FORMAL ANALYSIS OF SECURITY PROTOCOLS: PROTOCOL COMPOSITION LOGIC

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Ante Đerek
December 2006

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

(John C. Mitchell) Principal Adviser

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

(Dan Boneh)

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

(David L. Dill)

Approved for the University Committee on Graduate Studies.

Abstract

We develop Protocol Composition Logic (PCL) – a Floyd-Hoare style logic for axiomatic proofs of protocol properties that is sound with respect to the standard symbolic model of protocol execution and attack. PCL can express temporal ordering of actions and knowledge, naturally capturing security properties such as authentication and secrecy. The induction rule for proving invariants and the composition theorems allow us to prove properties of compound protocols by combining independent proofs of their components.

In order to bridge the gap between the symbolic and computational models of security, we develop Computational PCL (CLCP) – a cryptographically sound formal logic for proving protocol security properties without explicitly reasoning about probability, asymptotic complexity, or the actions of a malicious attacker. The approach rests on a new probabilistic, polynomial-time semantics for a subset of PCL. Using CPCL we formulate a specification of secure key exchange that is closed under general composition with steps that use the key and present sound formal proof rules based on this game-based condition.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Security protocols, such as authentication and key-exchange protocols, are difficult to design and debug. For example, the 802.11 Wired Equivalent Privacy (WEP) protocol, used to protect wireless communications from eavesdropping and other attacks, has several serious security flaws [19]. Anomalies and shortcomings have also been discovered in standards and proposed standards for Secure Sockets Layer [99, 80], the later 802.11i wireless authentication protocols [56, 57], Kerberos [63, 13, 29, 22], and others. Although many of these protocols may seem relatively simple, in comparison with more complex distributed systems, security protocols must achieve certain goals when an arbitrary number of 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 basis of modern secure networked systems, it is important to develop informative, accurate and usable 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 [79, 80, 88], 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.

Security analysis of network protocols is a successful scientific area with two important but historically independent foundations, one based on logic and symbolic computation, and one based on computational complexity theory. The symbolic

approach, which uses a highly idealized representation of cryptographic primitives, has been a successful basis for formal logics and automated tools. Conversely, the computational approach yields more insight into the strength and vulnerabilities of protocols, but it is more difficult to apply and it involves explicit reasoning about probability and computational complexity.

In the first part of this dissertation, we describe a specific logic (PCL), developed for the purpose of proving security properties of network protocols in the symbolic model, and give some examples of its use. The purpose of the second part of this dissertation is to suggest that formal reasoning can be used to reason about probabilistic polynomial-time protocols in the face of probabilistic polynomial-time attacks. We achieve this goal by proposing a computational semantics for subset of PCL. The resulting logic (CPCL) retains desirable aspects of PCL, while being faithful to the complexity-theoretic model of protocol execution and attack. While the expressivity of the resulting logic and the strength of the corresponding proof system are still not comparable to those of PCL, we are able to develop methods to model and analyze key exchange protocols using CPCL.

1.1 Protocol Composition Logic

Protocol Composition Logic (PCL) [32, 33, 34, 35, 36, 37, 38, 43, 44, 57, 89, 90] 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 [59] and dynamic logic [55], with the formula $\theta[P]_X\phi$ 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 components that X either generated (using a `new` action) or received in a way that did not hide them under encryption by a key not known 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 derived 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.

In comparison with a previous generation of protocol logics such as BAN logic [21], 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 [85] and Schneider's rank function method [93].

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.

The core of PCL was formulated in [44, 36]. Subsequent work on proof methods for PCL [32, 33, 34, 35, 90], as well as case studies using PCL [7, 57, 90] led to extensions and modifications to the syntax, semantics and proof system. This dissertation develops the basic concepts in a uniform notation and semantic setting and improves on the previous technical definitions and proofs. We focus on presenting the core of PCL rather than the case studies. Simple signature based authentication protocol and ISO-9798-3 key exchange protocols are used as the running examples.

1.2 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 with respect to the complexity-theoretic model of modern cryptography [37, 38, 89]. Computational PCL inherits its syntax and reasoning methods from PCL. However, the semantics of the logic is defined with respect to a probabilistic polynomial time model of protocol execution and attack.

Our central organizing idea 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 [83]. Since the logic does not mention probability explicitly, we consider a formula to be "true" if it holds with asymptotically overwhelming probability.

In the symbolic semantic model, the atomic formula $\mathsf{Has}(X, m)$ means that m is

in the set of values "derivable," by a simple fixed algorithm, from information visible to X. The simple fixed algorithm is central to the Dolev-Yao model. In replacing the symbolic semantics with a computational semantics based on probabilistic polynomial time, we replace the predicate Has with two predicates, Possess and Indist. Intuitively, $\mathsf{Possess}(X, m)$ means that there is an algorithm that computes the value of m with high probability from information available to X, while $\mathsf{Indist}(X, m)$ means that X cannot feasibly distinguish m from a random value chosen according to the same distribution. However, certain technical problems discussed in Section 3.6 lead us to work with slightly simplified semantics of these predicates that capture our intuition most strongly when the possessing principal is assumed honest (in the sense of following the protocol) and the predicate Indist only appears with positive polarity. Fortunately, these syntactic conditions are met in many formulas expressing authentication and secrecy properties.

Several groups of researchers have either formulated connections between symbolic logic and feasible probabilistic computation, or developed relationships between symbolic and computational models. In particular, Abadi and Rogaway [4] propose a logical characterization of indistinguishability by passive eavesdroppers that has been studied by a number of others, and Kapron and Impagliazzo suggest a formal logic for reasoning about probabilistic polynomial-time indistinguishability [61]. Some semantic connections between symbolic and computational models have been developed by a team at IBM Zurich, *e.g.,* [10], with other connections explored in a series of related papers by Micciancio, Warinschi and collaborators [77, 100, 31].

This dissertation develops the basic semantics of CPCL and presents a simple proof system that can be used as a basis for logics aimed at analyzing industrial level security protocols in the computational model. Also, we use CPCL to develop a compositional method for proving cryptographically sound security properties of key exchange protocols.

1.3 Organization

The rest of the dissertation is organized as follows. Protocol composition logic is developed in Chapter 2 and immediately used to prove authentication properties of example protocols. We also prove composition theorems and use them to prove security properties of compound protocols from properties of their components. Computational PCL is developed in Chapter 3 and used in Chapter 4 to analyze key exchange protocols. Chapter 5 summarizes other results associated with PCL, which are not presented in this dissertation. Related work is discussed in Chapter 6. Finally, Chapter 7 concludes the dissertation.

Acknowledgments

I would like to thank my advisor, John Mitchell, for providing valuable guidance in my research. I thank Nancy Durgin and Dusko Pavlovic for contributing towards the formulation of PCL, and Michael Backes, Changhua He, 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. Finally, I would like to thank Anupam Datta who has collaborated with me on almost every aspect of this dissertation.

Chapter 2

Protocol Composition Logic

Section 2.1 describes the syntax and operational semantics for the protocol programming language. Section 2.2 presents the syntax and semantics for PCL, with the proof system and soundness theorem in Section 2.3. An application of the formal system to an authentication protocol is presented in Section 2.4. Section 2.5 presents theorems about sequential and parallel composition of protocols and illustrates their use through application to a key exchange protocol.

2.1 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 [91, 20]) 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.

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

Figure 2.1: Challenge-response protocol as arrows-and-messages

Now, we show how protocols are represented with an example. Figure 2.1 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 2.2, 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 2.2. 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.

2.1.1 Protocol Programming Language

Our protocol programming language is based on standard process calculus. Originally introduced in [43, 44] under the name *cord calculus*, it was inspired by the strand space

$$
\begin{array}{lcl@{\qquad}lcl}
\mathbf{Init_{CR}} & \equiv & (\hat{Y})[& \mathbf{Resp_{CR}} & \equiv & ()[\\
 & & \texttt{new}\ m; & & & \texttt{receive}\ \hat{X}, \hat{Y}, x; \\
 & & \texttt{send}\ \hat{X}, \hat{Y}, m; & & & \texttt{new}\ n; \\
 & & \texttt{receive}\ \hat{Y}, \hat{X}, y, s; & & & r := \texttt{sign}\ (n, x, \hat{X}), \hat{Y}; \\
 & & \texttt{verify}\ s, (y, m, \hat{X}), \hat{Y}; & & & \texttt{send}\ \hat{Y}, \hat{X}, n, r; \\
 & & r := \texttt{sign}\ (y, m, \hat{Y}), \hat{X}; & & & \texttt{receive}\ \hat{X}, \hat{Y}, t; \\
 & & \texttt{send}\ \hat{X}, \hat{Y}, r; & & & \texttt{verify}\ t, (n, x, \hat{Y}), \hat{X}; \\
]_X() & & &]_Y() & &
\end{array}
$$

Figure 2.2: Roles of the Challenge-response protocol

formalism [47], 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 extended strand spaces by an operational semantics in the style of chemical abstract machine [18]. To represent the stores where the messages are to be received, we added variables, and a substitution mechanism expressed by simple reaction rules, corresponding to the basic communication and computation operations. The resulting process calculus of cords can thus be construed as a rudimentary protocol execution model, based on strand spaces and 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.

(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 2.1: Syntax of the Protocol Programming Language - terms

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

Table 2.2: Syntax of the Protocol Programming Language - actions

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 2.1 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.

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

$$[\texttt{receive}\ x; S]_X \mid [\texttt{send}\ t; T]_Y \longrightarrow [S(t/x)]_X \mid [T]_Y \quad (2.1)$$
$$[\texttt{match}\ p(t)/p(x); S]_X \longrightarrow [S(t/x)]_X \quad (2.2)$$
$$[\texttt{new}\ x; S]_X \longrightarrow [S(m/x)]_X \quad (2.3)$$
$$[x := \texttt{enc}\ t, K; S]_X \longrightarrow [S(ENC_K\{\!|t|\!\}/x)]_X \quad (2.4)$$
$$[x := \texttt{dec}\ ENC_K\{\!|t|\!\}, K; S]_X \longrightarrow [S(t/x)]_X \quad (2.5)$$
$$[x := \texttt{sign}\ t, K; S]_X \longrightarrow [S(SIG_K\{\!|t|\!\}/x)]_X \quad (2.6)$$
$$[\texttt{verify}\ SIG_K\{\!|t|\!\}, t, K; S]_X \longrightarrow [S]_X \quad (2.7)$$

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 2.3: Basic reaction steps

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.

2.1.2 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 2.3.

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 (2.1) 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 (2.2) 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 (2.3) 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 uniquie in the entire cord space.

Reactions (2.4) and (2.5) 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 (2.6) and (2.7) 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 [82] and a model presented by Dolev and Yao [41].

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 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 [12], 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:

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

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*.

Protocol properties

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

Lemma 2.1.1. (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 2.1.2. (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 2.1.3. *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 2.1.4. *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. □

2.2 Protocol Logic

2.2.1 Syntax

The formulas of PCL are given by the grammar in Table 2.4, where S may be any strand. Here, t and P denote a term and a thread, respectively. We use ϕ and ψ to

Action formulas

$$\begin{array}{lll} \mathsf{a} & ::= & \mathsf{Send}(P,t) \mid \mathsf{Receive}(P,t) \mid \mathsf{New}(P,t) \mid \mathsf{Encrypt}(P,t) \mid \\ & & \mathsf{Decrypt}(P,t) \mid \mathsf{Sign}(P,t) \mid \mathsf{Verify}(P,t) \end{array}$$

Formulas

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

Modal formulas

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

Table 2.4: Syntax of the logic

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 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.

2.2.2 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. The 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:

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

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

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 $R_1|_A$.

2.3 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.

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

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

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

2.3.4 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.

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

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

2.3.5 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'.

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

2.3.6 Preservation Axioms

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

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

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

2.3.7 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 $\quad \mathsf{Fresh}(X, t)[\mathtt{send}\ t']_X \mathsf{FirstSend}(X, t, t')$

where $t \subseteq t'$.

FS2 $\quad \mathsf{FirstSend}(X, t, t') \wedge \mathsf{a}(Y, t'') \supset \mathsf{Send}(X, t') < \mathsf{a}(Y, t'')$

where $X \neq Y$ and $t \subseteq t''$.

2.3.8 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 [67] 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 [43, 44], 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.

$$\frac{\mathsf{Start}(X)[\,]_X\ \phi \qquad \forall \rho \in \mathcal{Q}.\forall P \in BS(\rho).\ \phi\ [P]_X\ \phi}{\mathsf{Honest}(\hat{X}) \supset \phi}\ \mathbf{HON_Q}$$

no free variable in ϕ except X bound in $[P]_X$

2.3.9 Soundness

The soundness theorem for this proof system is proved, by induction on the length of proofs, in Appendix A.2. 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 $\Gamma \vdash_{\mathcal{Q}} \gamma$ 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 2.4).

