The formalism was originally described in [29,30] as *cord calculus*, a reference to the strand space formalism [31], which conveniently formalizes the practice of describing protocols by "arrows-and-messages", and displays the distributed traces of interacting processes. However, while strand spaces provide a global and static view of the information flow, we needed to analyze dynamics of distributed reasoning and computation. In order to formally capture the ways in which principals' actions (e.g. what they receive) may determine and change their later action (e.g. what they will send), we progressed from our initial attempt to use the strand model directly to a process calculus approach with operational semantics in the style of chemical abstract machine [9]. To represent the

(keys)	$K ::=$	k	basic key
		N	name
		$\overline{K}$	inverse key
(basic terms)	$u ::=$	x	basic term variable
		n	nonce
		N	name
		P	thread
		K	key
		u, u	tuple of basic terms
(terms)	$t ::=$	y	term variable
		u	basic term
		t, t	tuple of terms
		$ENC_K\{\|t\|\}$	term encrypted with key K
		$SIG_K\{\|t\|\}$	term signed with key $\overline{K}$

Table 1
Syntax of the Protocol Programming Language - terms

stores where the messages are to be received, we use process calculus variables, and a substitution mechanism expressed by simple reaction rules, corresponding to the basic communication and computation operations. In comparison with conventional process calculus, we needed a mechanism for identifying the principal executing a sequence of actions, so that access to cryptographic keys could be identified and restricted. The resulting process calculus provides a protocol execution model, based on accepted concepts from process calculi, strand spaces and the chemical abstract machine. Its formal components are as follows.

Terms

A basic algebra of *terms* t is assumed to be given. As usual, they are built from constants c and variables x, by a given set of constructors p, which in this case includes at least the tupling, the public key encryption $ENC_K\{|t|\}$, and the signature $SIG_K\{|t|\}$. We assume enough typing to distinguish the keys K from the principals $\hat{A}$, the nonces n and so on. Each type is given with enough variables.

As usual, the computation is modelled as term evaluation. The closed terms, that can be completely evaluated, are the contents of the messages exchanged in protocols. The terms containing free variables (i.e. pointers and references) cannot be sent. An example term is $\hat{X}, \hat{Y}, m$ sent in the first message of the CR protocol (see Figure 1), it is important to note that $\hat{X}, \hat{Y}$ are parts of the message specifying indented sender and the recipient rather than parameters to the send action.

For technical purposes, we make a distinction between *basic terms* u which do not contain cryptographic operations explicitly (although, they may contain

(actions)	$a ::=$	ϵ	the null action
		$\texttt{send}\ u$	send a term u
		$\texttt{receive}\ x$	receive term into variable x
		$\texttt{new}\ x$	generate new term x
		$\texttt{match}\ u/u$	match a term to a pattern
		$x := \texttt{sign}\ u, K$	sign the term u
		$\texttt{verify}\ u, u, K$	verify the signature
		$x := \texttt{enc}\ u, K$	encrypt the term u
		$x := \texttt{dec}\ u, K$	decrypt the term u
(strands)	$S ::=$	$[a; \ldots; a]_P$	
(roles)	$R ::=$	$(\vec{x})S(\vec{t})$	

Table 2
Syntax of the Protocol Programming Language - actions, strands and roles

variables whose value is, for example, an encryption) and *terms* t which may contain cryptographic primitives.

Names, keys, sessions and threads

We use $\hat{A}, \hat{B}, \ldots$ as *names* for protocol participants. We will overload the notation and also use $\hat{A}, \hat{B}, \ldots$ as designation for public-private key pairs of the corresponding agents. A particular participant might be involved in more than one session at a time. For example, agent $\hat{A}$ involved in the CR protocol might be acting as an initiator in two sessions with agents $\hat{B}$ and $\hat{C}$ and as a responder in another parallel session with $\hat{D}$. For this reason, we will give names to sessions and use A to designate a particular *thread* being executed by $\hat{A}$.

Actions, strands and roles

The set of actions contains nonce generation, encryption, decryption, signature generation and verification, pattern matching, testing and communication steps (sending and receiving). Pattern matching operator is used to construct and break tuples and perform equality checks. We will often omit the pattern matching operator and perform matching implicitly. For example, in the description of the CR protocol given in Figure 1 matching is implicitly done in the receive actions, if we were to completely write out actions there would be a `receive` x action followed by a `match` action analyzing the tuple, and performing the equality checks.

The list of actions will only contain basic terms which means that encryption cannot be performed implicitly; explicit `enc` action has to be used instead. For convenience, we assume that any variable will be assigned at most once, and the first occurrence of a particular variable has to be the assignment. Operational semantics of such single-assignment language will be significantly simpler as we can model the assignment with term substitution.

$$
\begin{array}{lrcl}
(1) & [\texttt{receive}\ x; S]_X \mid [\texttt{send}\ t; T]_Y & \longrightarrow & [S(t/x)]_X \mid [T]_Y \\
(2) & [\texttt{match}\ p(t)/p(x); S]_X & \longrightarrow & [S(t/x)]_X \\
(3) & [\texttt{new}\ x; S]_X & \longrightarrow & [S(m/x)]_X \\
(4) & [x := \texttt{enc}\ t, K; S]_X & \longrightarrow & [S(ENC_K\{\!|t|\!\}/x)]_X \\
(5) & [x := \texttt{dec}\ ENC_K\{\!|t|\!\}, K; S]_X & \longrightarrow & [S(t/x)]_X \\
(6) & [x := \texttt{sign}\ t, K; S]_X & \longrightarrow & [S(SIG_K\{\!|t|\!\}/x)]_X \\
(7) & [\texttt{verify}\ SIG_K\{\!|t|\!\}, t, K; S]_X & \longrightarrow & [S]_X
\end{array}
$$

Where the following conditions must be satisfied:
(1) $FV(t) = \emptyset$
(2) $m \notin FV(C) \cup FV(S)$, where C is the entire cord space

Table 3
Basic reaction steps

A *strand* is just a sequence of actions together with the designation of a thread performing the actions. A *role* is a strand with input and output interfaces used when performing sequential composition. All variables inside a role must be bound either by the input interface or by other actions. A *cord* is just a strand with no free variables, i.e. all ground terms are either constants or bound by a particular action.

2.2 Execution Model

Cord Spaces

A *cord space* is a multiset of cords, each annotated in the subscript by the name of the agent executing it. The multiset union is denoted by |, the empty multiset by []. The idea is that a cord space represents a group of processes ready to engage in communication and distributed computation. Their operational behavior is defined by the reaction rules in Table 3.

The required side conditions for each reaction are shown below them. The substitution (t/x) acts on the strand to the left. As usual, it is assumed that no free variable becomes bound after substitution, which is 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) 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 have been instantiated, so that it can be evaluated. Also, when the new nonce is generated via the `new` action, it is required that the resulting constant is unique in the entire cord space.

Reactions (4) and (5) are the encryption and decryption actions respectively. For example, the decryption action matches the pattern $ENC_K\{|p(t)|\}$ and substitutes t for x. Reactions (6) and (7) are the signature generation and signature verification actions respectively. As we already mentioned, since the assignment is modelled via term substitution, a single variable can be assigned only once.

Protocols

A *protocol* $\mathcal{Q}$ is a set of roles $\{\rho_1, \rho_2, \ldots, \rho_k\}$, each executed by zero or more honest principals in any run of $\mathcal{Q}$. Intuitively, these roles may correspond to the initiator, responder and the server, each specified by a sequence of actions to be executed in a single instance of a role. A protocol participant is called a *principal* and denoted by $\hat{A}, \hat{B}, \cdots$ etc. A single instance of a particular role executed by a principal will be called a *thread*. All threads of a single principal share static data such as long-term keys. This is formalized using static binding, described above. As a notational convenience, we will use X to denote a thread of a principal $\hat{X}$.

A *private key* is a key of form $\overline{X}$, which represents the decryption key in a public key cryptosystem. Private key $\overline{X}$ is only allowed to occur in the threads of principal $\hat{X}$. Moreover, it is only allowed to occur in the decryption pattern (corresponding to a participant decrypting a message encrypted by its public key) and in the signature construction (corresponding to a participant signing a message). These restrictions prevent private keys from being sent in a message. While some useful protocols might send private keys, we prevent roles from sending their private keys (in this paper) since this allows us to take secrecy of private keys as an axiom, shortening proofs of protocol properties.

Intruder roles

An *attack* is usually a process obtained by composing a protocol with another process, in such a way that the resulting runs, projected to the protocol roles, do not satisfy the protocol requirements. An *attacker*, or *intruder*, is a set of threads sharing all data in an attack, and playing roles in one or more protocol sessions. This intuition is captured in the definition of initial configurations below. The actions available for building the intruder roles usually include receiving and sending messages, decomposing them into parts, decrypting them by known keys, storing data, and even generating new data. This is the standard "Dolev-Yao model", which appears to have developed from positions taken by Needham and Schroeder [61] and a model presented by Dolev and Yao [28].

Buffer cord

Cords reactions, as we defined them, model synchronous communication – a message send action cannot happen in one cord unless a message receive action happens simultaneously. Since real communication networks are asynchronous, we need to introduce a buffer where sent messages can be stored until someone is ready to receive them. In order to model this with cords we introduce a *buffer cord* [`receive` x; `send` x], it models a message being received and than eventually send. We will require that all send and receive actions by principals and the intruder are performed via buffer cords and assume that in every protocol there are enough instances of the buffer cord to guarantee delivery of every message. Buffer cords are a part of the infrastructure rather than a part of the protocol, we assume that they are executed by special nameless agents. Unless otherwise specified, when we refer to a thread, we mean a non-buffer thread, similarly, when we refer to an action, we mean an action performed by a non-buffer thread. As demonstrated by [6], this synchronous process calculus extended with buffers faithfully represents asynchronous communication and corresponds to the usual definition of asynchronous process calculus.

Configurations and runs

Initial configuration of a protocol $\mathcal{Q}$ is determined by: (1) A set of principals, some of which are designated as honest. (2) A cordspace constructed by assigning roles of $\mathcal{Q}$ to threads of honest principals. (3) One or more intruder cords, which may use keys of dishonest principals. (4) A finite number of buffer cords, enough to accommodate every send action by honest threads and the intruder threads. A *run* R is a sequence of reaction steps from the initial configuration, subject to constraint that every send/receive reaction step happens between some buffer cord and some (non-buffer) thread. A particular initial configuration may give rise to many possible runs.

Events and traces

Since the protocol logic reasons about protocol runs, we need to introduce some additional notation for them. An *event* is a ground substitution instance of an action, i.e., an action in which all variables have been replaced by terms containing only constants. An event represents the result of a reaction step, viewed from the perspective of a single cord that participated in it. For example, if the thread A sends message m (into a receiving buffer cord), then the event `send` m is a send event of A. Alternatively, we can look at a run as a linear sequence of events starting from an initial configuration.

We use the following meta-notation to describe a reaction step of cord calculus:

$$EVENT(R, X, P, \vec{n}, \vec{x}) \equiv \big(([PS]_X \mid C \quad\longrightarrow\quad [S(\vec{n}/\vec{x})]_X \mid C'\big) \in R\big)$$

When $EVENT(R, X, P, \vec{n}, \vec{x})$ holds we will say that in *in run R, thread X executed action P, receiving data $\vec{n}$ into variables $\vec{x}$*, where $\vec{n}$ and $\vec{x}$ are the same length.

A *trace* is a list of events by some thread in a run. We use $R|_X$ to denote the events that occurred for thread X in run R. For a sequence of actions P, protocol $\mathcal{Q}$, run R and thread X, we say "*P matches $R|_X$*" if $R|_X$ is precisely σP, where σ is a substitution of values for variables. If P matches $R|_X$ using substitution σ, then σ is called the *matching substitution*.

2.2.1 Protocol properties

In this section we collect some properties of the protocols that will be useful in the rest of the paper.

Lemma 2.1 (No Telepathy) *Let $\mathcal{Q}$ be a protocol, R be an arbitrary run, and X be a thread. Let m be any message sent by X as part of cord ρ_i. Then every symbol in the term m is either generated in ρ_i, received in ρ_i, or was in the static interface of ρ_i.*

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

Lemma 2.2 (Asynchronous communication) *In every run, any thread that wished to send a message can always send it. Also, there is a strict linear order between all external actions.*

Proof. By definition, there are enough buffer cords in the initial configuration to provide a receive for every send action by a non-buffer thread. Since "external action" refers to a send or a receive by a non-buffer thread, it follows from the definition of a run that no two external actions can happen in the same step of the run. □

Lemma 2.3 *For every receive action there is a corresponding send action. More formally, if in run R, thread X executed action* `receive` *x, receiving data m into variable x then there exists a thread Y such that in the same run R thread Y executed the* `send` *m action.*

Proof. This follows from the definition of the basic cord calculus reaction steps and the definition of the buffer cords. □

Lemma 2.4 *For any initial configuration $\mathbf{C}$ of protocol $\mathcal{Q}$, and any run R, if principal $\hat{X} \in HONEST(\mathbf{C})$, then for any thread X performed by principal $\hat{X}$, $R|_X$ is a trace of a single role of $\mathcal{Q}$ executed by $\hat{X}$.*

Proof. This follows from the definition of initial configuration, which is constructed by assigning roles to threads of honest principals. □

3 Protocol Logic

3.1 Syntax

The formulas of PCL are given by the grammar in Table 4, where S may be any strand. Here, t and P denote a term and a *thread*, respectively. We use ϕ and ψ to indicate predicate formulas, and m to indicate a generic term we call a "message". A message has the form (source, destination, protocol-identifier, content), giving each message source and destination fields and a unique protocol identifier in addition to the message contents. The source field of a message may not identify the actual sender of the message since the intruder can spoof the source address. Similarly, the

Action formulas

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

Formulas

$$\phi ::= \mathsf{a} \,|\, \mathsf{a} < \mathsf{a} \,|\, \mathsf{Has}(P,t) \,|\, \mathsf{Fresh}(P,t) \,|\, \mathsf{Gen}(P,t) \,|\, \mathsf{FirstSend}(P,t,t) \,|\, \mathsf{Honest}(N) \,|\, t = t \,|\, \mathsf{Contains}(t,t) \,|\, \phi \wedge \phi \,|\, \neg\phi \,|\, \exists x.\phi \,|\, \mathsf{Start}(P)$$

Modal formulas

$$\Psi ::= \phi \; S \; \phi$$

Table 4
Syntax of the logic

principal identified by the destination field may not receive the message since the intruder can intercept messages. Nonetheless, the source and destination fields in the message may be useful for stating and proving authentication properties while the protocol-identifier is useful for proving properties of protocols.

Most protocol proofs use formulas of the form $\theta[P]_X\phi$, which means that after actions P are executed in thread 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 precise semantics discussed in the next section.

Action formulas

Action formulas are used to state that particular actions have been performed by various threads. Formula $\mathsf{Send}(X, m)$ means that principal $\hat{X}$ has send a message m in the thread X. Predicates $\mathsf{Receive}, \mathsf{Encrypt}, \mathsf{Sign}, \cdots$ etc. are similarly used to state that the corresponding actions have been performed. Action predicates are crucial in modelling authentication properties of the protocol. In PCL, a fact that $\hat{A}$ has authenticated $\hat{B}$ will be described by saying that $\hat{B}$ must have performed certain actions prescribed by the protocols.

Knowledge

Formula $\mathsf{Has}(X, x)$ means that principal $\hat{X}$ possesses information x in the thread X. This is "possess" 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. Formula $\mathsf{Fresh}(X, t)$ 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. Formula $\mathsf{Gen}(X, t)$ means that the term t originated in the thread X in the sense that it was "fresh" in X at some point.

Formula $\mathsf{Contains}(t_1, t_2)$ means that the term t_1 contains term t_2 as a subterm. Predicate Has can be used to model secrecy properties, for example, a fact that a term t is a shared secret between threads X and Y is captured by the logical formula $\forall Z.\mathsf{Has}(Z, t) \supset (Z = X \vee Z = Y)$.

Temporal ordering

Formula $\mathsf{Start}(X)$ means that the thread X did not execute any actions in the past. Formula, $\mathsf{a}_1 < \mathsf{a}_2$ means that both actions a_1 and a_2 happened in the run and moreover, that the action a_2 happened after the action a_1. Note that actions may not be unique. For example, a thread X might have received the same term multiple times, temporal ordering operator only states that *some* two actions a_1 and a_2 have happened in that order. Formula $\mathsf{FirstSend}(P, t, t')$ means that the thread P has send a term t (possibly as a part of some bigger message) and that the first such occurrence was an action when P send the message t'. Temporal ordering relation can be used to strengthen the authentication properties by imposing ordering between actions of different participants.

Honesty

Formula $\mathsf{Honest}(\hat{X})$ means 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, each thread X of principal $\hat{X}$ assumes a particular role of the protocol and does exactly the actions prescribed by that role.

Modal Formulas

Modal formulas attach assertions – *preconditions* and *postconditions* – to programs. Informally, formula of the form $\theta[P]_X\phi$ means that after actions P are executed in thread X, starting from a state where formula θ is true, formula ϕ is true about the resulting state of X.

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

The inductive definition of $\mathcal{Q}, R \models \phi$ is given below. Because a run is a sequence of reaction steps, each step resulting from a principal executing an action, is 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 thread in some role.

Action Formulas

Action formulas hold as a result of a thread executing a particular action in the run. Semantics of corresponding predicates is defined in a straightforward fashion; a particular action predicate holds in the run if the corresponding action happened in the run with the same terms as parameters.

$\mathcal{Q}, R \models \mathsf{Send}(A, m)$ if in the run R, thread A executed action `send` m.

$\mathcal{Q}, R \models \mathsf{Receive}(A, m)$ if there exists a variable x such that in the run R, thread A executed action `receive` x, receiving data m into variable x.

$\mathcal{Q}, R \models \mathsf{New}(A, m)$ if there exists a variable x such that in the run R, thread A executed action `new` x, receiving data m into variable x.

$\mathcal{Q}, R \models \mathsf{Encrypt}(A, ENC_K\{|m|\})$ if there exists a variable x such that in the run R, thread A executed action $x :=$ `enc` m, K, receiving data $ENC_K\{|m|\}$ into variable x.

$\mathcal{Q}, R \models \mathsf{Decrypt}(A, ENC_K\{|m|\})$ if there exists a variable x such that in the run R, thread A executed action $x :=$ `dec` $ENC_K\{|m|\}, K$, receiving data m into variable x.

$\mathcal{Q}, R \models \mathsf{Sign}(A, SIG_K\{|m|\})$ if there exists a variable x such that in the run R, thread A executed action $x :=$ `sign` m, K, receiving data $SIG_K\{|m|\}$ into variable x.

$\mathcal{Q}, R \models \mathsf{Verify}(A, SIG_K\{|m|\})$ if in the run R, thread A executed the signature verification action `verify` $SIG_K\{|m|\}, K$.

Predicate Has

We model knowledge using the predicate Has. Intuition behind the semantics definition for this predicate is simple, Has should hold for terms that are known directly, either as a free variable of the rule or as a result of receiving or generating a term. Furthermore, Has should hold for all terms that can be obtained from terms known directly via one or more "Dolev-Yao" operations (decomposing via pattern matching, or decryption with a known key or composing via encryption or tupling).

$\mathcal{Q}, R \models \mathsf{Has}(A, m)$ if there exists an i such that $\mathsf{Has}_i(A, m)$ where Has_i is inductively as follows:

$$\begin{array}{ll}
\mathsf{Has}_0(A, m) & \texttt{if } m \in FV(R|_A) \\
\mathsf{Has}_0(A, m) & \texttt{if } EVENT(R, A, (\texttt{new } x), m, x) \\
\mathsf{Has}_0(A, m) & \texttt{if } EVENT(R, A, (\texttt{receive } x), m, x)
\end{array}$$

$$\begin{array}{rl}
\mathsf{Has}_{i+1}(A,m) & \texttt{if } \mathsf{Has}_i(A,m) \\
\mathsf{Has}_{i+1}(A,(m,m')) & \texttt{if } \mathsf{Has}_i(A,m) \texttt{ and } \mathsf{Has}_i(A,m') \\
\mathsf{Has}_{i+1}(A,m) & \texttt{if } \mathsf{Has}_i(A,(m,m')) \texttt{ or } \mathsf{Has}_i(A,(m',m)) \\
\mathsf{Has}_{i+1}(A,ENC_K\{|m|\}) & \texttt{if } \mathsf{Has}_i(A,m) \texttt{ and } \mathsf{Has}_i(A,K) \\
\mathsf{Has}_{i+1}(A,m) & \texttt{if } \mathsf{Has}_i(A,ENC_K\{|m|\}) \texttt{ and } \mathsf{Has}_i(A,K) \\
\mathsf{Has}_{i+1}(A,m) & \texttt{if } \mathsf{Has}_i(A,m') \texttt{ and } m' = p(m) \\
 & \quad \texttt{and } EVENT(R,A,(\texttt{match } m'/p(y)),m,y)
\end{array}$$

Other Formulas

$\mathcal{Q},R \models \mathsf{Start}(A)$ if $R|_A$ is empty. Intuitively this formula means that A didn't execute any actions in the past.

$\mathcal{Q},R \models \mathsf{Fresh}(A,t)$ if $\mathcal{Q},R \models \mathsf{New}(A,t)$ holds and furthermore for all m such that $\mathcal{Q},R \models \mathsf{Send}(A,m)$ it holds that t is not a subterm of m.

$\mathcal{Q},R \models \mathsf{Gen}(A,t)$ if there is a prefix R' of R such that $\mathcal{Q},R' \models \mathsf{Fresh}(A,t)$ holds.

$\mathcal{Q},R \models \mathsf{FirstSend}(A,t,t')$ if t is a subterm of t', $\mathcal{Q},R \models \mathsf{Send}(A,t')$ holds and for all prefixes R' of R and all terms t'' such that $t \subseteq t''$ and $\mathcal{Q},R' \models \mathsf{Send}(A,t'')$ it has to be that $\mathcal{Q},R' \models \mathsf{Send}(A,t')$.

$\mathcal{Q},R \models \mathsf{Honest}(\hat{A})$ if $\hat{A} \in HONEST(\mathbf{C})$ in initial configuration $\mathbf{C}$ for R and all threads of $\hat{A}$ are in a "pausing" state in R. More precisely, $R|_{\hat{A}}$ is an interleaving of basic sequences of roles in Q.

$\mathcal{Q},R \models \mathsf{Contains}(t_1,t_2)$ if $t_2 \subseteq t_1$, where $\subseteq$ is the subterm relation between terms.

$\mathcal{Q},R \models (\phi_1 \wedge \phi_2)$ 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.

Modal Formulas

$\mathcal{Q},R \models \phi_1[P]_A\phi_2$ if $R = R_0R_1R_2$, 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 $\mathcal{Q},R_0R_1 \models \sigma\phi_2$, where σ is the substitution matching P to $R_1|_A$.

4 Proof System

The proof system combines a complete axiom system for first-order logic (not listed since any axiomatization will do), together with axioms and proof rules for protocol actions, temporal reasoning, and a specialized form of invariance rule.

4.1 Axioms for Protocol Actions

Axioms for protocol actions state properties that hold in the state as a result of executing certain actions (or not executing certain actions). We use a in the axioms to denote any one of the actions and a to denote the corresponding predicate in the logic. $\top$ denotes the boolean value *true*. Axiom **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 while **AA2** states that at the start of a thread any action predicate applied to the thread is false. Axiom **AA3** states that the predicate asserting thread X has not sent the term t remains false after any action that does not send a term that unifies with t, if it is false before the action. **AA4** states that after thread X does actions $a, \cdots, b$ in sequence, the action predicates, a and b, corresponding to the actions, are temporally ordered in the same sequence.

AA1 $\top[a]_X\ \mathsf{a}$

AA2 $\mathsf{Start}(X)[\,]_X\ \neg\mathsf{a}(\mathsf{X})$

AA3 $\neg\mathsf{Send}(X,t)[b]_X\neg\mathsf{Send}(X,t)$

if $\sigma\mathsf{Send}(X,t) \neq \sigma\mathsf{b}$ for all substitutions σ

AA4 $\top[a;\cdots;b]_X\mathsf{a} < \mathsf{b}$

The following axioms deal with properties of freshly generated nonces. Axiom **AN1** states that a particular nonce is generated by a unique thread. If thread X generates a new value n and does no further actions, then axiom **AN2** says that no one else knows n, and axiom **AN3** says that n is fresh, and axiom **AN4** says that X is the originating thread of nonce n.

AN1 $\mathsf{New}(X,x) \wedge \mathsf{New}(Y,x) \supset X = Y$

AN2 $\top[\mathtt{new}\ x]_X\ \mathsf{Has}(Y,x) \supset (Y = X)$

AN3 $\top[\mathtt{new}\ x]_X\ \mathsf{Fresh}(X,x)$

AN4 $\mathsf{Fresh}(X,x) \supset \mathsf{Gen}(X,x)$

4.2 Possession Axioms

The possession axioms characterize the terms that a principal can derive if it possesses certain other terms. **ORIG** and **REC** state respectively that a principal possesses a term if she freshly generated it (a nonce) or if she received it in some message. **TUP** and **ENC** enable construction of tuples and encrypted terms if the parts are known. **PROJ** and **DEC** allow decomposition of a tuple into its components and decryption of an encrypted term if the key is known.

ORIG $\mathsf{New}(X,x) \supset \mathsf{Has}(X,x)$

REC $\mathsf{Receive}(X,x) \supset \mathsf{Has}(X,x)$

TUP $\mathsf{Has}(X,x) \wedge \mathsf{Has}(X,y) \supset \mathsf{Has}(X,(x,y))$

ENC $\mathsf{Has}(X,x) \wedge \mathsf{Has}(X,K) \supset \mathsf{Has}(X,ENC_K\{|x|\})$

PROJ $\mathsf{Has}(X,(x,y)) \supset \mathsf{Has}(X,x) \wedge \mathsf{Has}(X,y)$

DEC $\mathsf{Has}(X,ENC_K\{|x|\}) \wedge \mathsf{Has}(X,K) \supset \mathsf{Has}(X,x)$

Axioms **AR1**, **AR2** and **AR3** are used to model obtaining information about structure of terms as they are being parsed. They allow us to plug in appropriate substitutions obtained by matching, signature verification and decryption actions to terms inside the action predicate a.

AR1 $\mathsf{a}(x)[\texttt{match}\ q(x)/q(t)]_X\ \mathsf{a}(t)$

AR2 $\mathsf{a}(x)[\texttt{verify}\ x, t, K]_X\ \mathsf{a}(SIG_K\{|t|\})$

AR3 $\mathsf{a}(x)[y := \texttt{dec}\ x, K]_X\ \mathsf{a}(ENC_K\{|y|\})$

4.3 Encryption and Signature

The next two axioms are aimed at capturing the black-box model of encryption and signature. Axiom **VER** refers to the unforgeability of signatures while axiom **SEC** stipulates the need to possess the private key in order to decrypt a message encrypted with the corresponding public key.

SEC $\mathsf{Honest}(\hat{X}) \wedge \mathsf{Decrypt}(Y, ENC_{\hat{X}}\{|x|\}) \supset (\hat{Y} = \hat{X})$

VER $\mathsf{Honest}(\hat{X}) \wedge \mathsf{Verify}(Y, SIG_{\hat{X}}\{|x|\}) \wedge \hat{X} \neq \hat{Y} \supset$
$\exists X.\mathsf{Send}(X, m) \wedge \mathsf{Contains}(m, SIG_{\hat{X}}\{|x|\})$

4.4 Generic Rules

These are generic Floyd-Hoare style rules for reasoning about program pre-conditions and post-conditions. For example, the generalization rule **G4** says that if ϕ is a valid formula (it holds in all runs of all protocols) then it can be used in a postcondition of any modal form.

$$\frac{\theta[P]_X\phi \qquad \theta[P]_X\psi}{\theta[P]_X\phi \wedge \psi}\ \mathbf{G1} \qquad\qquad \frac{\theta[P]_X\psi \qquad \phi[P]_X\psi}{\theta \vee \phi[P]_X\psi}\ \mathbf{G2}$$

$$\frac{\theta' \supset \theta \qquad \theta[P]_X\phi \qquad \phi \supset \phi'}{\theta'[P]_X\phi'}\ \mathbf{G3} \qquad\qquad \frac{\phi}{\theta[P]_X\phi}\ \mathbf{G4}$$

4.5 Sequencing Rule

Sequencing rule gives us a way of sequentially composing two cords P and P' when post-condition of P, matches the pre-condition of P'.

$$\frac{\phi_1[P]_X\phi_2 \qquad \phi_2[P']_X\phi_3}{\phi_1[PP']_X\phi_3}\ \mathbf{S1}$$

4.6 Preservation Axioms

The following axioms state that the truth of certain predicates continue to hold after further actions. **P1** states this for the predicates Has, FirstSend, a whereas **P2** states that freshness of a term holds across actions that do not send out some term containing it.

P1 $\mathsf{Persist}(X, t)[a]_X \mathsf{Persist}(X, t)$
for $\mathsf{Persist} \in \{\mathsf{Has}, \mathsf{FirstSend}, \mathsf{a}, \mathsf{Gen}\}$.

P2 $\mathsf{Fresh}(X, t)[a]_X \mathsf{Fresh}(X, t)$
where $t \not\subseteq a$.

4.7 Axioms and Rules for Temporal Ordering

The next two axioms give us a way of deducing temporal ordering between actions of different threads. Informally, $\mathsf{FirstSend}(X, t, t')$ says that a thread X generated a fresh term t and sent it out first in message t'. This refers to the first such send event and is formally captured by axiom **FS1**. Axiom **FS2** lets us reason that if a thread Y does some action with a term t'', which contains a term t, first sent inside a term t' by a thread X as a subterm, then that send must have occurred before Y's action.

FS1 $\mathsf{Fresh}(X, t)[\mathsf{send}\ t']_X \mathsf{FirstSend}(X, t, t')$
where $t \subseteq t'$.

FS2 $\mathsf{FirstSend}(X, t, t') \wedge \mathsf{a}(Y, t'') \supset \mathsf{Send}(X, t') < \mathsf{a}(Y, t'')$
where $X \neq Y$ and $t \subseteq t''$.

4.8 The Honesty Rule

The honesty rule is an invariance rule for proving properties about the actions of principals that execute roles of a protocol, similar in spirit to the basic invariance rule of LTL [47] and invariance rules in other logics of programs. The honesty rule is often used to combine facts about one role with inferred actions of other roles. For example, suppose Alice receives a signed response from a message sent to Bob. Alice may use facts about Bob's role to infer that Bob must have performed certain actions before sending his reply. This form of reasoning may be sound if Bob is honest, since honest, by definition in our framework, means "follows one or more roles of the protocol." The assumption that Bob is honest is essential because the intruder may perform arbitrary actions with any key that has been compromised. Since we have added preconditions to the protocol logic presented in [29,30], we reformulate the rule here is a more convenient form using preconditions and postconditions.