Theorem 2.3.1. *If* $\Gamma \vdash_{\mathcal{Q}} \gamma$, *then* $\Gamma \models_{\mathcal{Q}} \gamma$. *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 send 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 2.1.1 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 2.1.3 there is a corresponding send action for every receive, hence there exists a thread Z such that $EVENT(R, Z, \texttt{send}\ m, \emptyset, \emptyset)$ 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 $EVENT(R', Z, \texttt{send}\ m, \emptyset, \emptyset)$. By Lemma 2.1.1 $SIG_X\{\!|x|\!\}$ 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 2.1.2.

Sequencing rule Sequencing rule **S1** (see Section 2.3.5) 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 2.3.8) 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 2.1.4, 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**.

2.4 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 2.1. Our formulation of authentication is based on the concept of *matching conversations* [16] and is similar to the idea of proving authentication using *correspondence assertions* [101]. The same basic idea is also presented in [40] 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 2.1. 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

AA1	$\top[\texttt{verify}\ s,(y,m,\hat{X}),\hat{Y}]_X\mathsf{Verify}(X,SIG_{\hat{Y}}\{\!\|y,m,\hat{X}\|\!\})$	(2.8)
2.8, **P1**, **SEQ**	$\top[\mathbf{Init_{CR}}]_X\mathsf{Verify}(X,SIG_{\hat{Y}}\{\!\|y,m,\hat{X}\|\!\})$	(2.9)
2.9, **VER**	$\top[\mathbf{Init_{CR}}]_X\exists Y,t.\ \mathsf{Send}(Y,t)\wedge\mathsf{Contains}(t,SIG_{\hat{Y}}\{\!\|y,m,\hat{X}\|\!\})$	(2.10)
$\mathbf{HON_{Q_{CR}}}$	$(\mathsf{Honest}(\hat{Y})\wedge\mathsf{Send}(Y,t)\wedge\mathsf{Contains}(t,SIG_{\hat{Y}}\{\!\|y,m,\hat{X}\|\!\}))\supset$ $(\mathsf{New}(Y,m)\vee$ $(\mathsf{Receive}(Y,(\hat{X},\hat{Y},m))<\mathsf{Send}(Y,(\hat{Y},\hat{X},y,SIG_{\hat{Y}}\{\!\|y,m,\hat{X}\|\!\}))))$	(2.11)
2.10, 2.11	$\top[\mathbf{Init_{CR}}]_X\mathsf{Honest}(\hat{Y})\supset(\exists Y.\ \mathsf{New}(Y,m)\vee$ $(\mathsf{Receive}(Y,(\hat{X},\hat{Y},m))<\mathsf{Send}(Y,(\hat{Y},\hat{X},y,SIG_{\hat{Y}}\{\!\|y,m,\hat{X}\|\!\}))))$	(2.12)
AA1	$\top[\texttt{new}\ m]_X\mathsf{New}(X,y)$	(2.13)
2.13, **P1**, **SEQ**	$\top[\mathbf{Init_{CR}}]_X\mathsf{New}(X,y)$	(2.14)
2.12, 2.14, **AN1**	$\top[\mathbf{Init_{CR}}]_X\mathsf{Honest}(\hat{Y})\wedge\hat{Y}\neq\hat{X}\supset(\exists Y.$ $\mathsf{Receive}(Y,(\hat{X},\hat{Y},m))<\mathsf{Send}(Y,(\hat{Y},\hat{X},y,SIG_{\hat{Y}}\{\!\|y,m,\hat{X}\|\!\})))$	(2.15)

Table 2.5: Weak authentication for the initiator role of the CR protocol

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).

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

A formal proof of the weak authentication property for the initiator guaranteed by executing the CR protocol is presented in Table 2.5. 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:

- Lines (2.8)–(2.10) 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 Bobs 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 (2.11)–(2.12), 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 (2.13)–(2.15), we again reason 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:

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

Here, we are using msg_1, msg_2 and msg_3 as shortcuts for the corresponding messages in the protocol: $msg_1 \equiv (\hat{X}, \hat{Y}, m)$, $msg_2 \equiv (\hat{Y}, \hat{X}, y, SIG_{\hat{Y}}\{|y, m, \hat{X}|\})$, $msg_3 \equiv$

AN3	$\top[\texttt{new}\ m]_X \mathsf{Fresh}(X, m)$	(2.16)
FS1	$\mathsf{Fresh}(X, m)[\texttt{send}\ \hat{X}, \hat{Y}, m]_X$ $\mathsf{FirstSend}(X, m, (\hat{X}, \hat{Y}, m))$	(2.17)
2.16, 2.17, **SEQ**, **P1**	$\top[\mathbf{Init_{CR}}]_X \mathsf{FirstSend}(X, m, (\hat{X}, \hat{Y}, m))$	(2.18)
2.18, **FS2**	$\top[\mathbf{Init_{CR}}]_X \mathsf{Receive}(Y, (\hat{X}, \hat{Y}, m)) \wedge \hat{Y} \neq \hat{X} \supset$ $\mathsf{Send}(X, (\hat{X}, \hat{Y}, m)) < \mathsf{Receive}(Y, (\hat{X}, \hat{Y}, m))$	(2.19)
$\mathbf{HON_{Q_{CR}}}$	$(\mathsf{Honest}(\hat{Y}) \wedge \mathsf{Receive}(Y, (\hat{X}, \hat{Y}, m)) \wedge$ $\mathsf{Send}(Y, (\hat{Y}, \hat{X}, y, SIG_{\hat{Y}}\{\!|y, m, \hat{X}|\!\}))) \supset$ $\mathsf{FirstSend}(Y, y, (\hat{Y}, \hat{X}, y, SIG_{\hat{Y}}\{\!|y, m, \hat{X}|\!\}))$	(2.20)
AA1, **AR2**, **SEQ**	$\top[\mathbf{Init_{CR}}]_X \mathsf{Receive}(X, (\hat{Y}, \hat{X}, y, SIG_{\hat{Y}}\{\!|y, m, \hat{X}|\!\}))$	(2.21)
2.20, 2.21, **FS2**	$\top[\mathbf{Init_{CR}}]_X \mathsf{Honest}(\hat{Y}) \wedge \hat{Y} \neq \hat{X} \wedge$ $\mathsf{Receive}(Y, (\hat{X}, \hat{Y}, m)) \wedge$ $\mathsf{Send}(Y, (\hat{Y}, \hat{X}, y, SIG_{\hat{Y}}\{\!|y, m, \hat{X}|\!\})) \supset$ $\mathsf{Send}(Y, (\hat{Y}, \hat{X}, y, SIG_{\hat{Y}}\{\!|y, m, \hat{X}|\!\})) <$ $\mathsf{Receive}(X, (\hat{Y}, \hat{X}, y, SIG_{\hat{Y}}\{\!|y, m, \hat{X}|\!\}))$	(2.22)
AA4, **P1**	$\top[\mathbf{Init_{CR}}]_X$ $\mathsf{Receive}(X, (\hat{Y}, \hat{X}, y, SIG_{\hat{Y}}\{\!|y, m, \hat{X}|\!\})) <$ $\mathsf{Send}(X, (\hat{X}, \hat{Y}, SIG_{\hat{X}}\{\!|y, m, \hat{Y}|\!\}))$	(2.23)
2.15, 2.19, 2.22, 2.23	$\top[\mathbf{Init_{CR}}]_X \mathsf{Honest}(\hat{Y}) \wedge \hat{Y} \neq \hat{X} \supset \phi_{auth}$	(2.24)

Table 2.6: Strong authentication for the initiator role of the CR protocol

$(\hat{X}, \hat{Y}, SIG_{\hat{X}}\{\!|y, m, \hat{Y}|\!\})$. 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 2.6. Again, the formal proof naturally breaks down into three parts:

- Lines (2.16)–(2.19) 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 (2.20)–(2.22), 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 (2.23) we reason from Alice's actions that she sent out the third message after receiving the second message.

- Finally, in line (2.24), 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:

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

2. 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).

3. 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 2.5. 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:

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

As described in Section 2.3.8, 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 $(\mathbf{BS}_1, \ldots, \mathbf{BS}_4)$. These proofs are straightforward and we omit them here.

2.5 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 2.5.1. *(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 2.3 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.

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

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

2. 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,

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

3. 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.

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

4. 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 2.5.2 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.

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

Lemma 2.5.2. *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 2.5.3. *If $\vdash_{Q_1} \Gamma$ and $\Gamma \vdash \Psi$ and $\vdash_{Q_2} \Gamma$, then $\vdash_{Q_1|Q_2} \Psi$.*

Definition 2.5.4. *(Sequential Composition of Cords) Given cords*

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

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

their sequential composition is defined by

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

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 2.5.5. *(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 2.3), 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.

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

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

2. 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,

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

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

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

4. 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:

$$\Gamma_1 \cup \Gamma_2' \vdash \theta[P_1P_2]_X\psi$$

5. 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 2.5.6, 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 2.5.6. *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 2.5.7. *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 [57, 90, 89]to prove security properties over the complex control flow architectures of IEEE 802.11i and Kerberos V5.

Theorem 2.5.8. *If $\mathcal{Q}$ is a sequential composition of protocols $\mathcal{Q}_1$ and $\mathcal{Q}_2$ then we can conclude $\vdash_Q \mathsf{Honest}(\hat{X}) \supset \phi$ if the following conditions hold for all $r_1; r_2$ in $\mathcal{Q}$, where $r_1 \in \mathcal{Q}_1$ and $r_2 \in \mathcal{Q}_2$:*

1. *(Invariant induction)*
   - $\forall P \in BS(r_1).\ \vdash \theta_{r_1} \wedge \phi\ [P]_X\ \phi$ *and* $\forall P \in BS(r_2).\ \vdash \theta_{r_2} \wedge \phi\ [P]_X\ \phi$
2. *(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:

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

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

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

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

3. 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:

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

4. Prove that the invariants used in proving the properties of the protocols, $\Gamma_1 \cup \Gamma'_2$, hold for $\mathcal{Q}$ using theorem 2.5.8. From this and step (3), we can conclude:

$$\vdash_Q \theta[P_1P_2]_X\psi$$

2.5.1 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.

ISO-9798-3 Protocol as a Sequential Composition

Figure 2.3 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 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 2.1 and analyzed in Section 2.4. 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

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

Figure 2.3: *ISO-9798-3* protocol as arrows-and-messages

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:

$$\begin{array}{lcl} \mathbf{Init_{CR_0}} & \equiv & (\hat{Y}, m)[\\ & & \texttt{send}\ \hat{X}, \hat{Y}, m; \\ & & \texttt{receive}\ \hat{Y}, \hat{X}, y, s; \\ & & \texttt{verify}\ s, (y, m, \hat{X}), \hat{Y}; \\ & & r := \texttt{sign}\ (y, m, \hat{Y}), \hat{X}; \\ & & \texttt{send}\ \hat{X}, \hat{Y}, r; \\ ]_X() & & \end{array} \qquad \begin{array}{lcl} \mathbf{Resp_{CR_0}} & \equiv & (n)[\\ & & \texttt{receive}\ \hat{X}, \hat{Y}, x; \\ & & r := \texttt{sign}\ (n, x, \hat{X}), \hat{Y}; \\ & & \texttt{send}\ \hat{Y}, \hat{X}, n, r; \\ & & \texttt{receive}\ \hat{X}, \hat{Y}, t; \\ & & \texttt{verify}\ t, (n, x, \hat{Y}), \hat{X}; \\ ]_Y() & & \end{array}$$

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:

$$
\begin{array}{lcl}
\mathbf{Init_{ISO}} & \equiv & (\hat{Y}, m)[\\
 & & \texttt{new } x; \\
 & & gx := \texttt{expg } x; \\
 & & \texttt{send } \hat{X}, \hat{Y}, gx; \\
 & & \texttt{receive } \hat{Y}, \hat{X}, y, s; \\
 & & \texttt{verify } s, (y, gx, \hat{X}), \hat{Y}; \\
 & & r := \texttt{sign } (y, gx, \hat{Y}), \hat{X}; \\
 & & \texttt{send } \hat{X}, \hat{Y}, r; \\
]_X() & &
\end{array}
\qquad
\begin{array}{lcl}
\mathbf{Resp_{ISO}} & \equiv & (n)[\\
 & & \texttt{new } y; \\
 & & gy := \texttt{expg } y; \\
 & & \texttt{receive } \hat{X}, \hat{Y}, x; \\
 & & r := \texttt{sign } (gy, x, \hat{X}), \hat{Y}; \\
 & & \texttt{send } \hat{Y}, \hat{X}, gy, r; \\
 & & \texttt{receive } \hat{X}, \hat{Y}, t; \\
 & & \texttt{verify } t, (gy, x, \hat{Y}), \hat{X}; \\
]_Y() & &
\end{array}
$$

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 2.4. 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 2.4:

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

Base Diffie Hellman Protocol, DH_0: The property of the initiator role of the DH_0 protocol is given by the formula below.

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

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 2.5. 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:

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

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.

$$\Gamma_2 \vdash \mathsf{Start}(X)\,[\mathbf{Init_{DH_0}}]_X\,\mathsf{Fresh}(X, gx) \tag{2.25}$$

$$\Gamma_2 \vdash \mathsf{Fresh}(X, m)\,[\mathbf{Init_{CR_0}}]_X\,\mathsf{Honest}(\hat{Y}) \wedge \hat{Y} \neq \hat{X} \supset \phi_{auth} \tag{2.26}$$

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 2.25 matches the pre-condition of 2.26. 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 2.5.8 with the following preconditions to establish the invariants Γ_2:

$$\begin{aligned}
\text{For } \gamma_1 \;:\; & \theta_{Init_{DH_0}} \equiv \theta_{Resp_{DH_0}} \equiv \theta_{Resp_{CR_0}} \equiv \top \\
& \theta_{Init_{CR_0}} \equiv \mathsf{Gen}(X, m) \\
\text{For } \gamma_2 \;:\; & \theta_{Init_{DH_0}} \equiv \theta_{Resp_{DH_0}} \equiv \theta_{Init_{CR_0}} \equiv \top \\
& \theta_{Resp_{CR_0}} \equiv \mathsf{Fresh}(Y, n)
\end{aligned}$$

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.

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

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.

$$\begin{aligned} \vdash \quad & \mathsf{Start}(X)\,[\mathbf{Init_{DH_0}}]_X\,\mathsf{HasAlone}(X,x) \\ \vdash \quad & \mathsf{HasAlone}(X,x)\,\left[\mathbf{Init_{CR_0'}}\right]_X\,\mathsf{HasAlone}(X,x) \end{aligned}$$

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

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

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:

$$\begin{aligned} \vdash_{Q_{ISO}} \quad & \mathsf{Start}(X)\,\left[\mathbf{Init_{DH_0}}; \mathbf{Init_{CR_0'}}\right]_X\,\mathsf{Honest}(\hat{Y}) \supset \\ & \exists Y, y.\ (\mathsf{Has}(X, g^{xy}) \wedge (\mathsf{Has}(Z, g^{xy}) \supset (Z = X \vee Z = Y))) \end{aligned} \tag{2.27}$$

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.

Chapter 3

Computational PCL

Section 3.1 presents the syntax for defining roles of a protocol, while the syntax of the logic appears in Section 3.2. Some axioms and proof rules are described in Section 3.3, followed by a short example in Section 3.4. Section 3.5 presents the probabilistic polynomial-time execution and attacker model. The semantics of the logic are given in Section 3.6.

3.1 Protocol Syntax

Our"protocol programming language" is essentially the same as the one of PCL. The syntax of terms and actions is given in Table 3.1.

Names, sessions and threads: We use $\hat{X}, \hat{Y}, \ldots$ as *names* for protocol participants. Since a particular participant might be involved in more than one session at a time, we will give unique names to sessions and use $(\hat{X}, s)$ to designate a particular *thread* being executed by $\hat{X}$. All threads of a participant $\hat{X}$ share the same asymmetric key denoted by X. As a notational convenience, we will sometimes write X for an arbitrary thread of $\hat{X}$.

Terms, actions, and action lists: Terms name messages and their parts, such as nonces, keys, variables and pairs. For technical reasons, we distinguish *basic terms*

Terms:

N	::=	$\hat{X}$	(*name*)
S	::=	s	(*session*)
T	::=	(N, S)	(*thread*)
K	::=	X	(*asymmetric key*)
k	::=	x	(*symmetric key*)
n	::=	r	(*nonce*)
V	::=	x	(*term variable*)
t_B	::=	$V \mid K \mid T \mid N \mid n \mid (t_B, t_B)$	(*basic term*)
E	::=	$ENC_k\{\!\lvert t \rvert\!\}n$	(*encryption*)
Z	::=	$SIG_K\{\!\lvert t \rvert\!\}n$	(*signature*)
G	::=	t^n	(*exponentiation*)
t	::=	$t_B \mid E \mid Z \mid G \mid (t, t)$	(*term*)

Actions:

a	::=	
	|	`new` T, n
	|	$V :=$ `sign` T, t_B, K
	|	`verify` T, t_B, t_B, K
	|	$V :=$ `enc` T, t_B, k
	|	$V :=$ `dec` T, t_B, k
	|	$V :=$ `expg` T, n
	|	$k :=$ `dhkeyken` T, t_B, n
	|	$V, V :=$ `pick` t, t
	|	`match` $T, t_B/t_B$
	|	`send` T, t_B
	|	`receive` T, V

Table 3.1: Syntax of protocol terms and actions

from *terms* that may contain encryption. To account for probabilistic encryption, encrypted terms explicitly identify the randomness used for encryption. Specifically, $\{t\}^n_K$ indicates the encryption of t with key K using randomness n generated for the purpose of encryption. We write $m \subseteq m'$ when m is a subterm of $m' \in t$.

Actions include nonce generation, encryption, decryption, pattern matching, and communication steps (sending and receiving). An *ActionList* consists of a sequence of actions that contain only basic terms. This means that encryption cannot be performed implicitly; explicit `enc` actions, written as assignment, must be used instead. We assume that each variable will be assigned at most once, at its first occurrence. For any $s \in ActionList$, we write $s_{|X}$ to denote the subsequence of s containing only actions of a participant (or a thread) X.

Strands, roles, protocols and execution: A *strand* is an *ActionList*, containing actions of only one thread. Typically we will use notation $[ActionList]_X$ to denote a strand executed by thread X and drop the thread identifier from the actions themselves. A *role* is a strand together with a basic term representing the initial knowledge

Action Predicates:

$$\begin{array}{lll} \mathsf{a} & ::= & \mathsf{Send}(T,t) \mid \mathsf{Receive}(T,t) \mid \mathsf{Verify}(T,t,N) \mid \\ & & \mathsf{Sign}(T,t) \mid \mathsf{Encrypt}(T,t,k) \mid \mathsf{Decrypt}(T,t,k) \mid \\ & & \mathsf{New}(T,n) \end{array}$$

Formulas:

$$\begin{array}{lll} \varphi & ::= & \mathsf{a} \mid t = t \mid \mathsf{Start}(T) \mid \mathsf{Possess}(T,\ t) \mid \mathsf{Indist}(T,\ t) \mid \\ & & \mathsf{GoodKeyAgainst}(T,\ t) \mid \mathsf{Fresh}(T,\ t) \mid \\ & & \mathsf{Honest}(N) \mid \mathsf{Contains}(t,\ t) \mid \\ & & \mathsf{DHSource}(T,\ t) \mid \mathsf{PSource}(T,\ n,t,t) \mid \\ & & \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \exists V.\, \varphi \mid \forall V.\, \varphi \mid \neg\varphi \mid \varphi \supset \varphi \mid \varphi \Rightarrow \varphi \end{array}$$

Modal formulas:

$$\Psi \quad ::= \quad \varphi\ [Strand]_T\ \varphi$$

Table 3.2: Syntax of the logic

of the thread. A *protocol* is a finite set of *Roles*, together with a basic term representing the initial intruder knowledge.

An *execution strand* is a pair $ExecStrand ::= InitialState(\mathcal{I}); ActionList$ where $\mathcal{I}$ is a data structure representing the initial state of the protocol, as produced by the initialization phase from Section 3.5. In particular, this includes the list of agents and threads, the public/private keys and honesty/dishonesty tokens of each agent, and the roles played by each thread.

3.2 Logic Syntax

The syntax of formulas is given in Table 3.2. Most formulas have the same intuitive meaning in the computational semantics as in the symbolic model presented earlier, except for predicates Possess and Indist. We summarize the meaning of formulas informally below, with precise semantics in the next section. Description and semantics of constructs involving signatures and Diffie-Hellman key exchange is deferred to the next chapter.

For every protocol action, there is a corresponding action predicate which asserts

that the action has occurred in the run. For example, $\mathsf{Send}(X, t)$ holds in a run where the thread X has sent the term t. $\mathsf{Fresh}(X, t)$ means that the value of t generated by X is "fresh" in the sense that no one else has seen any messages containing t, while $\mathsf{Honest}(\hat{X})$ means that $\hat{X}$ is acting honestly, *i.e.,* the actions of every thread of $\hat{X}$ precisely follows some role of the protocol. The Source predicate is used to reason about the source of a piece of information, such as a nonce. Intuitively, the formula $\mathsf{Source}(Y, u, \{m\}^r_X)$ means that the only way for a thread X different from Y to know u is to learn u from the term $\{m\}^r_X$, possibly by some indirect path.

The predicate Fresh is definable by $\mathsf{Fresh}(X, v) \equiv \mathsf{New}(X, v) \wedge \neg(\exists u.\, \mathsf{Send}(X, u) \wedge \mathsf{Contains}(u, v))$ and classical implication is definable by $A \supset B \equiv \neg A \vee B$.

In the symbolic model [32, 36],the predicate Has states that a principal can "derive" a message or its contents from the information gathered during protocol execution. We use $\mathsf{Possess}(X, t)$ to state that it is possible to derive t by Dolev-Yao rules from X's view of the run and $\mathsf{Indist}(X, t)$ to state that no probabilistic polynomial-time algorithm, given X's view of the run, can distinguish t from a random value from the same distribution. Typically, we use $\mathsf{Possess}$ to say that some honest party obtained some secret, and Indist to say that the attacker does not have any partial information about a secret.

3.3 Proof System

Example axioms and rules are given in Table 3.3.These axioms express reasoning principles that can be justified using complexity-theoretic reductions, information-theoretic arguments, and asymptotic calculations. However, the advantage of the proof system is that its justification using cryptographic-style arguments is a one-time mathematical effort; protocol proofs can be carried out symbolically using the proof system without explicitly reasoning about probability and complexity. Another advantage of the axiomatic approach is that different axioms and rules rest on different cryptographic assumptions. Therefore, by examining the axioms and rules used in a specific proof, we can identify specific properties of the cryptographic primitives that are needed to guarantee protocol correctness. This provides useful information

Axioms:

$$
\begin{array}{rcl}
\textbf{AA1} & : & \top[\mathsf{a}]_X \mathsf{a} \\
\textbf{P} & : & \mathsf{Persist}(X,t)[a]_X \mathsf{Persist}(X,t) \\
 & & \text{with } \mathsf{Persist} \in \{\mathsf{Send}, \mathsf{Receive}, \mathsf{Verify}, \mathsf{Sign}, \mathsf{Encrypt}, \dots\} \\
\textbf{AN2} & : & \top[\texttt{new}\ x]_{\tilde{X}} \tilde{Y} \neq \tilde{X} \Rightarrow \mathsf{Indist}(\tilde{Y}, x) \\
\textbf{AN3} & : & \top[\texttt{new}\ x]_{\tilde{X}} \mathsf{Fresh}(\tilde{X}, x) \\
\textbf{S1} & : & \mathsf{Source}(\tilde{Y}, u, \{m\}^r_X) \wedge \neg \mathsf{DecryptsHonest}(\hat{X}, \{m\}^r_X) \wedge \hat{Z} \neq \hat{X} \wedge \hat{Z} \neq \hat{Y} \wedge \\
 & & \mathsf{Honest}(\hat{X}) \wedge \mathsf{Honest}(\hat{Y}) \Rightarrow \mathsf{Indist}(\tilde{Z}, u)
\end{array}
$$

Proof rules:

$$\frac{\theta[P]_X\varphi \quad \theta' \supset \theta \quad \varphi \supset \varphi'}{\theta'[P]_X\varphi'}\ \textbf{G3} \qquad \frac{\theta[P_1]_X\varphi \quad \varphi[P_2]_X\psi}{\theta[P_1P_2]_X\psi}\ \textbf{SEQ}$$

$$\frac{\varphi \quad \varphi \Rightarrow \psi}{\psi}\ \textbf{MP} \qquad \frac{\varphi}{\forall x.\varphi}\ \textbf{GEN}$$

$$\frac{Q \models \mathsf{Start}\,[]_X\,\varphi \quad \forall P \in S(Q), Q \models \varphi\,[P]_X\,\varphi}{\mathsf{Honest}(X) \supset \varphi}\ \textbf{HON}$$

Table 3.3: Fragment of the proof system

in protocol design because primitives that provide weaker properties often have more efficient constructions.

Axioms: Axioms **AN2** and **AN3** capture some of the properties of nonce generation. Informally, **AN2** states that if a thread X generates a fresh nonce x and does not perform any additional actions, then x is indistinguishable from a random value for all other threads. The soundness of this axiom is established by a simple information-theoretic argument. The informal interpretation of axiom **S1** (also called the "Source" axiom) is that, unless a ciphertext is decrypted, a thread which does not possess the decryption key cannot extract any partial information about the plaintext. The soundness of this axiom is proved by a complexity-theoretic reduction. Specifically, we show in [37] that if an attacker can break this property, then

there is another attacker that can break the underlying IND-CCA2 secure encryption scheme [14].