To a first approximation, the honesty rule says that if a property holds before each role starts, and the property is preserved by any sequence of actions that an honest principal may perform, then the property holds for every honest principal. An example property that can be proved by this method is that if a principal sends a signed message of a certain form, the principal must have received a request for this response. The proof of a property like this depends on the protocol, of course. For this reason, the antecedent of the honesty rule includes a set of formulas constructed from the set of roles of the protocol in a systematic way. A subtle issue is that the honesty rule only involves certain points in a protocol execution. This is not a fundamental limitation in the nature of invariants, but the result of a design tradeoff that was made in formulating the rule. More specifically, it is natural to assume that once a thread receives a message, the thread may continue to send messages and perform internal actions until the thread needs to pause to wait for additional input. Another way to regard this assumption is that we do not give the attacker control over the scheduling of internal actions or the point at which messages are sent. The attacker only has control over the network, not local computing. We therefore formulate our honesty rule to prove properties that hold in every pausing state of every honest rule. By considering fewer states, we consider more invariants true. By analogy with database transactions, for example, we consider a property an invariant if it holds after every "transaction" is completed, allowing roles to temporarily violate invariants as long as they preserve them before pausing. A similar convention is normally associated with loop invariants: a property is a loop invariant if it holds every time the top of the loop is reached; it is not necessary that the invariant hold at every point in the body of the loop.

Recall that a protocol $\mathcal{Q}$ is a set of roles $\{\rho_1, \rho_2, \ldots, \rho_k\}$, each executed by zero or more honest principals in any run of $\mathcal{Q}$. A sequence P of actions is a *basic sequence* of role ρ, written $P \in BS(\rho)$, if P is a contiguous subsequence of ρ such that either (i) P starts at the beginning of ρ and ends with the last action before the first receive, or (ii) P starts with a receive action and continues up to the last action before the next receive, or (iii) P starts with the last receive action of the role and continues through the end of the role. In the syntactic presentation below, we use the notation $\forall\rho \in \mathcal{Q}.\, \forall P \in BS(\rho).\ \phi[P]_X\phi$ to denote a finite set of formulas of the form $\phi[P]_X\phi$ - one for each basic sequence P in the protocol. The quantifiers $\forall\rho \in \mathcal{Q}$ and $\forall P \in BS(\rho)$ are not part of the syntax of PCL, but are meta-notation used to state this rule schema.

$$\frac{\mathsf{Start}(X)[\,]_X\ \phi \qquad \forall\rho \in \mathcal{Q}.\forall P \in BS(\rho).\ \phi\ [P]_X\ \phi}{\mathsf{Honest}(\hat{X}) \supset \phi}\ \mathbf{HON_Q} \quad \text{no free variable in } \phi \text{ except } X \text{ bound in } [P]_X$$

4.9 Soundness

The soundness theorem for this proof system is proved, by induction on the length of proofs, in Appendix B. Here we state the soundness theorem and demonstrate proofs for a few relevant proof rules and axioms. We write $\Gamma \vdash \gamma$ if γ is provable from the formulas in Γ and any axiom or inference rule of the proof system except the honesty rule ($\mathbf{HON_Q}$ for any protocol $\mathcal{Q}$). We write $\Gamma \vdash_Q \gamma$ if γ is provable from the formulas in Γ, the basic axioms and inference rules of the proof system and the honesty rule for protocol $\mathcal{Q}$ (i.e., $\mathbf{HON_Q}$ but not $\mathbf{HON_{Q'}}$ for any $\mathcal{Q}' \neq \mathcal{Q}$). Here γ is either a modal formula or a basic formula (i.e., of the syntactic form Ψ or ϕ in Table 4).

Theorem 4.1 *If $\Gamma \vdash_Q \gamma$, then $\Gamma \models_Q \gamma$. Furthermore, if $\Gamma \vdash \gamma$, then $\Gamma \models \gamma$.*

Axiom VER

$$\mathsf{Honest}(\hat{X}) \wedge \mathsf{Verify}(Y, SIG_{\hat{X}}\{|x|\}) \wedge \hat{X} \neq \hat{Y} \supset$$

$\exists X.\mathsf{Send}(X, m) \wedge \mathsf{Contains}(m, SIG_{\hat{X}}\{|x|\})$

Informally, **VER** says that if an agent $\hat{X}$ is honest, and some thread Y executed by principal $\hat{Y}$ has verified a signature $SIG_X\{|x|\}$ (i.e. a message signed with $\hat{X}$'s private key), then $\hat{X}$ must have sent the signature out in some thread X, as a part of some message. In other words, when $\hat{X}$ is honest, he is the only one who can sign messages with his public key. Therefore, every message signed by $\hat{X}$ must have originated from some thread X performed by principal $\hat{X}$.

Let $\mathcal{Q}$ be a protocol, and $\mathbf{C}$ be an initial configuration of $\mathcal{Q}$ such that $\hat{X} \in HONEST(\mathbf{C})$. Suppose that R is a run of $\mathcal{Q}$ starting from $\mathbf{C}$, such that $\mathcal{Q}, R \models \mathsf{Verify}(Y, SIG_X\{|x|\})$ for a thread Y such that $\hat{Y} \neq \hat{X}$. By the definition of the execution model, when $\hat{X} \in HONEST(\mathbf{C})$, only threads of $\hat{X}$ can construct signatures with $\hat{X}$'s private key. Since, $\hat{X} \neq \hat{Y}$, by Lemma 2.1 it has to be that the thread Y received term $SIG_X\{|x|\}$ as a part of some message m', i.e. there exists a term m' such that $EVENT(R, Y, (x), m', x)$ and $SIG_X\{|x|\} \subseteq m'$. By Lemma 2.3 there is a corresponding send action for every receive, hence there exists a thread Z such that $EVENT(R, Z, \mathtt{send}\ m, \emptyset, \emptyset)$ is true. Therefore, there exists at least one action in the run R where $SIG_X\{|x|\}$ is sent as a part of some message. Let R' be a shortest prefix of R such that, for some thread Z and for some term m such that $SIG_X\{|x|\} \subseteq m$, it is true that $EVENT(R', Z, \mathtt{send}\ m, \emptyset, \emptyset)$. By Lemma 2.1 $SIG_X\{|x|\}$ has to be either received or generated by Z, since R' is the shortest run in which $SIG_X\{|x|\}$ is sent out as a part of some message it has to be that the thread Z generated $SIG_X\{|x|\}$. By the definition of the execution model, and honesty of $\hat{X}$ it follows that Z is a thread of $\hat{X}$. Now, $Q, R \models \mathsf{Send}(Z, m) \wedge \mathsf{Contains}(m, SIG_Z\{|n|\}))$ holds by the semantics of the action predicates and Lemma 2.2.

Sequencing rule

Sequencing rule **S1** (see Section 4.5) gives us a way of sequentially composing two cords P and P' when post-condition of P, matches the pre-condition or P'. Assume that for all protocols $\mathcal{Q}$ and runs R of $\mathcal{Q}$ both $\mathcal{Q}, R \models \phi_1[P]_A\phi_2$ and $\mathcal{Q}, R \models \phi_2[P']_A\phi_3$ hold. We need to prove that $\mathcal{Q}, R \models \phi_1[PP']_A\phi_3$ for all $\mathcal{Q}$ and R. Let $\mathcal{Q}$ be a protocol and R a run of Q such that $R = R_0R_1R_2$, assume that $R_1|_A$ matches PP' under substitution σ, and $Q, R_0 \models \sigma\phi_1$. Run R can be written as $R = R_0R'_1R''_1R_2$ where $R'_1|_A$ matches P under σ and $R''_1|_A$ matches P' under σ. It follows that $Q, R_0R'_1 \models \sigma\phi_2$ and therefore $\mathcal{Q}, R_0R'_1R''_1 \models \sigma\phi_3$.

The Honesty rule

The honesty rule (see Section 4.8) is an invariance rule for inductively proving properties about the actions of principals that execute protocol roles. Assume that $\mathcal{Q}$ is a protocol and R is a run of $\mathcal{Q}$ such that $\mathcal{Q}, R \models \mathsf{Start}(X)[\ ]_X\phi$ and $\mathcal{Q}, R \models \phi\ [P]_X\ \phi$ for all roles $\rho \in Q$ and for all basic sequences $P \in BS(\rho)$. We must show that $\mathcal{Q}, R \models \mathsf{Honest}(\hat{X}) \supset \phi$. Assume $\mathcal{Q}, R \models \mathsf{Honest}(\hat{X})$. Then by the semantics of predicate "Honest" and Lemma 2.4, it has to be that $R|_X$ is a trace of a role of $\mathcal{Q}$ carried out by X and, moreover, thread X has to be in a pausing state at the end of R. Therefore a $R|_X$ is a concatenation of basic sequences of $\mathcal{Q}$. Now, $\mathcal{Q}, R \models \phi$ follows from the soundness of sequencing rule **S1**.

5 Example

In this section, we use the protocol logic to formally prove the authentication property of the three-way signature based challenge-response protocol (CR) described in Section 2. Our formulation of authentication is based on the concept of *matching conversations* [8] and is similar to the idea of proving authentication using *correspondence assertions* [77]. The same basic idea is also presented in [27] where it is referred to as *matching records of runs*. Simply put, it requires that whenever Alice and Bob accept each other's identities at the end of a run, their records of the run *match*, i.e., each message that Alice sent was received by Bob and vice versa, each send event happened before the corresponding receive event, and moreover the messages sent by each principal appear in the same order in both the records. Here we demonstrate the authentication property only for the initiator in the protocol, proof for the responder can be carried out along the same lines.

Weak authentication

First we show a weaker authentication property. If Alice has completed the initiator role of the protocol, apparently with Bob then Bob was involved in the protocol – he received the first message and sent out the corresponding second message. The formal property proved about the initiator role is

$$\vdash_{Q_{CR}} \top[\mathbf{Init_{CR}}]_X \mathsf{Honest}(\hat{Y}) \wedge \hat{Y} \neq \hat{X} \supset \phi_{weak-auth}.$$

The actions in the modal formula are the actions of the initiator role of CR, given in Section 2. The precondition imposes constraints on the free variables. In this example, the precondition is simply "true". The postcondition captures the security property that is guaranteed by executing the actions starting from a state where the precondition holds. In this specific example, the postcondition is a formula capturing the notion of weak authentication. Intuitively, this formula means that after executing the actions in the initiator role purportedly with $\hat{Y}$, $\hat{X}$ is guaranteed that $\hat{Y}$ was involved in the protocol at some point (purportedly with $\hat{X}$), provided that $\hat{Y}$ is honest (meaning that she always faithfully executes some role of the CR protocol and does not, for example, send out her private keys).

$$\phi_{weak-auth} \equiv \exists Y. \left(\mathsf{Receive}(Y, (\hat{X}, \hat{Y}, m)) < \mathsf{Send}(Y, (\hat{Y}, \hat{X}, y, SIG_{\hat{Y}}\{\!|y, m, \hat{X}|\!\}))\right)$$

A formal proof of the weak authentication property for the initiator guaranteed by executing the CR protocol is presented in Table 5. First order logic reasoning steps as well as applications of the generic rules are omitted for clarity. Details for the application of the honesty rule are postponed until later in this Section. The formal proof naturally breaks down into three parts:

(8)	$\mathbf{AA1}$	$\top[\mathtt{verify}\ s, (y, m, \hat{X}), \hat{Y}]_X \mathsf{Verify}(X, SIG_{\hat{Y}}\{\!\|y, m, \hat{X}\|\!\})$
(9)	$8, \mathbf{P1}, \mathbf{SEQ}$	$\top[\mathbf{Init_{CR}}]_X \mathsf{Verify}(X, SIG_{\hat{Y}}\{\!\|y, m, \hat{X}\|\!\})$
(10)	$9, \mathbf{VER}$	$\top[\mathbf{Init_{CR}}]_X \exists Y, t.\ \mathsf{Send}(Y, t) \wedge \mathsf{Contains}(t, SIG_{\hat{Y}}\{\!\|y, m, \hat{X}\|\!\})$
(11)	$\mathbf{HON_{Q_{CR}}}$	$(\mathsf{Honest}(\hat{Y}) \wedge \mathsf{Send}(Y, t) \wedge \mathsf{Contains}(t, SIG_{\hat{Y}}\{\!\|y, m, \hat{X}\|\!\})) \supset$ $(\mathsf{New}(Y, m) \vee$ $(\mathsf{Receive}(Y, (\hat{X}, \hat{Y}, m)) < \mathsf{Send}(Y, (\hat{Y}, \hat{X}, y, SIG_{\hat{Y}}\{\!\|y, m, \hat{X}\|\!\}))))$
(12)	$10, 11$	$\top[\mathbf{Init_{CR}}]_X \mathsf{Honest}(\hat{Y}) \supset (\exists Y.\ \mathsf{New}(Y, m) \vee$ $(\mathsf{Receive}(Y, (\hat{X}, \hat{Y}, m)) < \mathsf{Send}(Y, (\hat{Y}, \hat{X}, y, SIG_{\hat{Y}}\{\!\|y, m, \hat{X}\|\!\}))))$
(13)	$\mathbf{AA1}$	$\top[\mathtt{new}\ m]_X \mathsf{New}(X, y)$
(14)	$13, \mathbf{P1}, \mathbf{SEQ}$	$\top[\mathbf{Init_{CR}}]_X \mathsf{New}(X, y)$
(15)	$12, 14, \mathbf{AN1}$	$\top[\mathbf{Init_{CR}}]_X \mathsf{Honest}(\hat{Y}) \wedge \hat{Y} \neq \hat{X} \supset (\exists Y.$ $\mathsf{Receive}(Y, (\hat{X}, \hat{Y}, m)) < \mathsf{Send}(Y, (\hat{Y}, \hat{X}, y, SIG_{\hat{Y}}\{\!\|y, m, \hat{X}\|\!\})))$

Table 5
Weak authentication for the initiator role of the CR protocol

- Lines (8)–(10) assert what actions were executed by Alice in the initiator role. Specifically, in this part of the proof, it is proved that Alice has received and verified Bob's signature $SIG_{\hat{Y}}\{\!|y, m, \hat{X}|\!\}$. We then use the fact that the signatures of honest parties are unforgeable (axiom **VER**), to conclude that Bob must have sent out some message containing his signature.
- In lines (11)–(12), the honesty rule is used to infer that whenever Bob generates a signature of this form, he has either generated the nonce m (acting as an initiator) or he sent it to Alice as part of the second message of the protocol and must have previously received the first message from Alice (acting as a responder).
- Finally, in lines (13)–(15), we reason again about actions executed by Alice in order to deduce that the nonce m could not have been created by Bob. Therefore, combining the assertions, we show that the weak authentication property holds: If Alice has completed the protocol as an initiator, apparently with Bob, then Bob must have received the first message (apparently from Alice) and sent the second message to Alice.

Strong authentication

To obtain the stronger authentication property we need to assert temporal ordering between actions of Alice and Bob. As mentioned before, the final authentication property should state that: each message $\hat{X}$ sent was received by $\hat{Y}$ and vice versa, each send event happened before the corresponding receive event, and moreover the messages sent by each principal ($\hat{X}$ or $\hat{Y}$) appear in the same order in both the records. Similarly as before, the formal property proved about the initiator role is $\vdash_{Q_{CR}} \top[\mathbf{Init_{CR}}]_X \mathsf{Honest}(\hat{Y}) \wedge \hat{Y} \neq \hat{X} \supset \phi_{auth}$, but ϕ_{auth} now models the stronger property:

(16)	**AN3**	$\top[\texttt{new}\ m]_X \mathsf{Fresh}(X, m)$
(17)	**FS1**	$\mathsf{Fresh}(X, m)[\texttt{send}\ \hat{X}, \hat{Y}, m]_X \mathsf{FirstSend}(X, m, (\hat{X}, \hat{Y}, m))$
(18)	16, 17, **SEQ**, **P1**	$\top[\mathbf{Init_{CR}}]_X \mathsf{FirstSend}(X, m, (\hat{X}, \hat{Y}, m))$
(19)	18, **FS2**	$\top[\mathbf{Init_{CR}}]_X \mathsf{Receive}(Y, (\hat{X}, \hat{Y}, m)) \wedge \hat{Y} \neq \hat{X} \supset$ $\mathsf{Send}(X, (\hat{X}, \hat{Y}, m)) < \mathsf{Receive}(Y, (\hat{X}, \hat{Y}, m))$
(20)	$\mathbf{HON_{Q_{CR}}}$	$(\mathsf{Honest}(\hat{Y}) \wedge \mathsf{Receive}(Y, (\hat{X}, \hat{Y}, m)) \wedge$ $\mathsf{Send}(Y, (\hat{Y}, \hat{X}, y, SIG_{\hat{Y}}\{\!\vert y, m, \hat{X}\vert\!\}))) \supset$ $\mathsf{FirstSend}(Y, y, (\hat{Y}, \hat{X}, y, SIG_{\hat{Y}}\{\!\vert y, m, \hat{X}\vert\!\}))$
(21)	**AA1**, **AR2**, **SEQ**	$\top[\mathbf{Init_{CR}}]_X \mathsf{Receive}(X, (\hat{Y}, \hat{X}, y, SIG_{\hat{Y}}\{\!\vert y, m, \hat{X}\vert\!\}))$
(22)	20, 21, **FS2**	$\top[\mathbf{Init_{CR}}]_X \mathsf{Honest}(\hat{Y}) \wedge \hat{Y} \neq \hat{X} \wedge$ $\mathsf{Receive}(Y, (\hat{X}, \hat{Y}, m)) \wedge$ $\mathsf{Send}(Y, (\hat{Y}, \hat{X}, y, SIG_{\hat{Y}}\{\!\vert y, m, \hat{X}\vert\!\})) \supset$ $\mathsf{Send}(Y, (\hat{Y}, \hat{X}, y, SIG_{\hat{Y}}\{\!\vert y, m, \hat{X}\vert\!\})) <$ $\mathsf{Receive}(X, (\hat{Y}, \hat{X}, y, SIG_{\hat{Y}}\{\!\vert y, m, \hat{X}\vert\!\}))$
(23)	**AA4**, **P1**	$\top[\mathbf{Init_{CR}}]_X \mathsf{Receive}(X, (\hat{Y}, \hat{X}, y, SIG_{\hat{Y}}\{\!\vert y, m, \hat{X}\vert\!\})) <$ $\mathsf{Send}(X, (\hat{X}, \hat{Y}, SIG_{\hat{X}}\{\!\vert y, m, \hat{Y}\vert\!\}))$
(24)	15, 19, 22, 23	$\top[\mathbf{Init_{CR}}]_X \mathsf{Honest}(\hat{Y}) \wedge \hat{Y} \neq \hat{X} \supset \phi_{auth}$

Table 6
Strong authentication for the initiator role of the CR protocol

$$\begin{aligned}\phi_{auth} \equiv \exists Y.\ ((&\mathsf{Send}(X, msg_1) < \mathsf{Receive}(Y, msg_1)) \wedge \\ &(\mathsf{Receive}(Y, msg_1) < \mathsf{Send}(Y, msg_2)) \wedge \\ &(\mathsf{Send}(Y, msg_2) < \mathsf{Receive}(X, msg_2)) \wedge \\ &(\mathsf{Receive}(X, msg_2) < \mathsf{Send}(X, msg_3)))\end{aligned}$$

Here, we are using msg_1, msg_2 and msg_3 as shortcuts for the corresponding messages in the protocol: $msg_1 \equiv (\hat{X}, \hat{Y}, m)$, $msg_2 \equiv (\hat{Y}, \hat{X}, y, SIG_{\hat{Y}}\{\!\vert y, m, \hat{X}\vert\!\})$, $msg_3 \equiv (\hat{X}, \hat{Y}, SIG_{\hat{X}}\{\!\vert y, m, \hat{Y}\vert\!\})$. Note that we cannot deduce that the responder Y has received the third message as that property does not necessarily hold from the point of view of the initiator.

A formal proof of the strong authentication property for the initiator guaranteed by executing the CR protocol is presented in Table 6. Again, the formal proof naturally breaks down into three parts:

- Lines (16)–(19) reason about actions executed by Alice in the initiator role. Specifically, it is proved that the first occurrence of the nonce m on the network is in the first message send by Alice. Hence, all actions involving that nonce must happen after that send action.
- In lines (20)–(22), the honesty rule is used to infer the symmetrical property about Bob's nonce y. Hence, all actions involving that nonce must happen after

the send action by Bob in the second step of the protocol.

- In line (23) we reason from Alice's actions that she sent out the third message after receiving the second message.
- Finally, in line (24), the weak authentication property already proved is combined with the newly established temporal assertions to infer the final strong authentication property.

The proofs together are an instance of a general method for proving authentication results in the protocol logic. In proving that Alice, after executing the initiator role of a protocol purportedly with Bob, is indeed assured that she communicated with Bob, we usually follow these 3 steps:

(i) Prove the order in which Alice executed her send-receive actions. This is done by examining the actions in Alice's role.

(ii) Assuming Bob is honest, infer the order in which Bob carried out his send-receive actions. This is done in two steps. First, use properties of cryptographic primitives (like signing and encryption) to conclude that only Bob could have executed a certain action (e.g., generate his signature). Then use the honesty rule to establish a causal relationship between that identifying action and other actions that Bob always does whenever he executes that action (e.g, send msg_2 to Alice after having received msg_1 from her).

(iii) Finally, use the temporal ordering rules to establish an ordering between the send-receive actions of Alice and Bob. The causal ordering between messages sent by the peers is typically established by exploiting the fact that messages contain fresh data.

Proofs in the logic are therefore quite insightful. The proof structure often follows a natural language argument, similar to one that a protocol designer might use to convince herself of the correctness of a protocol.

Invariants

In both proofs the honesty rule is used to deduce that the other party in the protocol has performed certain actions or not. Formulas proved by the application of the honesty rule are called *invariants* and will play a crucial role in the composition method described in Section 6. This proof uses two invariants $\mathsf{Honest}(\hat{Y}) \supset \gamma_1$ and $\mathsf{Honest}(\hat{Y}) \supset \gamma_2$ where γ_1 and γ_2 are given by:

$$\begin{aligned}
\gamma_1 \equiv\ & \mathsf{Send}(Y, t) \wedge \mathsf{Contains}(t, SIG_{\hat{Y}}\{\!|y, m, \hat{X}|\!\}) \supset \\
& \quad (\mathsf{Gen}(Y, m) \vee \\
& \quad (\mathsf{Receive}(Y, (\hat{X}, \hat{Y}, m)) < \mathsf{Send}(Y, (\hat{Y}, \hat{X}, y, SIG_{\hat{Y}}\{\!|y, m, \hat{X}|\!\})))) \\
\gamma_2 \equiv\ & (\mathsf{Receive}(Y, (\hat{X}, \hat{Y}, m)) \wedge \mathsf{Send}(Y, (\hat{Y}, \hat{X}, y, SIG_{\hat{Y}}\{\!|y, m, \hat{X}|\!\}))) \supset \\
& \quad \mathsf{FirstSend}(Y, y, (\hat{Y}, \hat{X}, y, SIG_{\hat{Y}}\{\!|y, m, \hat{X}|\!\}))
\end{aligned}$$

As described in Section 4.8, the honesty rule depends on the protocol being analyzed. Recall that the protocol Q_{CR} is just a set of roles $Q_{CR} = \{\mathbf{Init_{CR}}, \mathbf{Resp_{CR}}\}$

each specifying a sequence of actions to be executed. The set of basic sequences of protocol Q_{CR} is given below.

$$\begin{aligned}
\mathbf{BS_1} &\equiv [\texttt{new}\ m; \texttt{send}\ \hat{X}, \hat{Y}, m;]_X \\
\mathbf{BS_2} &\equiv [\texttt{receive}\ \hat{Y}, \hat{X}, y, s; \texttt{verify}\ s, (y, m, \hat{X}), \hat{Y}; \\
&\qquad r := \texttt{sign}\ (y, m, \hat{Y}), \hat{X}; \texttt{send}\ \hat{X}, \hat{Y}, r;]_X \\
\mathbf{BS_3} &\equiv [\texttt{receive}\ \hat{X}, \hat{Y}, x; \texttt{new}\ n; r := \texttt{sign}\ (n, x, \hat{X}), \hat{Y}; \texttt{send}\ \hat{Y}, \hat{X}, n, r;]_Y \\
\mathbf{BS_4} &\equiv [\texttt{receive}\ \hat{X}, \hat{Y}, t; \texttt{verify}\ t, (n, x, \hat{Y}), \hat{X};]_Y
\end{aligned}$$

Therefore to apply the honesty rule we need to show that the invariants (γ_1, γ_2) are preserved by all the basic sequences ($\mathbf{BS}_1$, ..., $\mathbf{BS}_4$). These proofs are straightforward and we omit them here.

6 Protocol Composition

In this section, we explain sequential and parallel composition of protocols as syntactic operations on cords and present associated methods for proving protocol properties compositionally. Recall that a protocol is defined as a finite set of cords, one for each role of the protocol.

Definition 6.1 (Parallel Composition) The parallel composition $\mathcal{Q}_1 \mid \mathcal{Q}_2$ of protocols $\mathcal{Q}_1$ and $\mathcal{Q}_2$ is the union of the sets of cords $\mathcal{Q}_1$ and $\mathcal{Q}_2$.

For example, consider the protocol obtained by parallel composition of SSL 2.0 and SSL 3.0. The definition above allows an honest principal to simultaneously engage in sessions of the two protocols. Clearly, a property proved about either protocol individually might no longer hold when the two are run in parallel, since an adversary might use information acquired by executing one protocol to attack the other. Formally, some step in the logical proof of the protocol property is no longer correct. Since all the axioms and inference rules in Section 4 hold for all protocols, the only formulas used in the proof which might no longer be valid are those proved using the honesty rule, i.e., the protocol invariants. In order to guarantee that the security properties of the individual protocols are preserved under parallel composition, it is therefore sufficient to verify that each protocol respects the invariants of the other. This observation suggests the following four-step methodology for proving properties of the parallel composition of two protocols.

(i) Prove separately the security properties of protocols $\mathcal{Q}_1$ and $\mathcal{Q}_2$.

$$\vdash_{Q_1} \Psi_1 \ \textit{and} \vdash_{Q_2} \Psi_2$$

(ii) Identify the set of invariants used in the two proofs, Γ_1 and Γ_2. The formulas included in these sets will typically be the formulas in the two proofs, that were proved using the honesty rule. The proofs from the previous step can be decomposed into two parts—the first part proves the protocol invariants using the honesty rule for the protocol, while the second proves the protocol property using the invariants as hypotheses, but without using the honesty rule. Formally,

$$\vdash_{Q_1} \Gamma_1 \ and\ \Gamma_1 \vdash \Psi_1 \ and\ \vdash_{Q_2} \Gamma_2 \ and\ \Gamma_2 \vdash \Psi_2$$

(iii) Notice that it is possible to weaken the hypotheses to $\Gamma_1 \cup \Gamma_2$. The proof of the protocol properties is clearly preserved under a larger set of assumptions.

$$\Gamma_1 \cup \Gamma_2 \vdash \Psi_1 \ and\ \Gamma_1 \cup \Gamma_2 \vdash \Psi_2$$

(iv) Prove that the invariants, $\Gamma_1 \cup \Gamma_2$, hold for both the protocols. This step uses the transitivity of entailment in the logic: if $\vdash_Q \Gamma$ and $\Gamma \vdash \gamma$, then $\vdash_Q \gamma$. Since $\vdash_{Q_1} \Gamma_1$ was already proved in Step 1 - in this step it is sufficient to show that $\vdash_{Q_1} \Gamma_2$ and similarly that $\vdash_{Q_2} \Gamma_1$. By Lemma 6.2 below, we therefore have $\vdash_{Q_1|Q_2} \Gamma_1 \cup \Gamma_2$. From this and the formulas from step 3, we can conclude that the security properties of $\mathcal{Q}_1$ and $\mathcal{Q}_2$ are preserved under their parallel composition.

$$\vdash_{Q_1|Q_2} \Psi_1 \ and\ \vdash_{Q_1|Q_2} \Psi_2$$

Lemma 6.2 *If $\vdash_{Q_1} \psi$ and $\vdash_{Q_2} \psi$, then $\vdash_{Q_1|Q_2} \psi$, where the last step in the proof of ψ in both $\mathcal{Q}_1$ and $\mathcal{Q}_2$ uses the honesty rule and no previous step uses the honesty rule.*

Proof. Following the conclusion of the honesty rule, ψ must be of the form $\mathsf{Honest}(\hat{X}) \supset \phi$ for some formula ϕ. Suppose that the formula $\mathsf{Honest}(\hat{X}) \supset \phi$ can be proved for both $\mathcal{Q}_1$ and $\mathcal{Q}_2$ using the honesty rule. By the definition of the honesty rule, it has to be that $\vdash \mathsf{Start}(X)[]_X\phi$ and $\forall \rho \in \mathcal{Q}_1 \cup \mathcal{Q}_2 . \forall P \epsilon BS(\rho).\ \vdash \phi\ [P]_X\ \phi$. Every basic sequence P of a role in $\mathcal{Q}_1 \mid \mathcal{Q}_2$ is a basic sequence of a role in $\mathcal{Q}_1$, or a basic sequence of a role in $\mathcal{Q}_2$. It follows that $\vdash \phi\ [P]_X\ \phi$ and, therefore, by the application of the honesty rule, $\vdash_{\mathcal{Q}_1|\mathcal{Q}_2} \mathsf{Honest}(\hat{X}) \supset \phi$. □

Theorem 6.3 *If $\vdash_{Q_1} \Gamma$ and $\Gamma \vdash \Psi$ and $\vdash_{Q_2} \Gamma$, then $\vdash_{Q_1|Q_2} \Psi$.*

Definition 6.4 (Sequential Composition of Cords) Given cords

$$r = (x_0 \ldots x_{\ell-1})[R]_X(u_0 \ldots u_{m-1}),$$

$$s = (y_0 \ldots y_{m-1})[S]_Y(t_0 \ldots t_{n-1}),$$

their sequential composition is defined by

$$r; s = (x_0 \ldots x_{\ell-1})[RS']_X(t'_0 \ldots t'_{n-1}),$$

where S' and t'_i are the substitution instances of S and t_i respectively, such that each variable y_k is replaced by the term u_k. Furthermore, under this substitution, Y is mapped to X. Variables are renamed so that free variables of S, t_j and u_k do not become bound in $r; s$. RS' is the strand obtained by concatenating the actions in R with those in S'.

Definition 6.5 (Sequential Composition) A protocol $\mathcal{Q}$ is the sequential composition of two protocols $\mathcal{Q}_1$ and $\mathcal{Q}_2$ if each role of $\mathcal{Q}$ is obtained by the sequential composition of a cord of $\mathcal{Q}_1$ with a cord of $\mathcal{Q}_2$.

It is clear that the sequential composition of protocols does not yield a unique result. Typically, when we sequentially compose protocols we have a specific composition of roles in mind. For example, if we compose two two-party protocols, we might compose the corresponding initiator and responder roles.

The sequencing rule, **S1** (see Section 4), is the main rule used to construct a modular correctness proof of a protocol that is a sequential composition of several smaller subprotocols. It 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. In addition, just like in parallel composition, it is essential that the composed protocols respect each other's invariants. Our methodology for proving properties of the sequential composition of two protocols involves the following steps.