Inference rules: Inference rules include generic rules from modal logics (e.g. **G3**), sequencing rule **SEQ** used for reasoning about sequential composition of protocol actions and a rule (called the honesty rule) for proving protocol invariants using induction. These rules are analogous to proof rules from our earlier work [32, 36].

First-order axioms and rules: We use two implications: a conditional implication $\Rightarrow$, discussed and defined precisely in section 3.6, and a classical implication $\supset$ with $A \supset B \equiv \neg A \vee B$. While standard classical tautologies hold for classical implication, some familiar propositional or first-order tautologies may not hold when written using $\Rightarrow$ instead of $\supset$. However, modus ponens and the generalization rule above are sound. The soundness of modus ponens relies on the simple asymptotic fact that the sum of two negligible functions is a negligible function. In future work, we hope to develop a more complete proof system for the first-order fragment of this logic.

3.4 Example

In this section, we present a simple protocol and state a secrecy property that can be proved using the proof system. The interested reader is referred to [37, 44, 32, 36] for further explanation and examples. The two protocol roles are:

$$\begin{aligned} \textbf{Init} &\equiv [\texttt{new}\ x;\ y := \texttt{enc}\langle x, X\rangle, Y;\ \texttt{send}\ \hat{X}, \hat{Y}, y]_X \\ \textbf{Resp} &\equiv [\texttt{receive}\ z;\ \texttt{match}\ z/\langle \hat{X}, \hat{Y}, z'\rangle;\ z'' := \texttt{dec}\ z', Y]_Y \end{aligned}$$

The initiator generates a new nonce and sends it encrypted to the responder. The responder receives the message and recovers the nonce by decrypting the ciphertext. We can prove that if X completes the protocol with Y, then x will be a shared secret between them, provided both agents are honest. Formally,

$$\mathsf{Start}(X)[\textbf{Init}]_X \mathsf{Honest}(\hat{X}) \wedge \mathsf{Honest}(\hat{Y}) \wedge (Z \neq X) \wedge (Z \neq Y) \Rightarrow \mathsf{Indist}(Z, x)$$

Since the meaning of $\mathsf{Indist}(Z, x)$ (formally defined in Section 3.6) is that Z cannot distinguish the secret nonce x from a randomly chosen nonce, this formula expresses a standard form of secrecy used in the cryptographic literature.

3.5 Protocol Execution

Given a protocol, adversary, and value of the security parameter, we define a set of protocol traces, each associated with the random bits that produce this sequence of actions and additional randomness for algorithms used in the semantics of formulas about the run. The definition proceeds in two phases. In the initialization phase, we assign a set of roles to each principal, identify a subset which is honest, and provide all entities with private-public key pairs and random bits. In the execution phase, the adversary executes the protocol by interacting with honest principals, as in the accepted cryptographic model of [17].

Initialization: We fix the protocol Q, adversary A, security parameter η, and some randomness R of size polynomially bounded in η. Each principal and each thread (*i.e.*, an instance of a protocol role executed by the principal) is assigned a unique bitstring identifier. We choose a sufficiently large polynomial number of bitstrings $i \in I \subseteq \{0,1\}^\eta$ to represent the names of principals and threads. Randomness R is split into r_i for each honest $i \in I$ (referred to as "coin tosses of honest party i") and R_A (referred to as "adversarial randomness").

The adversary designates some of the principals as *honest* and the rest of the principals as *dishonest*. Intuitively, honest principles will follow one or more roles of the protocol faithfully. The adversary chooses a set of threads, and to each thread it assigns a strand (a program to be executed by that thread), under the restriction that all threads of honest principals are assigned roles of protocol Q.

The key generation algorithm $\mathcal{K}$ of a public-key encryption scheme $(\mathcal{K}, \mathcal{E}, \mathcal{D})$ is run on $\mathbf{1}^\eta$ for each participant a using randomness r_a, and producing a public-private key pair (pk_a, sk_a). The public key pk_a is given to all participants and to the adversary A; the private key is given to all threads belonging to this principal and to the adversary

if the principal is dishonest.

Generating Computational Traces: Following [17], we view an agent i trying to communicate with agent j in protocol session s as a (stateful) oracle $\Pi^s_{i,j}$. The state of each oracle is defined by a mapping λ from atomic symbols to bitstrings (with variables and nonces renamed to be unique for each role) and a counter c. Each oracle proceeds to execute a step of the protocol as defined by actions in the corresponding role's action list, when activated by the adversary.

We omit the details of communication between the adversary and the oracles, and focus on computational interpretation of symbolic protocol actions. Let a_c be the current action in the *ActionList* defining some role of participant i in session s, *i.e.*, $Thread = (i', s')$ where $i = \lambda(i'), s = \lambda(s')$.

If $a_c = (\texttt{new}\ (i', s'), v)$, then update λ so that $\lambda(v) = NonceGen(R_i)$, where $NonceGen$ is a nonce generation function(*e.g.*, $NonceGen$ simply extracts a fresh piece of R_i). If $a_c = (v := \texttt{enc}\ (i', s'), j, u)$, then update λ so that $\lambda(v) = \mathcal{E}(\lambda(u), pk_j, R_i)$ where $\mathcal{E}(\lambda(u), pk_j, R_i)$ is the result of executing the public-key encryption algorithm on plaintext $\lambda(u)$ with public key pk_j and fresh randomness extracted from R_i. For brevity, we omit computational interpretation of decryption and matching (pairing, unpairing, and equality-test) actions. Sending a variable $\texttt{send}\ (i', s'), v$ is executed by sending $\lambda(v)$ to the adversary, and receiving $\texttt{receive}\ (i', s'), v$ is executed by updating λ so that $\lambda(v) = m$ where m is the bitstring sent by the adversary.

At any time during the protocol execution, the adversary A may record any internal, private message on a special *knowledge tape*. This tape is not read by any participant of the protocol. However, its content will be made available to the test algorithms used to decide if a given security formula containing $\mathsf{Indist}(...)$ is valid or not. Let K be $[(i_1, m_1), .., (i_n, m_n)]$ the list of messages m_k written by A on the knowledge tape, indexed by the number of actions i_k already executed when m_k was written (position in the protocol execution). This index will be useful to remember a previous state of the knowledge tape.

At the end of the protocol execution, the adversary A outputs a pair of integers (p_1, p_2) on an *output tape*. When the security formula is a modal formula $\theta[P]_X\varphi$, these

two integers represent two positions in the protocol execution where the adversary claims that the formula is violated, i.e. that θ is true in p_1 but φ is false in p_2, with P between p_1 and p_2. Let O be this pair (p_1, p_2) of integers written on the output tape.

The symbolic trace of the protocol is the execution strand $e \in ExecStrand$ which lists, in the order of execution, all honest participant actions and the dishonest participant's `send` and `receive` actions. This strand contains two parts: $InitialState(\mathcal{I})$ stores the initialization data, and the rest is an ordered list of all exchanged messages and honest participants' internal actions.

Definition 3.5.1. *(Computational Traces) Given a protocol Q, an adversary A, a security parameter η, and a sequence of random bits $R \in \{0,1\}^{p(\eta)}$ used by the honest principals and the adversary, a* run *of the protocol is the tuple $\langle e, \lambda, O, K, R\rangle$ where e is the symbolic execution strand, $\lambda : Term(e) \to \{0,1\}^{p(\eta)}$ maps the symbolic terms in e to bitstrings, O is the pair of integers written on the output tape, and K is the indexed list of messages written on the knowledge tape. Finally, $p(x)$ is a polynomial in x.*

A computational trace *is a run with two additional elements: $R_T \in \{0,1\}^{p(\eta)}$, a sequence of random bits used for testing indistinguishability, and $\sigma : FVar(\varphi) \to \{0,1\}^{p(\eta)}$, a substitution that maps free variables in a formula to bitstrings. The set of computational traces is*

$$T_Q(A, \eta) = \{\langle e, \lambda, O, K, R, R_T, \sigma\rangle \mid R, R_T \text{ chosen uniformly}\}.$$

Definition 3.5.2. *(Participant's View) Given a protocol Q, an adversary A, a security parameter η, a participant X and a trace $t = \langle e, \lambda, O, K, R, R_T, \sigma\rangle \in T_Q(A, \eta)$, $View_t(X)$ represents $X's$ view of the trace. It is defined precisely as follows:*

If $\hat{X}$ is honest, then $View_t(X)$ is the initial knowledge of X, a representation of $e_{|X}$ and $\lambda(x)$ for any variable x in $e_{|X}$. If $\hat{X}$ is dishonest, then $View_t(X)$ is the union of the knowledge of all dishonest participants X' after the trace t (where $View_t(X')$ is defined as above for honest participants) plus K, the messages written on the knowledge tape by the adversary.

The following three definitions are used in the semantics of the predicate Indist(). Informally, based on some trace knowledge K, the distinguisher D tries to determine which of two bitstrings is the value of a symbolic term. One of the bitstrings will be the computational value of the term in the current run, while the other will be a random bitstring of the same structure, chosen in a specific way. The order of the two bitstrings presented to the distinguisher is determined by an LR Oracle using a random selector bit.

Definition 3.5.3. *(LR Oracle) The LR Oracle [14] is used to determine the order in which two bitstrings are presented depending on the value of the selector bit, i.e.* $LR(s_0, s_1, b) = \langle s_b, s_{1-b} \rangle$.

Definition 3.5.4. *(Distinguishing test input) Let u be a symbolic term and σ be a substitution that maps variables of u to bitstrings. We construct another bitstring $f(u, \sigma, r)$, whose symbolic representation is the same as u. Here, r is a sequence of bits chosen uniformly at random. The function f is defined by induction over the structure of the term u.*

- *Nonce* u *:* $f(u, \sigma, r) = r$
- *Name/Key* u *:* $f(u, \sigma, r) = \sigma(u)$
- *Pair* $u = \langle u_1, u_2 \rangle$ *:* $f(\langle u_1, u_2 \rangle, \sigma, r_1; r_2) = \langle f(u_1, \sigma, r_1), f(u_2, \sigma, r_2) \rangle$
- *Encryption* $u = \{v\}_K^n$*:* $f(\{v\}_K^n, \sigma, r_1; r_2) = \mathcal{E}(f(v, \sigma, r_1), \sigma(K), r_2)$

Definition 3.5.5. *(Distinguisher) A distinguisher D is a polynomial time algorithm which takes as input a tuple $\langle K, t, \langle s_0, s_1 \rangle, R, \eta \rangle$, consisting of knowledge K, symbolic term t, two bitstrings s_0 and s_1, randomness R and the security parameter η, and outputs a bit b'.*

The next definition is used while defining semantics of modal formulas. Given a set T of traces and a strand P of actions executed by a thread X, the set T_P includes only those traces from T which contain P. $Pre(T_P)$ is obtained from T_P by taking the initial segment of each trace upto the point where P starts. The precondition

of a modal formula is evaluated over this set. $PostT_P$ is similarly defined; the only difference is now the trace is cut at the point that P ends. The postcondition of a modal formula is evaluated over this set. The begin and end positions are determined by the component O in the trace.

Definition 3.5.6. *(Splitting computational traces) Let T be a set of computational traces and $t = \langle e, \lambda, O, K, R, R_T, \sigma \rangle \in T$. $O = \langle p_1, p_2 \rangle$, $e = InitialState(\mathcal{I}); s$, and $s = s_1; s_2; s_3$ with p_1, p_2 the start and end positions of s_2 in s. Given a strand P executed by participant X, we denote by T_P the set of traces in T for which there exists a substitution σ' which extends σ to variables in P such that $\sigma'(P) = \lambda(s_{2\,|X})$. The complement of this set is denoted by $T_{\neg P}$ and contains all traces which do not have any occurrence of the strand P. We define the set of traces $Pre(T_P) = \{t[s \leftarrow s_1, K \leftarrow K_{\leq p_1}, \sigma \leftarrow \sigma'] \mid t \in T_P\}$, where $K_{\leq p}$ is the restriction of the knowledge tape K to messages written before the position p. We define the set of traces $Post(T_P) = \{t[s \leftarrow s_1; s_2, K \leftarrow K_{\leq p_2}, \sigma \leftarrow \sigma'] \mid t \in T_P\}$.*

3.6 Computational Semantics

The semantics of a formula φ on a set T of computational traces is a subset $T' \subseteq T$ that respects φ in some specific way. For many predicates and connectives, the semantics is essentially straightforward. For example, an action predicate such as Send selects a set of traces in which a send occurs. However, the semantics of predicates Indist and Possess is inherently more complex.

Intuitively, an agent possesses the value of an expression (such as another agent's nonce or key) if the agent can compute this value from information it has seen, with high probability. If an agent is honest, and therefore follows the rules of the protocol, then it suffices to use a simple, symbolic algorithm for computing values from information seen in the run of a protocol. For dishonest agents, we would prefer in principle to allow any probabilistic polynomial-time algorithm. However, quantifying over such algorithms, in a way that respects the difference between positive and negative occurrences of the predicate in a formula, appears to introduce some technical complications. Therefore, in the interest of outlining a relatively simple form of

computational semantics, we will use a fixed algorithm. This gives a useful semantics for formulas where $\mathsf{Possess}(X, u)$ is used under the hypothesis that $\hat{X}$ is honest. We leave adequate treatment of the general case for future work.

Intuitively, an agent has partial information about the value of some expression if the agent can distinguish that value, when presented, from a random value generated according to the same distribution. More specifically, an agent has partial information about a nonce u if, when presented with two bitstrings of the appropriate length, one the value of u and the other chosen randomly, the agent has a good chance of telling which is which. As with Possess, there are technical issues associated with positive and negative occurrences of the predicate. For positive occurrences of Indist, we should say that *no* probabilistic polynomial-time algorithm has more than a negligible chance, where as for $\neg\mathsf{Indist}(\ldots)$ we want to say that *there exists* a probabilistic polynomial-time distinguisher. In order to present a reasonably understandable semantics, and establish a useful basis for further exploration of computational semantics of symbolic security logics, we give an interpretation that appears accurate for formulas that have only positive occurrences of Indist and could be somewhat anomalous for formulas that contain negative occurrences. This seems adequate for reasoning about many secrecy properties, since these are expressed by saying that at the end of any run of the protocol, a value used in the run is indistinguishable from random.

Conditional implication $\theta \Rightarrow \varphi$ is interpreted using the negation of θ and the conditional probability of φ given θ. This non-classical interpretation of implication seems to be essential for relating provable formulas to cryptographic-style reductions involving conditional probabilities. In particular, the soundness proof for the **SIG** axiom, presented in the next Chapter, uses the conditional aspect of this implication in a fundamental way. On the other hand, $\Rightarrow$ coincides with $\supset$ in formulas where Indist does not appear on the right hand size of the implication.

We inductively define the semantics $[\![\,\varphi\,]\!]\,(T, D, \epsilon)$ of a formula φ on the set T of traces, with distinguisher D and tolerance ϵ. The distinguisher and tolerance are not used in any of the clauses except for Indist, where they are used to determine whether the distinguisher has more than a negligible chance of distinguishing the given value from a random value. In definition 3.6.1 below, the tolerance is set to a negligible

function of the security parameter and $T = T_Q(A, \eta)$ is the set of traces of a protocol Q with adversary A.

- $[\![\mathsf{Send}(X, u)]\!] (T, D, \epsilon)$ is the collection of all $\langle e, \lambda, O, K, R, R_T, \sigma \rangle \in T$ such that some action in the symbolic execution strand e has the form $\texttt{send}\ Y, v$ with $\lambda(Y) = \sigma(X)$ and $\lambda(v) = \sigma(u)$. Recall that σ maps formula variables to bitstrings and represents the environment in which the formula is evaluated.

- $[\![\mathsf{a}(\cdot, \cdot)]\!] (T, D, \epsilon)$ for other action predicates a is similar to $\mathsf{Send}(X, u)$.

- $\left\| \left[\mathsf{Honest}(\hat{X}) \right] \right\| (T, D, \epsilon)$ is the collection of all $\langle e, \lambda, O, K, R, R_T, \sigma \rangle \in T$ where $e = InitialState(\mathcal{I}); s$ and $\sigma(X)$ is designated *honest* in the initial configuration $\mathcal{I}$. Since we are only dealing with static corruptions in this paper, the resulting set is either the whole set T or the empty set ϕ depending on whether a principal is honest or not.

- $[\![\mathsf{Start}(X)]\!] (T, D, \epsilon)$ includes all traces $\langle e, \lambda, O, K, R, R_T, \sigma \rangle \in T$ where $e = InitialState(\mathcal{I}); s$ and $\lambda(s)_{|\sigma(X)} = \epsilon$. Intuitively, this set contains traces in which X has executed no actions.

- $[\![\mathsf{Contains}(u, v)]\!] (T, D, \epsilon)$ includes all traces $\langle e, \lambda, O, K, R, R_T, \sigma \rangle \in T$ such that there exists a series of decryptions with $\{\lambda(k) \mid k \in Key\}$ and projections (π_1,π_2) constructing $\sigma(v)$ from $\sigma(u)$. This definition guarantees that the result is the whole set T if v is a symbolic subterm of u.

- $[\![\mathsf{ContainsOut}(u,\ v,\ t)]\!] (T, D, \epsilon)$ includes all traces $\langle e, \lambda, O, K, R, R_T, \sigma \rangle \in T$ such that there exists a series of projections (π_1,π_2) and decryptions with $\{\lambda(k) \mid k \in Key\}$, where $\sigma(t)$ is never decomposed, creating $\sigma(v)$ from $\sigma(u)$. This definition ensures that the result is the whole set T if v is a symbolic subterm of u but is not a subterm of t.

- $[\![\theta \wedge \varphi]\!] (T, D, \epsilon) = [\![\theta]\!] (T, D, \epsilon) \cap [\![\varphi]\!] (T, D, \epsilon)$.

- $[\![\theta \vee \varphi]\!] (T, D, \epsilon) = [\![\theta]\!] (T, D, \epsilon) \cup [\![\varphi]\!] (T, D, \epsilon)$.

- $[\![\neg\varphi]\!] (T, D, \epsilon) = T \setminus [\![\varphi]\!] (T, D, \epsilon)$.

- $[\![\exists x.\, \varphi]\!]\,(T, D, \epsilon) = \bigcup_\beta ([\![\varphi]\!]\,(T[x \leftarrow \beta], D, \epsilon)[x \leftarrow \sigma(x)])$
  with $T[x \leftarrow \beta] = \{t[\sigma[x \leftarrow \beta]] \mid t = \langle e, \lambda, O, K, R, R_T, \sigma\rangle \in T\}$, and β any bitstring of polynomial size.

- $[\![\theta \Rightarrow \varphi]\!]\,(T, D, \epsilon) = [\![\neg\theta]\!]\,(T, D, \epsilon) \cup [\![\varphi]\!]\,(T', D, \epsilon)$, where $T' = [\![\theta]\!]\,(T, D, \epsilon)$. Note that the semantics of φ is taken over the set T' given by the semantics of θ, as discussed earlier in this section.

- $[\![u = v]\!]\,(T, D, \epsilon)$ includes all traces $\langle e, \lambda, O, K, R, R_T, \sigma\rangle \in T$ such that $\sigma(u) = \sigma(v)$.

- $[\![\mathsf{DecryptsHonest}(Y,\, \{u\}^r_X)]\!]\,(T, D, \epsilon) = [\![\varphi]\!]\,(T, D, \epsilon)$ with $\varphi = \mathsf{Honest}(\hat{X}) \wedge \exists v.\, v := \mathtt{dec}\ Y,\, \{u\}^r_X$.

- $[\![\mathsf{Source}(Y,\, u,\, \{m\}^r_X)]\!]\,(T, D, \epsilon) = [\![\exists v. \forall w.\, \varphi]\!]\,(T, D, \epsilon)$ with :

$$\begin{aligned} \varphi \;=\;& \mathsf{New}(Y, u) \wedge \mathsf{Contains}(m,\, u) \\ \wedge\;& \mathsf{Contains}(v,\, \{m\}^r_X) \wedge \mathsf{Send}(Y, v) \\ \wedge\;& \neg\mathsf{ContainsOut}(v,\, u,\, \{m\}^r_X) \\ \wedge\;& (v \neq w \wedge \mathsf{Contains}(w,\, u)) \Rightarrow \neg\mathsf{Send}(Y, w) \end{aligned}$$

- $[\![\mathsf{Possess}(X,\, u)]\!]\,(T, D, \epsilon)$ includes all traces $t = \langle e, \lambda, O, K, R, R_T, \sigma\rangle \in T$ such that $\sigma(u)$ can be built from $View_t(\sigma(X))$ with the Dolev-Yao deduction rules.

- $[\![\mathsf{Indist}(X,\, u)]\!]\,(T, \epsilon, D) = T$ if

$$\frac{|\{D(View_t(\sigma(X)), u, LR(\sigma(u), f(u, \sigma, r), b), R_D, \eta) = b \mid t \in T\}|}{|T|} \leq \frac{1}{2} + \epsilon$$

  and the empty set ϕ otherwise. Here, the random sequence $b; r; R_D = R_T$, the testing randomness for the trace t.

- $[\![\theta[P]_X\varphi]\!]\,(T, D, \epsilon) = T_{\neg P} \cup [\![\neg\theta]\!]\,(Pre(T_P), D, \epsilon) \cup [\![\varphi]\!]\,(Post(T_P), D, \epsilon)$ with $T_{\neg P}$, $Pre(T_P)$, and $Post(T_P)$ as given by Definition 3.5.6.

Definition 3.6.1. *A protocol Q satisfies a formula φ, written $Q \models \varphi$, if $\forall A$ providing an active protocol adversary, $\forall D$ providing a probabilistic-polynomial-time distinguisher, $\forall \nu$ giving a negligible function, $\exists N$, $\forall \eta \geq N$,*

$$|\, [\![\, \varphi \,]\!]\, (T, D, \nu(\eta))\, |\ /\ |\, T\, |\ \geq\ 1 - \nu(\eta)$$

where $[\![\, \varphi \,]\!]\, (T, D, \nu(\eta))$ is the subset of T given by the semantics of φ and $T = T_Q(A, \eta)$ is the set of computational traces of protocol Q generated using adversary A and security parameter η, according to Definition 3.5.1.

Theorem 3.6.2. *(Soundness)* $\quad \forall Q,\ \forall \varphi, \quad Q \vdash \varphi \ \Rightarrow\ Q \models \varphi$

Chapter 4

Analyzing Key Exchange Protocols Using CPCL

4.1 Security Model

In this section, we describe some problems with inductive compositional reasoning about key indistinguishability and present an alternative condition that appears more suitable for our purposes. We also give a definition of secure session that uses a key. These definitions are formulated within a complexity-theoretic model, in the style of modern cryptographic game-based definitions [17]. In subsequent sections, we use these definitions to develop a cryptographically sound proof system for reasoning about key exchange protocols. The soundness proofs are subtle and involve complexity-theoretic reductions.

4.1.1 Key Indistinguishability

Our goal is to develop a formal method for compositional proofs of protocol suites involving key exchange protocols. Since we intend to apply our method to prove security properties of existing protocols in common use, as presented in RFCs and as currently implemented, we aim to accomodate conditions such as semantic security that may be weaker than the security conditions advocated by some cryptographers.

A central concept in many compositional proof methods [6, 45, 46, 25, 36] is an *invariant*. In developing compositional security proofs of complex protocols [57], we require that each protocol component respects the invariants of the other components in the system [36].

It appears that standard cryptographic security definitions for key exchange like *key indistinguishability* [17, 15] are *not* invariant in the manner needed for an inductive proof of composition. Even if a key exchange protocol, run by itself in isolation, produces a key that is indistinguishable from a random value chosen from the same distribution, key indistinguishability is generally lost as soon as the key is used to encrypt a message of a known form or with partially known possible content. Moreover, some situations allow one agent to begin transmitting encrypted data before the other agent finishes the last step of the key exchange, rendering key indistinguishability false at the point that the key exchange protocol finishes. (This appears to be the case for SSL [48]; see [81] for a discussion of data transfer before the key exchange *finished* messages are received.) Furthermore, some key exchange protocols even use the generated key during the protocol, preventing key indistinguishability. Fortunately, many protocols that use keys do not require key indistinguishability to provide meaningful security guarantees. In particular, semantic security [51] does *not* require that the keys used remain indistinguishable from random.

To circumvent the technical problems we encountered in working with key indistinguishability, we develop an alternative notion that is parameterized by the security goal of the application in which the resulting key is used. As concrete examples, we consider cases where the key is used for encryption or MAC. The security definition for key exchange requires that the key produced is "good" for that application, i.e. an adversary interacting with the encryption scheme using this key cannot win the security game for that scheme (for example, the IND-CPA game for encryption). The resulting definition for key exchange is invariant under composition with the application protocol which uses the key.

Some interesting issues related to key indistinguishability were raised by Rackoff, who proposed an example protocol that is explained in the appendix of [26]. Rackoff's example illustrates the difference between key indistinguishability based on an

attacker who uses a challenge (the key or a surrogate chosen randomly from the same distribution) to interact with subsequent steps of the key exchange protocol, and indistinguishability based on an attacker who cannot execute further protocol steps. The definition of key usability that we use in this paper is similar to the weaker notion of key indistinguishability in that the adversary who attempts to win, for example, the IND-CPA game for encryption, does not have the opportunity to interact further with other protocol participants. On the other hand, because protocols (such as SSL [48]) that provide key confirmation steps will also fail the stronger form of definition suggested by Rackoff's example, we consider the weaker condition we use advantageous for certain practical settings. Specifically, we believe that whatever security properties are enjoyed by practical key exchange protocols in current use, there is merit in being able to state and prove these properties precisely. We also believe that the setting presented in this paper provides a useful starting point for expressing and reasoning about stronger security conditions.