(i) Prove separately the security properties of protocols $\mathcal{Q}_1$ and $\mathcal{Q}_2$.

$$\vdash_{Q_1} \Psi_1 \ \textit{and} \ \vdash_{Q_2} \Psi_2$$

(ii) Identify the set of invariants used in the two proofs, Γ_1 and Γ_2. The formulas included in these sets will typically be the formulas in the two proofs, which were proved using the honesty rule. The proofs from the previous step can be decomposed into two parts—the first part proves the protocol invariants using the honesty rule for the protocol, while the second proves the protocol property using the invariants as hypotheses, but without using the honesty rule. Formally,

$$\vdash_{Q_1} \Gamma_1, \Gamma_1 \vdash \Psi_1 \ \textit{and} \ \vdash_{Q_2} \Gamma_2, \Gamma_2 \vdash \Psi_2$$

(iii) Weaken the hypotheses to $\Gamma_1 \cup \Gamma_2$. The proof of the protocol properties is clearly preserved under a larger set of assumptions.

$$\Gamma_1 \cup \Gamma_2 \vdash \Psi_1 \ \textit{and} \ \Gamma_1 \cup \Gamma_2 \vdash \Psi_2$$

(iv) If the post-condition of the modal formula Ψ_1 matches the pre-condition of Ψ'_2, then the two can be sequentially composed by applying the sequencing rule **S1**. Here Ψ'_2 is obtained from Ψ_2 by a substitution of the free variables determined by the sequential composition of the corresponding cords. This preserves the formulas proved in the previous steps since those formulas are true under all substitutions of the free variables. Assuming that Ψ_1 and Ψ'_2 are respectively $\theta[P_1]_X\phi$ and $\phi[P_2]_X\psi$, we have:

$$\Gamma_1 \cup \Gamma'_2 \vdash \theta[P_1P_2]_X\psi$$

(v) Prove that the invariants used in proving the properties of the protocols, $\Gamma_1 \cup \Gamma'_2$, hold for both the protocols. Since $\vdash_{Q_1} \Gamma_1$ was already proved in Step 1, in this step, it is sufficient to show that $\vdash_{Q_1} \Gamma'_2$ and similarly that $\vdash_{Q_2} \Gamma_1$. By Lemma 6.6, we therefore have $\vdash_{Q_3} \Gamma_1 \cup \Gamma'_2$, where $\mathcal{Q}_3$ is their sequential composition. From this and the formulas from steps 3 and 4, we can conclude that the security properties of $\mathcal{Q}_1$ and $\mathcal{Q}_2$ are preserved under their sequential composition and furthermore the following formula is provable.

$$\vdash_{Q_3} \theta[P_1P_2]_X\psi$$

Lemma 6.6 *If $\vdash_{Q_1} \psi$ and $\vdash_{Q_2} \psi$, then $\vdash_{Q_3} \psi$, where $\mathcal{Q}_3$ is a sequential composition of $\mathcal{Q}_1$ and $\mathcal{Q}_2$, and the last step in the proof of ψ in both $\mathcal{Q}_1$ and $\mathcal{Q}_2$ uses the honesty rule and no previous step uses the honesty rule.*

Proof. Following the conclusion of the honesty rule, ψ must be of the form $\mathsf{Honest}(\hat{X}) \supset \phi$ for some formula ϕ. Suppose that the formula $\mathsf{Honest}(\hat{X}) \supset \phi$ can proved in both $\mathcal{Q}_1$ and $\mathcal{Q}_2$ using the honesty rule. By the definition of the honesty rule, it has to be that $\vdash \mathsf{Start}(X)\ [\,]_X\ \phi$ and $\forall \rho \in \mathcal{Q}_1 \cup \mathcal{Q}_2 . \forall P \epsilon BS(\rho).\ \vdash \phi\ [P]_X\ \phi$. Let $\mathcal{Q}$ be a protocol obtained by the sequential composition of $\mathcal{Q}_1$ and $\mathcal{Q}_2$. Every basic sequence P of a role in $\mathcal{Q}$ has to be a basic sequence of a role in $\mathcal{Q}_1$, or a basic sequence of a role in $\mathcal{Q}_2$, or a concatenation of a basic sequence of a role in $\mathcal{Q}_1$ and a basic sequence of a role in $\mathcal{Q}_2$. In the first two cases, $\vdash \phi\ [P]_X\ \phi$ holds trivially, in the third case $\vdash \phi\ [P]_X\ \phi$ follows by one application of the sequencing rule **S1**. Therefore, by the application of the honesty rule, $\vdash_{\mathcal{Q}} \mathsf{Honest}(\hat{X}) \supset \phi$. □

Theorem 6.7 *If $\vdash_{Q_1} \Gamma_1$, $\Gamma_1 \vdash \theta[P_1]_X\phi$; $\vdash_{Q_2} \Gamma_2$, $\Gamma_2 \vdash \phi[P_2]_X\psi$; and $\vdash_{Q_1} \Gamma_2$, $\vdash_{Q_2} \Gamma_1$, then $\vdash_{Q_3} \theta[P_1P_2]_X\psi$, where $\mathcal{Q}_3$ is a sequential composition of $\mathcal{Q}_1$ and $\mathcal{Q}_2$.*

In the proof technique just described, the invariants sufficient for the proofs are independent of the order of the roles in the sequential composition. However, in many situations the knowledge of the information flow induced by the particular ordering facilitates proofs in an intuitive and effective way. Suppose $\mathcal{Q}$ is a sequential composition of protocols $\mathcal{Q}_1$ and $\mathcal{Q}_2$, and $r_1; r_2$ is a role of $\mathcal{Q}$ where r_1 and r_2 are roles of $\mathcal{Q}_1$ and $\mathcal{Q}_2$ respectively. In proving the invariance of a formula ϕ over the protocol segment r_2 we will use some history information from the prior execution of r_1. In the technical presentation below this history information appears as the preconditions θ_{r_i}. The *invariant induction* is the usual induction for the honesty rule strengthened by the preconditions. The *precondition induction* ensures that the preconditions employed actually hold at the corresponding state of protocol execution. This theorem builds on ideas developed in [38,68,67]to prove security properties over the complex control flow architectures of IEEE 802.11i and Kerberos V5.

Theorem 6.8 *If $\mathcal{Q}$ is a sequential composition of protocols $\mathcal{Q}_1$ and $\mathcal{Q}_2$ then we can conclude $\vdash_{\mathcal{Q}} \mathsf{Honest}(\hat{X}) \supset \phi$ if the following conditions hold for all $r_1; r_2$ in $\mathcal{Q}$, where $r_1 \in \mathcal{Q}_1$ and $r_2 \in \mathcal{Q}_2$:*

(i) *(Invariant induction)*
- $\forall P \in BS(r_1).\ \vdash \theta_{r_1} \wedge \phi\ [P]_X\ \phi$ *and* $\forall P \in BS(r_2).\ \vdash \theta_{r_2} \wedge \phi\ [P]_X\ \phi$

(ii) *(Precondition induction)*

- $\vdash \mathsf{Start}(X)[\,]_X \theta_{r_1}$ *and* $\vdash \theta_{r_1}\; r_1\; \theta_{r_2}$
- $\forall P \in BS(r_1).\ \vdash \theta_{r_1}[P]_X\ \theta_{r_1}$ *and* $\forall P \in BS(r_2).\ \vdash \theta_{r_2}[P]_X\ \theta_{r_2}$

With this background, the modified proof method can be divided into the following stages:

(i) Prove separately the security properties Ψ_1 and Ψ_2 assuming the set of invariants Γ_1 and Γ_2 respectively. Formally,

$$\Gamma_1 \vdash \Psi_1 \ \ and \ \ \Gamma_2 \vdash \Psi_2$$

(ii) Weaken the hypotheses to $\Gamma_1 \cup \Gamma_2$. The proof of the protocol properties is clearly preserved under a larger set of assumptions.

$$\Gamma_1 \cup \Gamma_2 \vdash \Psi_1 \ \ and \ \ \Gamma_1 \cup \Gamma_2 \vdash \Psi_2$$

(iii) If the post-condition of the modal formula Ψ_1 matches the pre-condition of Ψ_2', then the two can be sequentially composed by applying the sequencing rule **S1**. Here Ψ_2' is obtained from Ψ_2 by a substitution of the free variables determined by the sequential composition of the corresponding cords. This preserves the formulas proved in the previous steps since those formulas are true under all substitutions of the free variables. Assuming that Ψ_1 and Ψ_2' are respectively $\theta[P_1]_X\phi$ and $\phi[P_2]_X\psi$, we have:

$$\Gamma_1 \cup \Gamma_2' \vdash \theta[P_1 P_2]_X \psi$$

(iv) Prove that the invariants used in proving the properties of the protocols, $\Gamma_1 \cup \Gamma_2'$, hold for $\mathcal{Q}$ using theorem 6.8. From this and step (iii), we can conclude:

$$\vdash_Q \theta[P_1 P_2]_X \psi$$

6.1 An Example of Protocol Composition

In this section we demonstrate the protocol composition methodology with an example. First we show how the *ISO-9798-3* key exchange protocol can be obtained by composing an abstract signature-based challenge-response protocol and a simple protocol based on Diffie-Hellman key exchange. Next we show how the authentication properties and secrecy properties of the *ISO-9798-3* protocol are proved from properties of its parts. Extensions to the logic for handling Diffie-Hellman primitives are presented in Appendix A.

6.1.1 ISO-9798-3 Protocol as a Sequential Composition

Figure 3 shows the *ISO-9798-3* protocol in the informal arrows-and-messages notation. The goal of the protocol is to obtain an authenticated shared secret between the two parties. Mutual authentication is provided using exactly the same mechanism as in the CR protocol except two fresh Diffie-Hellman exponentials are

$$
\begin{aligned}
A &\rightarrow B : g^a \\
B &\rightarrow A : g^b, SIG_B\{\!|g^b, g^a, A|\!\} \\
A &\rightarrow B : SIG_A\{\!|g^b, g^a, B|\!\}
\end{aligned}
$$

Fig. 3. *ISO-9798-3* protocol as arrows-and-messages

exchanged instead of nonces. Authenticated shared secret is obtained combining the authentication property with the properties of the Diffie-Hellman key exchange primitive.

The CR_0 protocol can be thought of as an abstraction of the CR protocol described in Section 2 and analyzed in Section 5. While the CR protocol generates new nonces in each role, the roles of the CR_0 protocol obtains terms m and n via the input interface and use them in place of nonces. Intuition here is that the authentication property of the CR protocol only depends on the fact that m and n are *fresh* in the sense that no other threads may use them until they are send by the originating threads, while the exact structure of terms m and n is irrelevant. Roles of the CR_0 protocol are given below:

```
$\mathbf{Init_{CR_0}} \equiv (\hat{Y}, m)[$
        $\texttt{send}\ \hat{X}, \hat{Y}, m;$
        $\texttt{receive}\ \hat{Y}, \hat{X}, y, s;$
        $\texttt{verify}\ s, (y, m, \hat{X}), \hat{Y};$
        $r := \texttt{sign}\ (y, m, \hat{Y}), \hat{X};$
        $\texttt{send}\ \hat{X}, \hat{Y}, r;$
    $]_X()$
```

```
$\mathbf{Resp_{CR_0}} \equiv (n)[$
        $\texttt{receive}\ \hat{X}, \hat{Y}, x;$
        $r := \texttt{sign}\ (n, x, \hat{X}), \hat{Y};$
        $\texttt{send}\ \hat{Y}, \hat{X}, n, r;$
        $\texttt{receive}\ \hat{X}, \hat{Y}, t;$
        $\texttt{verify}\ t, (n, x, \hat{Y}), \hat{X};$
    $]_Y()$
```

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. It can be represented by a single role that computes the new exponent and outputs the corresponding nonce via the output interface.

$$\mathbf{Init_{DH_0}} \equiv \mathbf{Resp_{DH_0}} \equiv [\texttt{new}\ x; gx := \texttt{expg}\ x]_X(gx)$$

The *ISO-9798-3* protocol is a sequential composition of these two protocols. The cords of *ISO-9798-3* are obtained by sequential composition of the cord of DH_0 with the two cords of CR_0. When sequentially composing cords, we substitute the output parameters of the first cord for the input parameters of the second and α-rename bound variables to avoid variable capture. The roles of the *ISO-9798-3* protocol are therefore:

```
Init_ISO ≡ (Ŷ, m)[                         Resp_ISO ≡ (n)[
        new x;                                     new y;
        gx := expg x;                              gy := expg y;
        send X̂, Ŷ, gx;                             receive X̂, Ŷ, x;
        receive Ŷ, X̂, y, s;                        r := sign (gy, x, X̂), Ŷ;
        verify s, (y, gx, X̂), Ŷ;                   send Ŷ, X̂, gy, r;
        r := sign (y, gx, Ŷ), X̂;                   receive X̂, Ŷ, t;
        send X̂, Ŷ, r;                              verify t, (gy, x, Ŷ), X̂;
]_X()                                      ]_Y()
```

6.1.2 Compositional Proof Sketch

As we just demonstrated, the *ISO-9798-3* protocol can be constructed by a sequential composition of DH_0 and CR_0. Here, we describe the key secrecy property of DH_0 and the mutual authentication property of CR_0. 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. In doing so, we follow the method for proving sequential composition results presented in the previous section.

Challenge Response Protocol, CR

A proof of the mutual authentication property guaranteed by executing the CR_0 protocol is essentially the same as the proof for the CR protocol presented in Section 5. The difference is that we use preconditions instead of the explicit `new` actions to deduce the freshness of m. The property proved for the CR_0 protocol is:

$$\Gamma_2 \vdash \mathsf{Fresh}(X, m)[\mathbf{Init_{CR_0}}]_X \mathsf{Honest}(\hat{Y}) \wedge \hat{Y} \neq \hat{X} \supset \phi_{auth}$$

Here, ϕ_{auth} models the authentication property, while Γ_2 contains an appropriate modification of the two invariants γ_1 and γ_2 used in the proof as described in Section 5:

$$\begin{aligned}\gamma_1 \equiv\ & \mathsf{Send}(Y, t) \wedge \mathsf{Contains}(t, SIG_{\hat{Y}}\{\!|y, m, \hat{X}|\!\}) \supset \\ & \quad (\mathsf{Gen}(Y, m) \vee \\ & \quad (\mathsf{Receive}(Y, (\hat{X}, \hat{Y}, m)) < \mathsf{Send}(Y, (\hat{Y}, \hat{X}, y, SIG_{\hat{Y}}\{\!|y, m, \hat{X}|\!\})))) \\ \gamma_2 \equiv\ & (\mathsf{Receive}(Y, (\hat{X}, \hat{Y}, m)) \wedge \mathsf{Send}(Y, (\hat{Y}, \hat{X}, y, SIG_{\hat{Y}}\{\!|y, m, \hat{X}|\!\}))) \supset \\ & \quad \mathsf{FirstSend}(Y, y, (\hat{Y}, \hat{X}, y, SIG_{\hat{Y}}\{\!|y, m, \hat{X}|\!\}))\end{aligned}$$

Base Diffie-Hellman Protocol, DH_0

The property of the initiator role of the DH_0 protocol is given by the formula below. We use $\mathsf{HasAlone}$ as a shortcut expressing that a term is private to a thread, i.e. $\mathsf{HasAlone}(X,t) \equiv \mathsf{Has}(X,t) \wedge (\mathsf{Has}(Y,t) \supset X = Y$.

$$\Gamma_1 \vdash \mathsf{Start}(X)[\mathtt{new}\ x; gx := \mathtt{expg}\ x]_X \mathsf{HasAlone}(X,x) \wedge \mathsf{Fresh}(X,gx)$$

This formula follows easily from the axioms and rules of the logic. It states that after carrying out the initiator role of DH_0, X possesses a fresh Diffie-Hellman exponential g^x and is the only one who possesses the exponent x. This property will be useful in proving the secrecy condition of the *ISO-9798-3* protocol. The set of invariants used in this proof, Γ_1, 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_0. In doing so, we follow the methodology for proving sequential composition results outlined in Section 6. Let us go back and look at the form of the logical formulas characterizing the initiator roles of DH_0 and CR_0. We have:

$$\Gamma_1 \vdash \mathsf{Start}(X)\ [\mathbf{Init_{DH_0}}]_X\ \mathsf{Fresh}(X,gx)$$
$$\Gamma_2 \vdash \mathsf{Fresh}(X,m)\ [\mathbf{Init_{CR_0}}]_X\ \mathsf{Honest}(\hat{Y}) \wedge \hat{Y} \neq \hat{X} \supset \phi_{auth}$$

At this point, Step 1 and Step 2 of the proof method are complete. For Step 3, we note that since Γ_1 is empty, $\Gamma_2 \cup \Gamma_1$ is simply Γ_2.