We emphasize that the definition and subsequent theorems below apply to any cryptographic primitive that satisfies the application game condition, e.g., the **ENC** axiom (presented in Section 4.2) holds for any encryption scheme that satisfies the IND-CPA condition.

4.1.2 Key Usability

While there are many desirable properties a "good" key exchange protocol might satisfy, such as key freshness, high key entropy, and agreement, one essential property is that the key should be suitable for use. Specifically, an adversary who interacts with the the key exchange protocol should not be able to extract information that can compromise the application protocol which uses the resulting key. This is the main idea underlying the security definition that we develop in this section. The specific applications that we focus on here are symmetric encryption and message authentication codes. But the definition can be extended in a natural manner to cover other primitives. There are several ways in which this intuition can be translated into a security definition. We present one such definition below.

We define usability of keys obtained through a key exchange protocol Σ with respect to a class of applications S via a two-phase experiment. The experiment involves a two-phase adversary $\mathcal{A} = (\mathcal{A}_e, \mathcal{A}_c)$. In the *key exchange phase*, the honest parties run sessions of the protocol following the standard execution model: each principal executes multiple sessions of the protocol (as both initiator and responder) with other principals; the communication between parties is controlled by the adversary $\mathcal{A}_e$. At the end of the key exchange phase, the adversary selects a challenge session sid among all sessions executed by the honest parties, and outputs some state information St representing the information $\mathcal{A}_e$ was able to gather during its execution. Let k be the key locally output by the honest party executing the session sid. At this point, the experiment enters its second phase—the *challenge phase* where the goal of the adversary is to demonstrate an attack against a scheme $\Pi \in S$ which uses the key k. After $\mathcal{A}_e$ receives as input St, it starts interacting with Π according to the game used for defining security of the application protocols in S. For example, if S is a set of encryption schemes, then the relevant game may be IND-CPA, IND-CCA1, or IND-CCA2 [50]. Since the specific task we treat in this paper is secure sessions, we formalize the case when the game defines IND-CPA security. Thus, in playing the game, $\mathcal{A}_c$ has access to a left-right encryption oracle under k, and in addition, it receives as input the state information from $\mathcal{A}_e$. The advantage of the adversary is defined as for the standard IND-CPA game with the difference that the probability is taken over the random coins of the honest parties (used in the execution of the protocol), the coins of the two adversaries, and the coins used for encryption in the challenge phase. The keys obtained by running the key exchange protocol are usable for the schemes in S if this advantage is bounded above by a negligible function of the security parameter, for *all* encryption schemes in S. The universal quantification over schemes is used to capture the fact that the security property is guaranteed for all encryption schemes which satisfy the IND-CPA condition. This idea is formally stated as Definition 4.1.1 below.

We note that in this definition the adversary $\mathcal{A}_c$ is not allowed to interact with the key exchange protocol in the challenge phase. In subsequent work, we plan to explore variants of this definition with other forms of communication between the adversaries

$\mathcal{A}_e$ and $\mathcal{A}_c$.

Definition 4.1.1. *Consider the following experiment* $\mathbf{Exp}^{ke_b}_{\mathcal{A},\Sigma,\Pi}(\eta)$*, involving an adversary* $\mathcal{A} = (\mathcal{A}_e, \mathcal{A}_c)$*, a key exchange protocol* Σ *and an encryption scheme* $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$*. The experiment is parametrized by the bit* b*.*

- *The adversary* $\mathcal{A}_e$ *is given as input the security parameter and can make the following queries:*

  – Request that a new principal i *be added to the system: new pairs of encryption/decryption keys are generated for the principal via* $(pk_i, sk_i) \xleftarrow{\$} \mathcal{K}(\eta)$*. The principal is willing to engage in any number of sessions of the key exchange protocol as both initiator and responder, with any other principal in the system.*

  – Send a message m *to a protocol session: the receiving party processes* m *and returns to* $\mathcal{A}_e$ *the answers it computes according to the protocol.*

  – At some point $\mathcal{A}_e$ *finishes its execution and outputs* (sid, St)*, that is a session identifier and some state information.*

- *Adversary* $\mathcal{A}_c$ *is given the state information* St *and is provided access to a left-right encryption oracle* $\mathcal{E}(LR(\cdot,\cdot,b),k)$ *keyed with the key* k *locally output in session* sid*.*

- *Adversary* $\mathcal{A}_c$ *plays the standard IND-CPA game: it submits pairs of equal-length messages* (m_0, m_1) *to the encryption oracle and obtains in return* $\mathcal{E}(m_b, k)$*.*

- *At some point* $\mathcal{A}_c$ *finishes its execution and outputs a guess bit* d*, which is also the output of the experiment.*

For any adversary $\mathcal{A} = (\mathcal{A}_e, \mathcal{A}_c)$ *we define its advantage as:*

$$\mathbf{Adv}^{ke}_{\mathcal{A},\Sigma,\Pi}(\eta) = Pr[\mathbf{Exp}^{ke_1}_{\mathcal{A},\Sigma,S}(\eta) = 1] - Pr[\mathbf{Exp}^{ke_0}_{\mathcal{A},\Sigma,S}(\eta) = 1]$$

and say that keys exchanged through Σ *are usable in schemes in* S *if for any* $\Pi \in S$ *the advantage of any probabilistic, polynomial-time adversary* $\mathcal{A}$ *is negligible.*

The definition can be easily modified to define a similar usability property of keys for other primitives, for example, message authentication codes, by appropriately changing the security game that is played in the second phase.

The above definition of usability is consistent with accepted definitions of symmetric key-based primitives based on security against adversaries that are allowed arbitrary uses of the primitive in a priori unknown settings. In addition, our model considers the possibility that key generation is accomplished using a key exchange protocol instead of a non-interactive algorithm. The adversary is provided with auxiliary information obtained by interacting with this protocol.

4.1.3 Logical Formalization

Key exchange The game described in Section 4.1.2 is used to define the semantics of a "Goodkey" predicate and to provide the basis for computational proofs of the soundness of formal axioms using this predicate. In fact, the basic predicate of the logic is GoodKeyAgainst, which asserts that a key is good against a specific agent; the truth of this predicate at any point in the run of one or more protocols will depend on the additional information available to that agent from observations about the protocol steps and actions of any protocol adversary. In logical proofs involving key exchange protocols and their use, we use a derived predicate SharedKey, which asserts that a key is good against any agent not among those sharing the key. (The agents who share the key are arguments to the predicate.)

Secure sessions Formulas involving SharedKey are also used to reason about protocols that use a key generated by a key exchange protocol. A key obtained by running a key exchange protocol may be used to encrypt and authenticate subsequent communication in what is commonly referred to as a *secure session* or *secure channel*. As an example, we will use a formula involving SharedKey as a precondition to a proof of security of a simple one-message protocol in which the sender encrypts some data using a symmetric encryption scheme. Such a session provides *secrecy* if, assuming the sender flips a coin and sends one of two known messages m_0 or m_1 depending on the outcome, the attacker's probability of determining which message was sent is close

to 1/2. We express these additional probabilistic properties using other predicates of Computational PCL. While secure sessions typically provide integrity and replay protection guarantees, in this paper, we only focus on the secrecy property.

Composition We obtain a security proof of the composition of key exchange with secure sessions using a general composition theorem [36]. In carrying out the composition, we ensure that the two protocols do not degrade each other's security guarantees. Technically, this is accomplished by checking that each component respects the invariants of the other. It is important to notice the distinction between the "conditional" composition used in this logic and "universal" composition approaches [24, 10]. Specifically, the PCL composition theorems only apply when a protocol is composed with protocol steps that respect specified invariants. No guarantees are expressed or implied for composition with protocols that violate invariants that are needed.

Because of the requirements on composition, some provable properties may only hold under relatively stringent conditions. In our framework, it may be possible to prove security properties of a key exchange protocol, assuming certain invariants, even if the key exchange protocol is augmented to provide a decryption oracle after one party has completed the protocol. However, the condition given in Definition 4.1.1 must be understood as only guaranteeing usefulness of the key material itself, apart from some aspects of interaction between the key exchange protocol and protocols that use the key. In our framework, the way the key is used is handled by the composition theorems. In particular, a key exchange protocol that provides a decryption oracle will generally not satisfy the requirements of the composition theorem since the subsequent key usage protocol will typically require that no such mechanisms are present.

4.2 Proof System

A first step towards developing a proof system faithful to the complexity-theoretic semantics is given in Section 3.3. In this section, we describe the predicates and axioms for reasoning about Diffie-Hellman, symmetric encryption, and signature primitives introduced in this paper. The soundness theorem for the extended proof system is

in Section 4.3. We reiterate that the advantage of using the proof system is that its justification using cryptographic-style arguments is a one-time mathematical effort; protocol proofs can be carried out symbolically using the proof system without explicitly reasoning about probability and complexity.

Diffie-Hellman key exchange: Reasoning about Diffie-Hellman key exchange can be divided into two steps. First we reason about symbolic actions of honest participants; then we deduce computational key secrecy properties from the fact that honest principals follow certain rules when dealing with Diffie-Hellman key material.

The DHSource predicate is used to reason about the source of a piece of information, such as a nonce. Intuitively, the formula $\mathsf{DHSource}(X, x)$ means that the thread X created the nonce x, and in all subsequent actions of that thread it appears only in `expg` and `dhkeyken` actions. In other words, $\mathsf{DHSource}(X, x)$ holds if a thread only uses x "inside" exponential g^x or a key $k = KeyGen(PRF(y^x))$.

We extend the proof system with the following axioms used for reasoning about the DHSource axiom.

$$\begin{array}{ll} \textbf{S0} & \top\ [\texttt{new}\ x]_X\ \mathsf{DHSource}(X, x) \\ \textbf{S1} & \mathsf{DHSource}(X, x)[\mathsf{a}]_X\ \mathsf{DHSource}(X, x) \\ & \text{where } (x \not\sqsubseteq a \text{ or } \mathsf{a} = \texttt{expg}\ x \\ & \text{or } \mathsf{a} = \texttt{dhkeyken}\ y, x \text{ and } x \not\sqsubseteq y) \end{array}$$

Axioms **S0** and **S1** model introduction and persistence of the DHSource predicate. Informally, after a thread X creates a new nonce x, $\mathsf{DHSource}(X, x)$ holds (axiom **S0**), and it continues to hold (axiom **S1**) as long as the thread does not use x, other than creating a public key g^x, and creating and using a shared key $v = y^x$. Axioms **S0** and **S1** capture simple properties about information flow within a program and we prove their soundness using direct arguments. When we use these axioms in a formal proof, we are essentially performing induction over symbolic actions of honest parties proving that they treat Diffie-Hellman exponents in a correct way.

The result of a good key exchange protocol is a shared key k which is indistinguishable from a randomly chosen key by a polynomial-time attacker. As discussed in the introduction, after the key is used (e.g. to establish a secure session), partial information about the key is revealed to the attacker. In our model we capture the quality of a key using the predicate $\mathsf{GoodKeyAgainst}$. Informally we wish to capture the property that the key can be securely used for encryption, even if the attacker has some partial information about the key. A bit more formally, $\mathsf{GoodKeyAgainst}(X, k)$ holds whenever no probabilistic polynomial-time algorithm, given X's view of the run, can win the IND-CPA game if the challenger uses key k instead of a key generated using the key generation algorithm. We often use the shorthand $\mathsf{SharedKey}(A, B, k)$ to denote, that k is a good key against everyone except A and B. More precisely $\mathsf{SharedKey}(A, B, k) \equiv \forall X(X = A \vee X = B \vee \mathsf{GoodKeyAgainst}(X, k))$.

We extend the proof system with the following axiom used for establishing key secrecy in Diffie-Hellman key exchange.

$$\textbf{DH} \quad \mathsf{Honest}(\hat{X}, \hat{Y}) \wedge \mathsf{DHSource}(X, x) \wedge \mathsf{DHSource}(Y, y) \\ \wedge \mathsf{KeyGen}(X, k, x, g^y) \implies \mathsf{SharedKey}(X, Y, k)$$

Axiom **DH** says that if two threads of honest agents X and Y use their respective private key in a safe way, then the shared key generated from g^{xy} can be safely used with any IND-CPA secure encryption scheme. Note that this axiom does not capture key agreement: threads X and Y could in principle have different keys (or no keys at all). Key agrement has to be established by other methods. We prove the soundness of axiom **DH** using a cryptographic-style reduction to the security of the underlying IND-CPA secure encryption scheme and to the Decisional Diffie-Hellman assumption. The proof employs a hybrid argument and is sketched in Section 4.3.

Signatures The signature axiom is given below.

$$\textbf{SIG} \quad \mathsf{Verify}(X, m, \hat{Y}) \wedge \mathsf{Honest}(\hat{X}, \hat{Y}) \implies \exists Y.\mathsf{Sign}(Y, m)$$

Informally, this axiom says that if a thread $\hat{X}$ performs a successful signature verification step using a public key of an honest party $\hat{Y}$ then there has to be a thread Y of agent $\hat{Y}$ that performed the signature operation on the same message. This axiom captures unforgeability of signatures and its soundness is proved by reduction to the CMA-security game for signatures. A complete proof will appear in the full version of this paper.

Symmetric Encryption In the symbolic model [32, 36], the predicate Has states that a principal can "derive" a message or its contents from the information gathered during protocol execution. Since secrecy in the computational model involves absence of any partial information, we use the predicate $\mathsf{Indist}(X, t)$ to state that no probabilistic polynomial-time algorithm, given X's view of the run, can distinguish the actual bitstring corresponding to the term t from a random bitstring chosen from the same distribution.

The $\mathsf{PSource}(X, b, m, k)$ predicate means that the bit b and the message m were chosen by X via a `pick` action, and in all subsequent actions of that thread b does not appear, and m appears only inside encryptions with the key k. We use it for expressing security properties of symmetric encryption schemes and reasoning about protocols which use such schemes.

We extend the proof system with the following axioms used for reasoning about

symmetric encryption.

$$
\begin{array}{ll}
\textbf{PS0} & \top\ [(m,b) = \texttt{pick}\ m_0, m_1]_X\ \mathsf{PSource}(X,b,m,k) \\
\textbf{PS1} & \mathsf{PSource}(X,b,m,k)[\mathtt{a}]_X\ \mathsf{PSource}(X,b,m,k) \\
 & (m,b \not\sqsubseteq \mathtt{a} \text{ or } \mathtt{a} = \texttt{enc}\ m,k) \\
\textbf{ENC} & \mathsf{PSource}(X,b,m,k) \wedge \mathsf{Honest}(\hat{X},\hat{Y}) \wedge \\
 & \mathsf{SharedKey}(X,Y,k) \wedge \\
 & (\mathsf{Decrypts}(Y,m,k) \wedge \mathsf{Contains}(m,m') \wedge \\
 & \mathsf{Send}(Y,m'') \supset \neg\mathsf{Contains}(m'',m')) \wedge \\
 & (\mathsf{Decrypts}(X,m,k) \wedge \mathsf{Contains}(m,m') \wedge \\
 & \mathsf{Send}(X,m'') \supset \neg\mathsf{Contains}(m'',m')) \wedge \\
 & \wedge Z \neq X \wedge Z \neq Y \implies \mathsf{Indist}(Z,b)
\end{array}
$$

Axiom **ENC** captures the properties of an IND-CPA encryption scheme. Informally, in a scenario where k is a shared key between threads X and Y, if X chooses a message m and the bit b via a `pick` action, and both threads follow the rules of the IND-CPA game (i.e. the do not send parts of messages they decrypt) then the bit b should be indistinguishable from a random bit to any other party.

Discussion The presented axioms deduce computational properties based on symbolic actions executed by individual honest parties. This resembles the setup in defining security properties of cryptographic primitives using games. For example, in the IND-CPA game the challenger is required to generate a random key and use it for encryption only. If this syntactic property is satisfied, then the security condition (semantic security) is guaranteed to hold for all computational adversaries interacting with the challenger. The predicates DHSource and PSource are used to exactly state symbolic constraints for actions of honest parties. The axioms **DH** and **ENC** bridge the gap between the symbolic and computational world and are proven sound by reduction to security properties of corresponding primitives.

In Section 4.1.1 we pointed out that the key indistinguishability property is not

invariant under composition. Specifically, focusing on the Diffie-Hellman example, we could have formulated the **DH** axiom to guarantee key indistinguishability by modifying the DHSource predicate to preclude the case where the resulting secret is used as a key. The resulting axiom could be proven sound by a similar reduction. However, this axiom will not be useful in a proof involving a composition of a key exchange protocol with a protocol that uses a key.

4.3 Computational Semantics and Soundness Theorem

In this section, we outline the main ideas behind the computational semantics and present semantics for the predicates introduced in this paper. We also state the soundness theorem for the proof system and sketch the proof for one representative axiom making a connection between validity of logical formulas and standard security definitions of cryptographic primitives. A complete presentation of the computational semantics is contained in [37].

The meaning of a formula is defined with respect to a set of computational traces, where each trace corresponds to one particular execution of the protocol with all the parameters fixed (including the randomness of the attacker and honest parties). Intuitively, the meaning of a formula φ on a set T of computational traces is a subset $T' \subseteq T$ that respects φ in some specific way. For example, an action predicate such as Send selects a set of traces in which a send occurs. The semantics of predicates Indist and GoodKeyAgainst are more complex and involve a second phase of execution where the distinguisher tries to guess the secret value or break the encryption scheme.

We inductively define the semantics $[\![\, \varphi \,]\!]\,(T, D, \epsilon)$ of a formula φ on the set T of traces, with distinguisher D and tolerance ϵ. The distinguisher and tolerance are not used in any of the clauses except for Indist and GoodKeyAgainst, where they are used to determine whether the distinguisher has more than a negligible chance of distinguishing the given value from a random value or wining an IND-CPA game,

respectively. A protocol Q will satisfy a formula φ, written $Q \models \varphi$ if for all adversaries, distinguishers, and sufficiently large security parameter, $[\![\varphi]\!] (T, D, \epsilon)$ is an overwhelming subset of the set of all possible traces produced by the interaction of protocol Q and attacker A.

Every trace $t \in T$ includes a set of symbolic actions executed by the honest participants, as well as the mapping λ assigning bitstrings to all terms appearing in symbolic actions. A trace t also includes a mapping σ assigning bitstrings to free formula variables. Informally, σ represents the environment in which the formula is evaluated. Technically, as the binding operators (such as quantifiers) are parsed, σ is used to keep track of the values assigned to the variables by these operators.

- $[\![\mathsf{DHSource}(X, x)]\!] (T, D, \epsilon)$ is the collection of all traces $t \in T$ such that for all basic terms y with $\sigma(x) = \lambda(y)$ there is a single symbolic action $(\texttt{new}\ (X), y)$, and term y does not appear in any symbolic actions except maybe in $(v := \texttt{expg}\ (X, y))$ and $(v := \texttt{dhkeyken}\ (X, z, y))$ for some term z different from y.

  Notice that the condition $\sigma(x) = \lambda(y)$ is used to tie the formula variable x to a trace variable y by requiring that they both evaluate to the same bitstring. Therefore, if we fix a particular trace $t \in T$ with an environment σ, then $t \in [\![\mathsf{DHSource}(X, x)]\!] (T, D, \epsilon)$ if in the trace t, the thread X created a new nonce (represented by a trace variable y) with a bitstring value equal to that of $\sigma(x)$, and used the variable y only inside an exponentiation action or a key generation action.

- $[\![\mathsf{PSource}(X, b, m, k)]\!] (T, D, \epsilon)$ is the collection of all traces $t \in T$ such that for all basic terms m', b' with $\sigma(m) = \lambda(m')$ and $\sigma(b) = \lambda(b')$, such that there is symbolic action $((m', b') := \texttt{pick}\ X, m_1, m_2)$, terms b' and m' do not appear in any symbolic actions except maybe in $(v := \texttt{enc}\ X, m', k')$, with $\sigma(k) = \lambda(k')$.

- $[\![\mathsf{GoodKeyAgainst}(X, k)]\!] (T, D, \epsilon)$ is the complete set of traces T if the distinguisher D, who is given a complete X's view of the run, has an advantage greater than ϵ in winning the IND-CPA game against a challenger using the

bitstring corresponding to term k, and empty set $\emptyset$ otherwise. Here the probability is taken by choosing an uniformly random trace $t \in T$ (which includes the randomness of all parties, the attacker as well as the distinguisher randomness).

- $[\![\mathsf{Sign}(X, m)]\!] (T, D, \epsilon)$ is a collection of all traces where X performs a symbolic signing operation of on a variable whose bitstring value corresponds to the bitstring value of m.
- $\left|\left[\mathsf{Verify}(\hat{X}, m, \hat{Y}) \right]\right| (T, D, \epsilon)$ is a collection of all traces where X performs a successful symbolic signature verification operation where the bitstring value of the signed text corresponds to the bitstring value of m, and the bitstring value of the agent name corresponds to the bitstring value of $\hat{Y}$.

Note that the predicates defined by reference to a set of symbolic actions by a thread X only make sense if the agent $\hat{X}$ is honest and therefore its threads only performing symbolic actions. For the threads of dishonest agents, we can define the semantics of these terms arbitrarily. In all provable formulas, these predicates will be always used in conjunction with the assumption that the corresponding agent is honest.

The proof system used in this paper consists of axioms and proof rules presented in [37] extended with the axioms introduced in Section 4.2: **S0**, **S1** and **SH** modelling properties of the Diffie-Hellman key exchange, **SIG** modelling signatures and **PS0**, **PS1** and **ENC** modellig symmetric encryption.

Theorem 4.3.1 (Soundness)**.** *The proof system [37] extended with axioms from Section 4.2 is sound for the semantics [37] extended with clauses above.*

This soundness theorem is proved by showing that every axiom is a valid formula and that all proof rules preserve validity. For some axioms and proof rules soundness will follow directly from the execution model or by information theoretic reasoning. Axioms stating properties of cryptographic primitives are proved sound by transforming a protocol and an attacker breaking the axiom to an attacker on the game defining the security property of the cryptographic primitive. Below we give a proof sketch

for the soundness of the **DH** and **SIG** axioms (introduced and discussed informally in Section 4.2).

Proof sketch for axiom **DH**. Assuming the axiom is false we deduce that there exists an adversary A, and a distinguisher D such that:

$$|[\![\, \neg\varphi \,]\!]\,(T, D, \nu(\eta))| \geq \nu(\eta)$$

is non-negligible (as a function of η). By unwinding the semantics of $\neg\varphi$, we obtain that there exists three parties b_X, b_Y and b_Z such that the set:

$$[\![\, \neg\varphi \,]\!]\,(T, D, \nu(\eta))[X \to b_X][Y \to b_Y][Z \to b_Z]$$

is of non-negligible size in rapport with $|T|$. More precisely, with non-negligible probability, the distinguisher D can successfully break the IND-CPA game played against a standard left-right encryption oracle keyed with the key that party b_X outputs at the end of the protocol execution with party b_Y, provided that D has access to the view associated to party b_Z. Given adversary A and distinguisher D we explain how to construct two adversaries A_1 and A_2, one against the DDH assumption and the other one against the IND-CPA security of the encryption scheme, such that at least one of these two adversaries has non-negligible probability of wining the corresponding security game.

We consider two execution scenarios that involve A and D. In the first scenario adversary A interacts with the protocol as sketched in Section 4.1. Then, D plays the IND-CPA game in which the key for encryption is the key obtained by party b_X. At the end of the execution D outputs a guess bit d. Let $\mathsf{guess}_1(A, D)$ be the event that the output of D coincides with the bit b of the left-right oracle to which it has access. By the assumption that D is successful we conclude that $P_1 = \Pr[\mathsf{guess}(A, D)_1]$ is non-negligible. (Here, to simplify notation we omit to explicitly show the dependence of the event on the security parameter.)

In the second scenario, the execution of A proceeds also as sketched in Section 4.1. However, D plays the IND-CPA games against encryption oracles in which the key has

been randomly generated (in particular, this key is independent from the execution of the protocol). Let $\mathsf{guess}_2(A, D)$ be the event that D correctly guesses the bit that parameterizes the left-right oracle, and let $P_2 = \Pr[\mathsf{guess}_2(A, D) = b]$.

Intuitively, if the DDH assumption holds, adversary D should not observe any difference between the two different execution scenarios that we consider. Formally, we construct the following adversary A_1 against the DDH assumption. The adversary takes as input a triple $(X = g^x, Y = g^y, Z = g^z)$ and works as follows. It executes adversary A as a subroutine, and emulates for A the behavior of the honest parties. The difference is that for parties b_X and b_Y, the adversary does not generates the values x and y needed for the execution, but whenever it needs to send g^x and g^y it sends X and Y respectively. Notice that here we crucially use that $\mathsf{DHSource}(\hat{X}, x)$ and $\mathsf{DHSource}(\hat{Y}, y)$ hold, since this implies that parties b_X and b_Y only send the values x and y as exponents. (Otherwise, it would be impossible to carry out the simulation).