(25) $\Gamma_2 \vdash \mathsf{Start}(X)\ [\mathbf{Init_{DH_0}}]_X\ \mathsf{Fresh}(X,gx)$

(26) $\Gamma_2 \vdash \mathsf{Fresh}(X,m)\ [\mathbf{Init_{CR_0}}]_X\ \mathsf{Honest}(\hat{Y}) \wedge \hat{Y} \neq \hat{X} \supset \phi_{auth}$

We are ready to move on to Step 4. We first substitute the output parameters of the initiator cord for DH_0 for the input parameters of the initiator cord of CR_0. This involves substituting gx for m. We refer to the modified protocol as CR_0'. Since the validity of formulas is preserved under substitution, the following formula is valid.

$$\Gamma_2[gx/m] \vdash \mathsf{Fresh}(X,gx)\ \left[\mathbf{Init_{CR_0'}}\right]_X\ \mathsf{Honest}(\hat{Y}) \wedge \hat{Y} \neq \hat{X} \supset \phi_{auth}[gx/m]$$

Note that the post-condition of 25 matches the pre-condition of 26. We can therefore compose the two formulas by applying the sequencing rule **S1**. The resulting formula is:

$$\Gamma_2[gx/m] \vdash \mathsf{Start}(X)\ \left[\mathbf{Init_{DH_0}}; \mathbf{Init_{CR_0'}}\right]_X\ \mathsf{Honest}(\hat{Y}) \wedge \hat{Y} \neq \hat{X} \supset \phi_{auth}[gx/m]$$

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_0 is guaranteed assuming that the invariants in Γ_2 are satisfied. Finally, we use theorem 6.8 with the following preconditions to establish the invariants Γ_2:

For γ_1 :

$$\theta_{Init_{DH_0}} \equiv \theta_{Resp_{DH_0}} \equiv \theta_{Resp_{CR_0}} \equiv \top$$
$$\theta_{Init_{CR_0}} \equiv \mathsf{Gen}(X,m)$$

For γ_2 :

$$\theta_{Init_{DH_0}} \equiv \theta_{Resp_{DH_0}} \equiv \theta_{Init_{CR_0}} \equiv \top$$
$$\theta_{Resp_{CR_0}} \equiv \mathsf{Fresh}(Y, n)$$

Therefore, we conclude that the protocol *ISO-9798-3* , a sequential composition of DH_0 and CR_0 respects the invariants in Γ_2. This completes the compositional proof for the mutual authentication property.

$$\vdash_{Q_{ISO}} \mathsf{Start}(X)\, [\mathbf{Init_{ISO}}]_X\, \mathsf{Honest}(\hat{Y}) \wedge \hat{Y} \neq \hat{X} \supset \phi_{auth}[gx/m]$$

The other main step involves proving that the secrecy property of DH_0 is preserved under sequential composition with CR_0, since CR_0' does not reveal the Diffie-Hellman exponents. The following two formulas are easily provable.

$$\vdash \mathsf{Start}(X)\, [\mathbf{Init_{DH_0}}]_X\, \mathsf{HasAlone}(X, x)$$
$$\vdash \mathsf{HasAlone}(X, x)\, \Big[\mathbf{Init_{CR_0'}}\Big]_X\, \mathsf{HasAlone}(X, x)$$

Therefore, by applying the sequencing rule **S1** again, we have the secrecy condition for the *ISO-9798-3* protocol:

$$\vdash \mathsf{Start}(X)\, \Big[\mathbf{Init_{DH_0}}; \mathbf{Init_{CR_0'}}\Big]_X\, \mathsf{HasAlone}(X, x)$$

Since the set of invariants is empty, Step 2, Step 3 and Step 5 follow trivially. The rest of the proof uses properties of the Diffie-Hellman method of secret computation to prove the following logical formula:

$$(27)\ \vdash_{Q_{ISO}} \mathsf{Start}(X)\, \Big[\mathbf{Init_{DH_0}}; \mathbf{Init_{CR_0'}}\Big]_X\, \mathsf{Honest}(\hat{Y}) \supset$$
$$\exists Y, y.\ (\mathsf{Has}(X, g^{xy}) \wedge (\mathsf{Has}(Z, g^{xy}) \supset (Z = X \vee Z = Y)))$$

Intuitively, the property proved is that if $\hat{Y}$ is honest, then $\hat{X}$ and $\hat{Y}$ are the only people who know the Diffie-Hellman secret g^{xy}. In other words, the *ISO-9798-3* protocol can be used to compute an authenticated shared secret.

7 Other Results

In this section, we summarize other results associated with PCL and point the interested reader to the relevant articles for further details.

7.1 PCL Proof Methods

In [22], we extend PCL with higher-order features (function variables) and present an *abstraction-refinement proof method* for reasoning about security protocols. The main idea is to view changes in a protocol as a combination of finding a meaningful "protocol template" that contains function variables in messages, and producing the refined protocol as an instance of the template. Using higher-order protocol logic, we can develop a single proof for all instances of a template. A template can also be instantiated to another template, or a single protocol may be an instance of more than one template, allowing separate protocol properties to be proved modularly. To give a simple example, suppose we have a protocol containing messages that use symmetric encryption, and suppose that some useful property of this protocol is preserved if we replace symmetric encryption by use of a keyed hash. We can capture the relationship between these two protocols by writing an "abstract" protocol template with function variables in the positions occupied by either encryption or keyed hash. Then the two protocols of interest become instances of the template. In addition, a similar relationship often works out for protocol proofs. If we start with a proof of some property of the protocol that contains symmetric encryption, some branches of the proof tree will establish properties of symmetric encryption that are used in the proof. If we replace symmetric encryption by a function variable, then the protocol proof can be used to produce a proof about the protocol template containing function variables. This is accomplished by replacing each branch that proves a property of symmetric encryption by a corresponding hypothesis about the function variable. Once we have a proof for the protocol template obtained by abstracting away the specific uses of symmetric encryption, we can consider replacing the function variable with keyed hash. If keyed hash has the properties of symmetric encryption that were used in the initial proof, we can use proofs of these properties of keyed hash in place of the assumptions about the function variable. Thus an abstraction step and an instantiation step bring us both from a protocol with symmetric encryption to a protocol with keyed hash, and from a proof of the initial protocol to a proof of the final one. The role of the protocol template in this process is to provide a unified proof that leads from shared properties of two primitives (symmetric encryption or keyed hash) to a protocol property that holds with either primitive.

While the current paper focuses on authentication proofs, we have also developed a proof method for establishing secrecy properties [68]. Our general approach involves showing that every protocol agent that receives data protected by one of a chosen set of encryption keys only sends sensitive data out under encryption by another key in the set. This reduces a potentially complicated proof about arbitrary runs involving arbitrarily many agents and a malicious attacker to a case-by-case analysis of how each protocol step might save and send data. We formalize this form of inductive reasoning about secrecy in a set of new axioms and inference rules that are added to PCL and prove soundness of the system over a conventional symbolic protocol execution model. The extended logic may be used to prove authentication or secrecy, independently and in situations where one property may depend upon the other. Among other challenges, the inductive secrecy rule presented here is carefully designed to be sound for reasoning about arbitrarily many simultaneous protocols sessions, and powerful enough to prove meaningful properties about complex protocols used in practice. While the reasoning principles are similar to the "rank function method" [71] and work using the strand space execution model [74], our main technical contribution is a set of mechanizable formal rules that codify the non-formal mathematical arguments in these earlier papers. Another point of technical difference is that we carry out our induction only over the steps of the protocol without requiring any explicit reasoning over possible actions of a malicious attacker.

7.2 PCL Applications

PCL has been used to analyze a number of industrial security protocols including the IEEE 802.11i wireless LAN security standard [38] (of which SSL/TLS is a component), Kerberos V5 [68], and the IETF GDOI standard for secure group communication [56].

The IEEE 802.11i standard allows a network access point to mutually authenticate itself with user devices before providing connectivity. The protocol consists of several parts, including an 802.1X authentication phase using TLS over EAP, a 4-Way Handshake to establish a fresh session key, and an optional Group Key Handshake for group communications. Motivated by previous vulnerabilities in related wireless protocols and evolution in 802.11i to provide better security, we carry out a formal proof of correctness using PCL. Our proof consists of separate proofs of specific security properties for 802.11i components - the TLS authentication phase, the 4-Way Handshake protocol and the Group Key Handshake protocol. Using a new form of PCL composition principle, formulated as *staged composition* in this paper, we combine the component proofs to show that any staged use of the protocol components achieves the stated security goals. It follows that the components compose securely for a range of failure recovery control flows, including the improvements proposed in [37]. The general result also proves security for other configurations presented in the 802.11i Standards document, including the use of a Pre-Shared Key (PSK) or cached Pair-wise Master Key (PMK). In addition to devising a new composition principle for PCL, we also extend the logic to handle local memory associated with reusing generated nonces. The memory feature is needed to prove correctness of an unusual feature of the improved 4-Way Handshake protocol [37] that involves reusing a nonce to avoid a Denial of Service (DoS) attack. Furthermore, the formal proof for the TLS protocol has independent interest since TLS is widely used independent of 802.11i (e.g. [75]).

Kerberos [43] is widely used for authenticated client-server interaction in local area networks. The basic protocol has three sections, each involving an exchange between the client and a different service. In recent work [68], we develop a formal proof that is modular, with the proof for each section assuming a precondition and establishing a postcondition that implies the precondition of the following section. One advantage of this modular structure is illustrated by our proof for the PKINIT [18] version that uses public-key infrastructure instead of shared secret keys in the initial steps. Since only the first section of PKINIT is different, the proofs for the second and third sections of the protocol remain unchanged. While lengthy machine-checked proofs of Kerberos were previously given [7], and non-formal mathematical proofs have been developed for other abstractions of Kerberos [15], this is the first concise formal logic proof of secrecy and authentication for Kerberos and PKINIT.

7.3 Computational PCL

While the work described so far is in the symbolic model of protocol execution and attack (also called the “Dolev-Yao” model), we have also developed *Computational*

PCL—a logic which is sound wrt the complexity-theoretic model of modern cryptography [25,26,67]. Computational PCL inherits its syntax and reasoning methods from PCL. However, the semantics of the logic is defined wrt to a probabilistic polynomial time model of protocol execution and attack.

Our central organizing idea (cf. [25]) is to interpret formulas as operators on probability distributions on traces. Informally, representing a probability distribution by a set of equi-probable traces (each tagged by the random sequence used to produce it), the meaning of a formula ϕ on a set T of traces is the subset $T' \subseteq T$ in which ϕ holds. This interpretation yields a probability: the probability that ϕ holds is the ratio $|T'|/|T|$. Conjunction and disjunction are simply intersection and union. There are several possible interpretations for implication, and it is not clear at this point which will prove most fruitful in the long run. Currently, we interpret $\phi \Rightarrow \psi$ as the union of $\neg\phi$ and the composition of ψ with ϕ; the latter uses the conditional probability of ψ given ϕ. This interpretation supports a soundness proof for a sizable fragment of the protocol logic, and resembles the probabilistic interpretation of implication in [62]. Since the logic does not mention probability explicitly, we consider a formula "true" if it holds with asymptotically overwhelming probability.

In subsequent work, we formulate a specification of secure key exchange that is closed under general composition with steps that use the key and use the logic to establish security properties of the ISO-9798-3 protocol [26]. We also develop a proof method for establishing computational secrecy properties and apply it to the Kerberos V5 protocol [67].

8 Related Work

A variety of methods and tools have been developed for analyzing the security guarantees provided by network protocols. The main lines of work include specialized logics [13,73,32], process calculi [2,1,44,65] and tools [53,72], as well as theorem-proving [64,63] and model-checking methods [45,59,66,70,10,3] using general purpose tools. (The cited papers are representative but not exhaustive; see [55] for a more comprehensive survey.)

There are several points of difference among these approaches. While most model-checking tools can only analyze a finite number of concurrent sessions of a protocol, some of the logics, process calculi, and theorem-proving techniques yield protocol security proofs without bounding the number of sessions. With the exception of the BAN family of logics [13], most approaches involve explicit reasoning about possible attacker actions. Finally, while security properties are interpreted over individual traces in the majority of these methods, in the process calculi-based techniques, security is defined by an equivalence relation between a real protocol and an ideal protocol, which is secure by construction. Inspite of these differences, all of these approaches use the same symbolic model of protocol execution and attack. This model seems to have developed from positions taken by Needham-Schroeder [61], Dolev-Yao [28], and much subsequent work by others.

PCL shares several features with BAN [13], a specialized protocol logic. It is designed to be a logic for authentication, with relevant secrecy concepts. Both logics annotate programs with assertions and use formulas for concepts like “freshness”, “sees”, “said”, and “shared secret”. Furthermore, neither logic requires explicit reasoning about the actions of an attacker.

On the other hand, PCL differs from BAN on some aspects since it addresses known problems with BAN. BAN had an abstraction step in going from the program for the protocol to its representation as a logical formula. PCL avoids the abstraction phase since formulas contain the program for the protocol. PCL uses a dynamic logic set-up: after a sequence of actions is executed, some property holds in the resulting state. It is formulated using standard logical concepts: predicate logic and modal operators, with more or less standard semantics for many predicates and modalities. Temporal operators can be used to refer specifically to actions that have happened and the order in which they occurred. Formulas are interpreted over traces and the proof system is sound with respect to the standard symbolic model of protocol execution and attack. On the other hand, BAN was initially presented without semantics. Although subsequently, model-theoretic semantics was defined, the interpretation and use of concepts like “believes” and “jurisdiction” remained unclear. Finally, PCL formulas refer to specific states in protocol. For example, x may be fresh at one step, then no longer fresh. In contrast, BAN statements are persistent making it less expressive.

PCL also shares several common points with the Inductive Method [64]. Both methods use the same trace-based model of protocol execution and attack; proofs use induction and provable protocol properties hold for an unbounded number of sessions. One difference is the level of abstraction. Paulson reasons explicitly about traces including possible intruder actions whereas basic reasoning principles are codified in PCL as axioms and proof rules. Proofs in PCL are significantly shorter and do not require any explicit reasoning about an intruder. Finally, while Paulson’s proofs are mechanized using Isabelle, most proofs in PCL are hand-proofs. However, PCL is amenable to automation and a tool implementation effort is underway. An interesting recent effort that is similar to PCL, and the use of templates for abstract protocols mentioned in Section 7.1, is reported in [4].

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 [33] or something very similar to it [39]. 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 [54], 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 [74]. 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 [31]. A somewhat different approach is used by Lynch [46] 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 [17], 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 when they are run in parallel with certain other protocols. This study further reinforces the importance of methods for reasoning about the composability of protocols. We believe that the results presented in this dissertation represent significant progress in this direction. The methods presented in Section 6 provide a way to implicitly characterize, using invariants, a class of protocols with which a specific protocol can be safely composed. In particular, our formalism justifies some of the design principles discussed by the authors. One recommendation is 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. If this is indeed the case, then in our formalism, the invariant for the protocol under consideration would fail to hold in such an environment, and the composition proof would therefore not go through. However, this seems like an overly conservative design approach since not every two protocols which use the same encryption keys interfere with each other's security. The invariant-preservation method can help identify protocols which can run safely in parallel even if they share keys. We note that the above principle has been followed in the design of real-world protocols like IKE [36]. Also, Guttman and Fábrega have proved a theoretical result to the same effect in their strand space model [34]. Another rule of thumb (also recommended by Kelsey, Schneier and Wagner in [42]), 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 idea is also founded on similar intuition. To give an example, in our logic, an invariant in proving an authentication property could be: "if Bob generated a signature of a particular form, he sent it in response to a particular message of a protocol"; adding the unique protocol identifier inside the signature will ensure that this invariant is trivially satisfied for all other protocols, thereby allowing composability. However, many existing protocols do not follow this principle.

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 modelling techniques [48,49,50,51,52], 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 these negative results about composability.

There are some important differences between the way that we reason about incremental protocol construction and alternative approaches such as “universal composability” [16]. In universal composability, properties of a protocol are stated in a strong form so that the property will be preserved under a wide class of composition operations. In contrast, our protocol proofs proceed from various assumptions, including invariants that are assumed to hold in any environment in which the protocol operates. The ability to reason about protocol parts under assumptions about the way they will be used offers greater flexibility and appears essential for developing modular proofs about certain classes of protocols.

Finally, we note that although there are some similarities between the composition paradigm of PCL and the assume-guarantee paradigm in distributed computing [58], there is also one important difference. In PCL, while composing protocols, we check that each protocol respects the invariants of the other. This step involves an induction argument over the steps of the two protocols. There is no reasoning about attacker actions. One way to see the similarity with assume-guarantee is that each protocol is proved secure assuming some property of the other protocol and then discharging this assumption. The difference lies in the fact that the assumption made does not depend on the attacker although the environment for each protocol includes the attacker in addition to the other protocol.

9 Conclusions

Proving security properties of network protocols is a hard problem. One source of difficulty is concurrency—security properties have to be guaranteed in an environment where many sessions of multiple protocols simultaneously execute and the attacker can use information acquired from one session to defeat the security goals of another. Existing methods based on model-checking are useful for finding bugs, but do not guarantee protocol security for an unbounded number of sessions. On the other hand, explicit reasoning about traces containing honest principals’ and attacker’s actions using theorem-proving approaches require considerable effort and expertise. We have therefore developed PCL—a logic for proving security properties of protocols. The proof system for PCL codifies high-level reasoning principles for security protocols and thereby allows succinct proofs of practical protocols (2–3 pages). PCL supports compositional reasoning about security protocols and has been applied to a number of industry standards including SSL/TLS, IEEE 802.11i, Kerberos V5, and GDOI, in several cases identifying serious security vulnerabilities. While the logic was originally developed for the symbolic “Dolev-Yao” model of protocol execution and attack, a variant of the logic with similar reasoning principles has also been developed for the computational model used by cryptographers. We believe that this logic will prove useful in analyzing other protocols of practical import as well as in the education of students on topics related to security protocols and their design and analysis. One significant direction for future work is to develop useful tool support for the logic.

Acknowledgements

We would like to thank our collaborators on this project. We thank Nancy Durgin and Dusko Pavlovic for contributing towards the formulation of PCL, and Michael Backes, Changhua He, Jean-Pierre Seifert, Mukund Sundarajan and Mathieu Turuani for collaborations on case studies. The variant of PCL developed for the computational model is joint work with Vitaly Shmatikov, Mathieu Turuani, and Bogdan Warinschi.

References

[1] Abadi, M. and C. Fournet, *Mobile values, new names, and secure communication*, in: *28th ACM Symposium on Principles of Programming Languages*, 2001, pp. 104–115.

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

[3] Armando, A., D. A. Basin, Y. Boichut, Y. Chevalier, L. Compagna, J. Cuéllar, P. H. Drielsma, P.-C. Héam, O. Kouchnarenko, J. Mantovani, S. Mödersheim, D. von Oheimb, M. Rusinowitch, J. Santiago, M. Turuani, L. Viganò and L. Vigneron, *The AVISPA tool for the automated validation of internet security protocols and applications.*, in: *Computer Aided Verification, 17th International Conference, CAV 2005, Proceedings*, Lecture Notes in Computer Science **3576** (2005), pp. 281–285.

[4] Auty, M. and G. Lowe, *A calculus for security protocol development*, Submitted for publication (2006). URL http://web.comlab.ox.ac.uk/oucl/work/gavin.lowe/Papers/synthesis.pdf

[5] Backes, M., A. Datta, A. Derek, J. C. Mitchell and M. Turuani, *Compositional analysis of contract signing protocols*, in: *Proceedings of 18th IEEE Computer Security Foundations Workshop* (2005).

[6] Beauxis, R. and C. Palamidessi, *On the asynchronous nature of the asynchronous π-calculus* (2006), manuscript.

[7] Bella, G. and L. C. Paulson, *Kerberos version IV: Inductive analysis of the secrecy goals*, in: J.-J. Quisquater, editor, *Proceedings of the 5th European Symposium on Research in Computer Security* (1998), pp. 361–375.

[8] Bellare, M. and P. Rogaway, *Entity authentication and key distribution*, in: *Advances in Cryrptology - Crypto '93 Proceedings*, pp. 232–249.

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

[10] Blanchet, B., *An Efficient Cryptographic Protocol Verifier Based on Prolog Rules*, in: *14th IEEE Computer Security Foundations Workshop (CSFW-14)* (2001), pp. 82–96.

[11] Borisov, N., I. Goldberg and D. Wagner, *Intercepting mobile communications: the insecurity of 802.11*, in: *Proceedings of the 7th Annual International Conference on Mobile Computing and Networking*, 2001, pp. 180–189.

[12] Boyd, C. and A. Mathuria, "Protocols for Authentication and Key Establishment," Springer-Verlag, 2003.

[13] Burrows, M., M. Abadi and R. Needham, *A logic of authentication*, ACM Transactions on Computer Systems **8** (1990), pp. 18–36.

[14] Butler, F., I. Cervesato, A. Jaggard, A. Scedrov and C. Walstad, *Formal analysis of Kerberos 5* (2006), Theoretical Computer Science, in press.

[15] Butler, F., I. Cervesato, A. D. Jaggard and A. Scedrov, *Verifying confidentiality and authentication in kerberos 5.*, in: *Software Security - Theories and Systems, Second Mext-NSF-JSPS International Symposium, ISSS 2003*, Lecture Notes in Computer Science **3233** (2003), pp. 1–24.

[16] Canetti, R., *Universally composable security: A new paradigm for cryptographic protocols*, in: *Proc. 42nd IEEE Symp. on the Foundations of Computer Science*, IEEE, 2001, full version available at `http://eprint.iacr.org/2000/067/`.

[17] Canetti, R., 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* (2003), pp. 339–355.

[18] Cervesato, I., A. Jaggard, A. Scedrov, J.-K. Tsay and C. Walstad, *Breaking and fixing public-key kerberos*, Technical report.
URL `ftp://ftp.cis.upenn.edu/pub/papers/scedrov/pkinit.pdf`

[19] Cervesato, I., A. Jaggard, A. Scedrov, J.-K. Tsay and C. Walstad, *Breaking and fixing public-key Kerberos*, in: *Proc. 11-th Asian Computing Science Conference (ASIAN'06), Springer LNCS, to appear.*, 2006, preliminary report on `http://eprint.iacr.org/2006/009`.

[20] Datta, A., 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* (2003), pp. 109–125.

[21] Datta, A., A. Derek, J. C. Mitchell and D. Pavlovic, *Secure protocol composition (extended abstract)*, in: *Proceedings of ACM Workshop on Formal Methods in Security Engineering*, 2003, pp. 11–23.

[22] Datta, A., A. Derek, J. C. Mitchell and D. Pavlovic, *Abstraction and refinement in protocol derivation*, in: *Proceedings of 17th IEEE Computer Security Foundations Workshop* (2004), pp. 30–45.

[23] Datta, A., A. Derek, J. C. Mitchell and D. Pavlovic, *Secure protocol composition*, in: *Proceedings of 19th Annual Conference on Mathematical Foundations of Programming Semantics* (2004).

[24] Datta, A., A. Derek, J. C. Mitchell and D. Pavlovic, *A derivation system and compositional logic for security protocols*, Journal of Computer Security **13** (2005), pp. 423–482.

[25] Datta, A., A. Derek, J. C. Mitchell, V. Shmatikov and M. Turuani, *Probabilistic polynomial-time semantics for a protocol security logic.*, in: *Proceedings of the 32nd International Colloquium on Automata, Languages and Programming (ICALP '05)*, Lecture Notes in Computer Science (2005), pp. 16–29.

[26] Datta, A., A. Derek, J. C. Mitchell and B. Warinschi, *Computationally sound compositional logic for key exchange protocols*, in: *Proceedings of 19th IEEE Computer Security Foundations Workshop* (2006), pp. 321–334.

[27] Diffie, W., P. C. V. Oorschot and M. J. Wiener, *Authentication and authenticated key exchanges*, Designs, Codes and Cryptography **2** (1992), pp. 107–125.

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

[29] Durgin, N., J. C. Mitchell and D. Pavlovic, *A compositional logic for protocol correctness*, in: *Proceedings of 14th IEEE Computer Security Foundations Workshop* (2001), pp. 241–255.

[30] Durgin, N., J. C. Mitchell and D. Pavlovic, *A compositional logic for proving security properties of protocols*, Journal of Computer Security **11** (2003), pp. 677–721.

[31] Fábrega, F. J. T., 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* (1998), pp. 160–171.

[32] Gong, L., R. Needham and R. Yahalom, *Reasoning About Belief in Cryptographic Protocols*, in: D. Cooper and T. Lunt, editors, *Proceedings 1990 IEEE Symposium on Research in Security and Privacy* (1990), pp. 234–248.

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

[34] Guttman, J. D. and F. J. T. Fábrega, *Protocol independence through disjoint encryption*, in: *Proceedings of 13th IEEE Computer Security Foundations Workshop* (2000), pp. 24–34.

[35] Harel, D., D. Kozen and J. Tiuryn, "Dynamic Logic," Foundations of Computing, MIT Press, 2000.

[36] Harkins, D. and D. Carrel, *The Internet Key Exchange (IKE)* (1998), rFC 2409.

[37] He, C. and J. C. Mitchell, *Security analysis and improvements for IEEE 802.11i.*, in: *Proceedings of the Network and Distributed System Security Symposium, NDSS 2005* (2005).

[38] He, C., M. Sundararajan, A. Datta, A. Derek and J. C. Mitchell, *A modular correctness proof of IEEE 802.11i and TLS*, in: *CCS '05: Proceedings of the 12th ACM conference on Computer and communications security*, 2005, pp. 2–15.

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

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

[41] Hoare, C. A. R., *Communicating sequential processes*, Commun. ACM **21** (1978), pp. 666–677.

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

[43] Kohl, J. and B. Neuman, *The Kerberos network authentication service (version 5)*, IETF RFC 1510 (1993).

[44] Lincoln, P. D., J. C. Mitchell, M. Mitchell and A. Scedrov, *Probabilistic polynomial-time equivalence and security protocols*, in: *Formal Methods World Congress, vol. I*, number 1708 in Lecture Notes in Computer Science (1999), pp. 776–793.

[45] Lowe, G., *Some new attacks upon security protocols*, in: *Proceedings of 9th IEEE Computer Security Foundations Workshop* (1996), pp. 162–169.

[46] Lynch, N., *I/O automata models and proofs for shared-key communication systems*, in: *Proceedings of 12th IEEE Computer Security Foundations Workshop* (1999), pp. 14–29.

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

[48] Mantel, H., *On the Composition of Secure Systems*, in: *Proceedings of the IEEE Symposium on Security and Privacy* (2002), pp. 88–101.

[49] McCullough, D., *Noninterference and the composability of security properties*, in: *Proceedings of the IEEE Symposium on Security and Privacy* (1988), pp. 177–186.

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

[51] McLean, J., *Security models and information flow*, in: *Proceedings of the IEEE Symposium on Security and Privacy* (1990).

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

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

[54] Meadows, C., *Analysis of the Internet Key Exchange protocol using the NRL protocol analyzer*, in: *Proceedings of the IEEE Symposium on Security and Privacy* (1998).

[55] Meadows, C., *Open issues in formal methods for cryptographic protocol analysis*, in: *Proceedings of DISCEX 2000* (2000), pp. 237–250.

[56] Meadows, C. and D. Pavlovic, *Deriving, attacking and defending the GDOI protocol.*, in: *Computer Security - ESORICS 2004, 9th European Symposium on Research Computer Security, Proceedings*, Lecture Notes in Computer Science **3193** (2004), pp. 53–72.

[57] Milner, R., "A Calculus of Communicating Systems," Springer-Verlag, 1982.

[58] Misra, J. and K. M. Chandy, *Proofs of networks of processes*, IEEE Transactions on Software Engineering **7** (1981), pp. 417–426.

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

[60] Mitchell, J. C., V. Shmatikov and U. Stern, *Finite-state analysis of ssl 3.0*, in: *Proceedings of the Seventh USENIX Security Symposium*, 1998, pp. 201–216.

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

[62] Nilsson, N. J., *Probabilistic logic*, Artificial Intelligence **28** (1986), pp. 71–87.

[63] Paulson, L., *Mechanized proofs for a recursive authentication protocol*, in: *Proceedings of 10th IEEE Computer Security Foundations Workshop*, 1997, pp. 84–95.

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

[65] Ramanathan, A., J. C. Mitchell, A. Scedrov and V. Teague, *Probabilistic bisimulation and equivalence for security analysis of network protocols*, in: *Foundations of Software Science and Computation Structures, 7th International Conference, FOSSACS 2004, Proceedings*, Lecture Notes in Computer Science **2987** (2004), pp. 468–483.

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

[67] Roy, A., A. Datta, A. Derek and J. C. Mitchell, *Inductive proof method for computational secrecy* (2006), manuscript.

[68] Roy, A., A. Datta, A. Derek, J. C. Mitchell and J.-P. Seifert, *Secrecy analysis in protocol composition logic.* (2006), to appear in Proceedings of 11th Annual Asian Computing Science Conference, December 2006.

[69] Ryan, P., S. Schneider, M. Goldsmith, G. Lowe and B. Roscoe, "Modelling and Analysis of Security Protocols," Addison-Wesley, 2001.

[70] Schneider, S., *Security properties and CSP*, in: *IEEE Symp. Security and Privacy*, 1996.

[71] Schneider, S., *Verifying authentication protocols with csp*, IEEE Transactions on Software Engineering (1998), pp. 741–58.

[72] Song, D., *Athena: a new efficient automatic checker for security protocol analysis*, in: *Proceedings of 12th IEEE Computer Security Foundations Workshop* (1999), pp. 192–202.

[73] Syverson, P. and P. van Oorschot, *On unifying some cryptographic protocol logics*, in: *Proceedings of 7th IEEE Computer Security Foundations Workshop*, 1994, pp. 14–29.

[74] Thayer, F. J., J. C. Herzog and J. D. Guttman, *Mixed strand spaces*, in: *Proceedings of 12th IEEE Computer Security Foundations Workshop* (1999), pp. 72–82.

[75] *"Verified by Visa" security program*, Internet Resource.
URL https://usa.visa.com/personal/security/vbv/

[76] Wagner, D. and B. Schneier, *Analysis of the ssl 3.0 protocol*, in: *Proceedings of the 2nd USENIX Workshop on Electronic Commerce*, 1996.

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

A Extending the Logic with Diffie-Hellman Primitive

In order to keep the description of the core PCL simple, we introduce Diffie-Hellman primitive as well as the associated proof rules and axioms as extension to the logic. Our treatment of Diffie-Hellman primitive in this symbolic model is straight forward. Exponentials such as $g^a \ mod \ p$ and shared secret $g^{ab} \ mod \ p$ will be represented by special terms $g(a)$ and $h(a, b)$ respectively. Similarly to the black-box model of encryption and signature, we will assume that the only way to compute these terms is via specified symbolic actions. Therefore, abstract away the number-theoretic properties of Diffie-Hellman key exchange scheme.

DH1 $\mathsf{Computes}(X, g^{ab}) \supset \mathsf{Has}(X, g^{ab})$

DH2 $\mathsf{Has}(X, g^{ab}) \supset$
$(\mathsf{Computes}(X, g^{ab}) \vee \exists m.(\mathsf{Receive}(X, m) \wedge \mathsf{Contains}(m, g^{ab})))$

DH3 $(\mathsf{Receive}(X, m) \wedge \mathsf{Contains}(m, g^{ab})) \supset$
$\exists Y, m'.(\mathsf{Computes}(Y, g^{ab}) \wedge \mathsf{Send}(Y, m') \wedge \mathsf{Contains}(m', g^{ab}))$

DH4 $\mathsf{Fresh}(X, a) \supset \mathsf{Fresh}(X, g^a)$

$$\mathsf{Computes}(X, g^{ab}) \equiv (\ (\mathsf{Has}(X, a) \wedge \mathsf{Has}(X, g^b)) \vee (\mathsf{Has}(X, b) \wedge \mathsf{Has}(X, g^a))\ )$$

Table A.1
Diffie-Hellman Axioms

Programming Language and the Execution Model

Set of terms of PCL (see Table 1 in Section 2) is extended with constructs $g(n)$ and $h(n, n)$, where n is a nonce. Informally, $g(a)$ and $h(a, b)$ will stand for $g^a \ mod\ p$ and $g^{ab} \ mod\ p$ respectively. To improve readability will often use g^a and g^{ab} instead of $g(a)$ and $h(a, b)$.

Set of actions of PCL (see Table 1 in Section 2) is extended with constructs $x :=$ `expg` n and $x :=$ `dhkeyken` t, n modelling creation of the exponential g^a given a nonce a and the creation of the shared secret g^{ab} given an exponential g^b and a nonce a. Operational semantics of these two actions is defined in a straight forward manner, terms $g(n)$ and $h(a, b)$ respectively are substituted for the variable x.

Protocol Logic

We do not introduce additional formulas to the logic, we do need, however to redefine semantics of a few predicates. Semantics of predicate Fresh is extended so that $\mathsf{Fresh}(X, g^x)$ is true if and only if $\mathsf{Fresh}(X, x)$ is true. Semantics of predicate Gen is redefined in a similar fashion. Semantics of predicate Has is redefined to model the Diffie-Hellman property $(g^a)^b = (g^b)^a$, formally if $\mathsf{Has}(X, a)$ and $\mathsf{Has}(X, g^b)$ are true then $\mathsf{Has}(g^{ab})$ and $\mathsf{Has}(X, g^{ba}$ are both true.

Proof System

Table A.1 presents the rules specific to the way that Diffie-Hellman secrets are computed. The predicate Computes() is used as a shorthand to denote the fact that the only way to compute a Diffie-Hellman secret is to possess one exponent and the other exponential. Axiom **DH1** states that if X can compute the Diffie-Hellman secret, then she also possesses it. Axiom **DH2** captures the intuition that the only way to possess a Diffie-Hellman secret is to either compute it directly or obtain it from a received message containing it. Axiom **DH3** states that if a principal receives a message containing a Diffie-Hellman secret, someone who has computed the secret must have previously sent a (possibly different) message containing it. Axiom **DH4** captures the intuition that if a is fresh at some point of a run, then g^a is also fresh at that point.

B Soundness of Axioms and Proof Rules

In this section we prove the soundness of the axioms and proof rules used in the proof system, hence proving Theorem 4.1. We omit proofs for standard axioms and rules of temporal logic.

B.1 Axioms for protocol actions

AA1 $\top[a]_X \; \mathsf{a}$

Informally, this axiom says that if a is an action, and a the corresponding action predicate, when thread X executes a, in the resulting state a holds. Let $\mathcal{Q}$ be a protocol, and let $R = R_0R_1R_2$ be a run such that $R_1|_X$ matches a under substitution σ and $\mathcal{Q}, R_0 \models \sigma\phi$, we need to prove that $\mathcal{Q}, R_0R_1 \models \sigma\mathsf{a}$. Since $R_1|_X$ matches a under substitution σ, R_1 has to contain action $\sigma\mathsf{a}$, and therefore, by the semantics of the action predicates it has to be that $\mathcal{Q}, R_0R_1 \models \mathsf{a}$. Now, by the definition of modal formulas we have $Q \models \top[a]_X\mathsf{a}$.

AA2 $\mathsf{Start}(X)[\,]_X \; \neg\mathsf{a}(\mathsf{X})$

AA3 $\neg\mathsf{Send}(X,t)[b]_X\neg\mathsf{Send}(X,t)$ *if* $\sigma\mathsf{Send}(X,t) \neq \sigma\mathsf{b}$ *for all substitutions* σ

AA4 $\top[a;\cdots;b]_X\mathsf{a} < \mathsf{b}$

Axiom **AA2** simply says that no action predicate can hold if thread X executed no actions. Axiom **AA3** says that $\neg\mathsf{Send}(X,t)$ is preserved as long as no send actions is performed that unifies with the term t. Soundness of these two axioms trivially follows from the semantics of action predicates and predicate Start. Soundness of axiom **AA4** directly follows from the semantics of modal formula and the temporal ordering operator.

AN1 $\mathsf{New}(X,x) \wedge \mathsf{New}(Y,x) \supset X = Y$

AN2 $\phi[(\nu n)]_X \; \mathsf{Has}(Y,n) \supset (Y = X)$

AN3 $\phi[(\nu n)]_X \; \mathsf{Fresh}(X,n)$

AN4 $\mathsf{Fresh}(X,x) \supset \mathsf{Gen}(X,x)$

Informally, axioms **AN1** and **AN2** say that fresh nonces are unique and initially secret to the originating thread. If a process X generates a new value m and takes no further actions, then X is the only thread who knows m. The soundness of this axiom follows from the definition of the execution model and the semantics of the predicate "Has". For a detailed proof see [30]. Axiom **AN3** states that the newly created value is fresh exactly after creation. The soundness of this axiom follows directly from the semantics of the predicate Fresh. Axiom **AN4** is trivially sound by the semantics of predicate Gen.

B.2 Possession axioms

PROJ $\mathsf{Has}(X,(x,y)) \supset \mathsf{Has}(X,x) \wedge \mathsf{Has}(X,y)$
TUP $\mathsf{Has}(X,x) \wedge \mathsf{Has}(X,y) \supset \mathsf{Has}(X,(x,y))$
ENC $\mathsf{Has}(X,x) \wedge \mathsf{Has}(X,K) \supset \mathsf{Has}(X, ENC_K\{|x|\})$
DEC $\mathsf{Has}(X, ENC_K\{|x|\}) \wedge \mathsf{Has}(X,K) \supset \mathsf{Has}(X,x)$
This set of axioms describes ways in which a thread can accumulate knowledge. Informally, these axioms say that if a thread has all the necessary parts to build some term then he has the term itself. Also, a thread can decompose tuples and decrypt messages encrypted with a known key. Soundness of these axioms follows directly from the semantics of the predicate "Has". Here, we prove the soundness of axiom **ENC**, proofs for other axioms are similar.

When $Q,R \not\models \mathsf{Has}(X,x) \wedge \mathsf{Has}(X,K)$ then $Q,R \models$ **ENC** holds trivially. Otherwise, by the semantics of "$\wedge$", $Q,R \models \mathsf{Has}(X,x)$ and $Q,R \models \mathsf{Has}(X,K)$ both hold. That means, that $\mathsf{Has}_i(X,x)$ and $\mathsf{Has}_j(X,K)$ for some i and j. Assuming $i \geq j$, we have $\mathsf{Has}_i(X,K)$ and therefore $\mathsf{Has}_{i+1}(X, ENC_K\{|x|\})$.

ORIG $\mathsf{New}(X,n) \supset \mathsf{Has}(X,n)$
REC $\mathsf{Receive}(X,x) \supset \mathsf{Has}(X,x)$
Informally, these axioms make connection between knowledge of a thread and the actions executed by that thread in the past. A thread has all terms it creates or receives. Soundness of these axioms follows directly from the semantics of the predicate "Has".

AR1 $\mathsf{a}(x)[\texttt{match}\ q(x)/q(t)]_X\ \mathsf{a}(t)$
AR2 $\mathsf{a}(x)[\texttt{verify}\ x,t,K]_X\ \mathsf{a}(SIG_K\{|t|\})$
AR3 $\mathsf{a}(x)[y := \mathtt{dec}\ x,K]_X\ \mathsf{a}(ENC_K\{|y|\})$
Axioms **AR1**, **AR2** and **AR2** are used to model obtaining information about structure of terms as they are being parsed. We prove soundness of axiom **AR1**, proofs for other two axioms are similar. Let $\mathcal{Q}$ be a protocol, and let $R = R_0R_1R_2$ be a run such that $R_1|_X$ matches $(q(x)/q(t))$ under substitution σ and $\mathcal{Q}, R_0 \models \sigma\mathsf{a}(x)$, we need to prove that $\mathcal{Q}, R_0R_1 \models \sigma\,\mathsf{a}(t)$. Since $R_1|_X$ matches $(q(x)/q(t))$ under substitution σ, and events of R_1 only contain ground terms, it has to be that σx is same as σt, and therefore $\mathcal{Q}, R_0 \models \mathsf{a}(t)$. Clearly, formulas of the form $\mathsf{a}(t)$ remain valid as new actions are executed, hence $\mathcal{Q}, R_0R_1 \models \sigma\mathsf{a}(t)$.

B.3 Encryption and signature

SEC $\mathsf{Honest}(\hat{X}) \wedge\ \mathsf{Decrypt}(Y, ENC_X\{|n|\}) \supset (\hat{Y} = \hat{X})$
Informally, **SEC** says that if an agent $\hat{X}$ is honest, and some thread Y executed by principal $\hat{Y}$ has decrypted a message $ENC_X\{|n|\}$ (i.e. a message encrypted with $\hat{X}$'s public key), then $\hat{Y}$ must be $\hat{X}$. In other words, if $\hat{X}$ is honest, then only threads executed by $\hat{X}$ can decrypt messages encrypted $\hat{X}$'s private key. For a detailed soundness proof of this axiom see [30].

B.4 Preservation axioms

P1 $\mathsf{Persist}(X,t)[a]_X\mathsf{Persist}(X,t)$ *where* $\mathsf{Persist} \in \{\mathsf{Has}, \mathsf{FirstSend}, \mathsf{a}, \mathsf{Gen}\}$
Informally this axiom says that the for some formulas stay valid when a thread does additional actions. Since the semantics of the predicate "Has" is based on the existence of a certain event in a run, adding additional events to the run cannot make this predicates false. Also, action predicates, predicates FirstSend and Gen are trivially preserved when additional actions are added to the run.

P2 $\mathsf{Fresh}(X,t)[a]_X\mathsf{Fresh}(X,t)$ *where* $t \not\subseteq a$
Informally this axiom says that a nonce n remains fresh as long as it is not explicitly used as a parameter in any action send out as a part of some message m. The soundness of this axiom follows from the semantics of the predicate Fresh.

B.5 Temporal ordering of actions

FS1 $\mathsf{Fresh}(X,t)[\mathsf{send}\ t']_X\mathsf{FirstSend}(X,t,t')$ *where* $t \subseteq t'$
FS2 $\mathsf{FirstSend}(X,t,t') \wedge \mathsf{a}(Y,t'') \supset \mathsf{Send}(X,t') < \mathsf{a}(Y,t'')$ *where* $X \neq Y$ *and* $t \subseteq t''$
Axiom FS1 says that the FirstSend(X, t, t') predicate holds if a thread X sends the term t' containing t starting from a state where t is fresh. Soundness of this axiom follows directly from the semantics of predicates Fresh and FirstSend. Axiom **FS2** says that the all actions a involving the term t which was fresh at some point, must have happened after the first time that t was send out. The soundness of this axioms follows from the semantics of the predicate FirstSend, semantics of temporal operator and Lemmas 2.1 and 2.3.

B.6 Axioms for Diffie-Hellman key exchange

$$\mathsf{Computes}(X, g^{ab}) \equiv ((\mathsf{Has}(X,a) \wedge \mathsf{Has}(X,g^b)) \vee (\mathsf{Has}(X,b) \wedge \mathsf{Has}(X,g^a))\ )$$

DH1 $\mathsf{Computes}(X,g^{ab}) \supset \mathsf{Has}(X,g^{ab})$
Informally, this axiom says that if some thread has all necessary information to compute the Diffie-Hellman secret, then he also has the Diffie-Hellman secret itself. The soundness of this axiom follows directly from the semantics of the predicate "Has".

DH2 $\mathsf{Has}(X,g^{ab}) \supset (\mathsf{Computes}(X,g^{ab}) \vee \exists m.(\ \mathsf{Receive}(X,m) \wedge \mathsf{Contains}(m,g^{ab})))$
Informally, this axiom says that the only way to have a Diffie-Hellman secret is to compute it from one exponent and one exponential or receive it as a part of some message. To prove the axiom we have to check all the cases in the semantics of the predicate "Has".

DH3 $(\ \mathsf{Receive}(X, m) \wedge \mathsf{Contains}(m, g^{ab})) \supset$
$\exists Y, m'.(\mathsf{Computes}(Y, g^{ab}) \wedge\ \mathsf{Send}(Y, m') \wedge \mathsf{Contains}(m', g^{ab}))$

Informally, this axiom says that if someone receives a Diffie-Hellman shared secret then there must be some thread that send it and computed it himself. Let R be a run in which X receives a message m containing g^{ab} at some point. By Lemma 2.3, that means that in the run R there exists someone who send a message m containing g^{ab}. Let R' be a shortest prefix of R in which some agent Y sends some message m' containing g^{ab} at some point. Since R' is a shortest such prefix, that means that Y could not receive a message m'' containing g^{ab}. By axiom **DH2** that means that Y must have computed g^{ab} himself.

DH4 $\mathsf{Fresh}(X, a) \supset \mathsf{Fresh}(X, g^a)$

Informally, this axiom states that a Diffie-Hellman exponential is fresh as long as the exponent is fresh. The soundness of this axiom follows directly from the semantics of the predicate "Fresh".

B.7 Generic rules

G1 follows from the semantics of "$\wedge$" and "$\theta[P]_X\phi$". Let $R = R_0R_1R_2$. If R_1 does not match $P|_X$ or $Q, R_0 \not\models \theta$ then trivially $Q, R \models \theta[P]_X\phi \wedge \psi$. Otherwise, it has to be that $Q, R_0R_1 \models \phi$ and $Q, R_0R_1 \models \psi$, and $Q, R \models \theta[P]_X\phi \wedge \psi$ follows from the semantics of "$\wedge$". Validity of axioms **G2** and **G3** can be verified similarly. Axiom **G4** is trivially valid because if ϕ is true after any run, then ϕ is true after a specific run that contains actions P.