When A finishes its execution, adversary A_1 flips a bit b and simulates for D the left-right encryption oracle parameterized by b and keyed by the key generated from Z. When D finishes and outputs a bit d adversary A outputs 1 if $d = b$ and 0 otherwise.

Notice that when (X, Y, Z) are such that $z = xy$, the view of (A, D) is as in the normal execution of the protocol, and thus we have that $\Pr[A_1 = 1 | Z = g^{xy}] = \Pr[\mathsf{guess}_1(A, D)]$. When Z is such that z is randomly chosen, the view of (A, D) is as in the alternative execution scenario that we consider, and thus we obtain that $\Pr[A_1 = 1 | Z = g^z] = \Pr[\mathsf{guess}_2(A, D)]$.

The advantage that A_1 has in breaking the DDH assumption is thus:

$$\mathbf{Adv}_{\mathrm{DDH},A_1}(\eta) = \Pr[\mathsf{guess}_1(A, D)] - \Pr[\mathsf{guess}_2(A, D)] \tag{4.1}$$

Next, we bound the probability of $\mathsf{guess}_2(A, D)$. Intuitively, if the encryption scheme is IND-CPA secure, no adversary should be able to win the IND-CPA game in the second execution scenario (since the key used in the oracle is a randomly generated key, thus independent from that generated in the key exchange phase).

Formally, we construct adversary A_2 against the IND-CPA security of the encryption scheme. The adversary has access to a left-right encryption oracle parameterized by a bit b and proceeds as follows. It runs adversary A as a subroutine and simulates for A the execution of the honest parties involved in the protocol. Thus, it generates the encryption and decryption keys of the honest parties and then receives and outputs messages as prescribed by the protocol. Whenever A finishes its execution, adversary A_2 provides to D the view of party b_Z, whatever state information A has output, and offers access to his own oracle (parameterized by a bit b to be guessed). Notice that if at any point, in order to carry out the simulation adversary A needs to output the encryption of some message m under the key k of the oracle (this is the case for example when the parties exchange confirmation messages using the exchanged key), A_2 can simply submit (m, m) to its encryption oracle.

The guess of A is whatever D outputs. The key observation is that the view of the pair (A, D) is exactly as in the second execution scenario that we consider. Thus A_2 successfully guesses the bit b precisely when, following the execution we have just described, the distinguisher D outputs b. Thus, we obtain that:

$$\begin{aligned}&\mathbf{Adv}_{\text{IND-CPA},A_2}(\eta) = \\ &\Pr[A_2 \text{ wins the IND-CPA game}] = \Pr[\mathsf{guess}_2(A, D)]\end{aligned} \tag{4.2}$$

By Equations (4.1) and (4.2) we get that:

$$\Pr[\mathsf{guess}_1(A, D)] = \mathbf{Adv}_{\text{DDH},A_1}(\eta) + \mathbf{Adv}_{\text{IND-CPA},A_2}(\eta)$$

Since the left-hand side term is a non-negligible function so is at least one of the summands on the right-hand side. Thus, either the DDH assumption is not valid, or the encryption scheme is not IND-CPA secure. □

Proof sketch for axiom **SIG**. Intuitively, if this axiom is not satisfied, there is a protocol and an adversary such that in a non-negligible fraction of traces where participant $\hat{Y}$ successfully verifies a signature on some message m by honest participant $\hat{X}$, yet this signature has not been produced by $\hat{X}$. This contradicts the security of the

signature scheme.

Assume that there exists some adversary A such that for any ν:

$$|\,[\![\,\varphi\,]\!]\,(T, D, \nu(\eta))\,|\;/\;|\,T\,|\;\leq\;1-\nu(\eta) \tag{4.3}$$

for infinitely many security parameters η. We construct an adversary B against the digital signature scheme $\Sigma = (\mathcal{K}, \mathcal{S}, \mathcal{V})$. Recall that since B is against Σ, it has access to a signing oracle under some honestly generated key sk and takes as input the corresponding verification key vk. Adversary B works as follows. First, it selects two random identities a and b and then it emulates the execution of A against possible protocol participants. Adversary B plays the roles of all parties involved in the protocol, and in particular it generates their signing/verification keys. The only exception is party b for which the associated public key is set to pk. Notice that B can faithfully carry out the simulation. Signatures of b are produced using the signing oracle to which B has access; signatures of all other parties are calculated using the corresponding signing keys.

During the execution, whenever party a needs to verify a signature σ on some message m, allegedly created by b, it verifies if it has queried m to its oracle. If this is not the case, then B outputs an attempted forgery (m, σ).

It remains to show that the success probability of B is non-negligible. From Equation 4.3 we conclude that A is such that

$$|[\![\,\neg\varphi\,]\!]\,(T, D, \nu(\eta))| \geq \nu(\eta)$$

for infinitely many security parameter η. For any pair of session identifier names b_X and b_Y we write T_{b_X,b_Y} for the set $T[X \to b_X][Y \to b_Y]$.

Since $[\![\,\neg\varphi\,]\!]\,(T, D, \nu(\eta))$ is the intersection of the sets $[\![\,\neg\varphi\,]\!]\,T[X \to b_X][Y \to b_Y]$ for all possible pairs of identifiers (b_X, b_Y) and there are polynomially many such identifiers, we deduce that there exists one such pair (a, b) for which the set $[\![\,\neg\varphi\,]\!]\,T_{(a,b)}$ has non-negligible size (in rapport with $|T|$).

Spelling out the above, we obtain that adversary A satisfies the following: with non-negligible probability it has an execution where X is mapped to a, Y is mapped

$$
\begin{array}{lcl}
\textbf{Init}(Y) & \equiv & [\\
 & & \texttt{new}\ x; \\
 & & gx := \texttt{expg}\ x; \\
 & & \texttt{send}\ \hat{X}, \hat{Y}, gx; \\
 & & \texttt{receive}\ \hat{Y}, \hat{X}, z, s; \\
 & & \texttt{verify}\ s, (z, gx, \hat{X}), \hat{Y}; \\
 & & r := \texttt{sign}\ (gx, z, \hat{Y}), \hat{X}; \\
 & & \texttt{send}\ \hat{X}, \hat{Y}, r; \\
 & & k := \texttt{dhkeyken}\ x, z; \\
]_X & &
\end{array}
\qquad
\begin{array}{lcl}
\textbf{Resp} & \equiv & [\\
 & & \texttt{receive}\ \hat{X}, \hat{Y}, w; \\
 & & \texttt{new}\ y; \\
 & & gy := \texttt{expg}\ y; \\
 & & r := \texttt{sign}\ (gy, w, \hat{X}), \hat{Y}; \\
 & & \texttt{send}\ \hat{Y}, \hat{X}, gy, r; \\
 & & \texttt{receive}\ \hat{X}, \hat{Y}, t; \\
 & & \texttt{verify}\ t, (w, gy, \hat{Y}), \hat{X}; \\
 & & k := \texttt{dhkeyken}\ y, w; \\
]_Y & &
\end{array}
$$

Figure 4.1: Roles of the ISO-9798-3 protocol

to b, both parties a and b are honest and party b verifies some signature σ on some message m which was not signed by a. Since the view of A, when executed as a subroutine by B, is precisely as in its normal execution against the protocol, with non-negligible probability at some point party a would need to verify a signature σ on some message m which was not produced by b. When B selects $b_X = a$ and $b_Y = b$ (event which happens with non-negligible probability), B outputs a successful forgery since (m, σ) since B does not query m to its oracle. □

4.4 Examples

4.4.1 Key Exchange

In this section, we use the protocol logic to formally prove a property of the ISO 9798-3 protocol. The ISO 9793-3 protocol is defined by a set of two roles $Q_{ISO} = \{\textbf{Init}, \textbf{Resp}\}$, comprising one role **Init** for the initiator of the protocol and one program **Resp** for the responder. The roles of the protocol, written using the protocol language are given in Figure 2.3.

Writing $\textbf{Init} = \textbf{Init}(Y)$ for the protocol steps of initiator $\hat{X}$ communicating with responder $\hat{Y}$, the security guarantee for the initiator is expressed by the following

S0	$\top\ [\texttt{new}\ x;]_X\ \mathsf{DHSource}(X, x)$	(4.4)
S1, (4.4)	$\top\ [\mathbf{Init}]_X\ \mathsf{DHSource}(X, x)$	(4.5)
AA1	$\top\ [\texttt{verify}\ s, (z, gx, \hat{X}), \hat{Y};]_X\ \mathsf{Verify}(X, (z, gx, \hat{X}), \hat{Y})$	(4.6)
P, **SEQ**, (4.6)	$\top\ [\mathbf{Init}]_X\ \mathsf{Verify}(X, (z, gx, \hat{X}), \hat{Y})$	(4.7)
SIG, (4.7)	$\top\ [\mathbf{Init}]_X\ \exists Y.\mathsf{Sign}(Y, (z, gx, \hat{X}))$	(4.8)
$\mathbf{HON_{ISO}}$	$\mathsf{Honest}(\hat{Y}) \wedge \mathsf{Sign}(Y, (z, gx, \hat{X})) \supset \exists y.\ z = g^y \wedge \mathsf{DHSource}(Y, y)$	(4.9)
(4.10)	$\top\ [\mathbf{Init}]_X\ \mathsf{Honest}(\hat{X}, \hat{Y}) \supset \exists Y.\exists y.\ z = g^y \wedge \mathsf{DHSource}(Y, y)$	(4.10)
(4.11), (4.5)	$\top\ [\mathbf{Init}]_X\ \mathsf{Honest}(\hat{X}, \hat{Y}) \supset \exists Y.\exists y.\ z = g^y \wedge \mathsf{DHSource}(X, x) \wedge \mathsf{DHSource}(Y, y)$	(4.11)
(4.12), **DH**	$\top\ [\mathbf{Init}]_X\ \mathsf{Honest}(\hat{X}, \hat{Y}) \supset \exists Y.\exists y.\ z = g^y \wedge \mathsf{SharedKey}(X, Y, k)$	(4.12)

Table 4.1: Secrecy proof for the initiator in the ISO-9798-3 Protocol

formula:

$$Q_{ISO} \ \vdash\ \top\ [\mathbf{Init}]_X\ \mathsf{Honest}(\hat{X}, \hat{Y}) \supset \exists Y.\exists y.\ z = g^y \wedge \mathsf{SharedKey}(X, Y, k)$$

This formula states the key usability property for the initiator when ISO-9798-3 protocol is run in isolation. More precisely, after the initiator $\hat{X}$ completes the initiator steps with $\hat{Y}$ then, assuming both agents are honest in all of their threads, there is one thread Y of agent $\hat{Y}$ that has established a shared key with $\hat{X}$. The meaning of the predicate SharedKey in this formula is defined using the game condition explained in Section 4.1. Note that this property is guaranteed against any polynomial time adversary when any number of sessions are executed concurrently. The only additional assumption is that honest agents follow roles of the ISO-9798-3 protocol.

The formal proof, given in Table 4.1, illustrates some general properties of our method. This example proof is modular, consisting of three distinct parts. In the first part of the proof, steps (1)-(2), we establish that value x generated by the

initiator is only used to create g^x and the final key g^{xy}. The reasoning in this step is non-cryptographic, and only relies on the structure of the program of the initiator. In the second step the analog property for y is established based on the security of the signature scheme, and the structure of the responding program. Finally, the axiom that captures the DDH assumption is used to conclude that the key derived from g^{xy} is secure. Notice that the axioms **SIG** and **DH** are used independently to establish different security properties. The two properties are only combined in the last step of the proof.

The modular symbolic proof can be compared with conventional computational arguments, such as the computational proof of the same property by reduction. The reduction proof starts from an adversary against the key derived from g^{xy} and constructs two different adversaries, one against the DDH assumption and the other one against the signature scheme. The assumptions on signatures and DDH are used intertwined. Specifically, to argue that the adversary against DDH is successful, it is necessary to argue that the values that occur in a possible simulation are created by the honest party, and consequently, to argue that the signature scheme is secure. More generally, the analysis of protocols that use more primitives, and where the reduction uses multiple adversaries, the proofs become increasingly complex. In contrast, evidence drawn from work with the symbolic version of the logic indicates that axiomatic proofs are at the same level of complexity as our proof for the ISO-9798-3 protocol [1].

4.4.2 Secure Sessions

We formalize the definition of secure sessions presented in Section 4.1.3 for a protocol $Q_{SS} = \{\mathbf{InitS}, \mathbf{RespS}\}$ below.

$$\begin{aligned} Q_{SS} \quad \vdash \quad & [(m, b) = \texttt{pick}\ m_0, m_1; \mathbf{InitS}(\mathbf{Y}, \mathbf{m})]_X \\ & \mathsf{Honest}(\hat{X}, \hat{Y}) \wedge Z \neq X \wedge Z \neq Y \implies \mathsf{Indist}(Z, b) \end{aligned}$$

In words, this formula states that if initiator $\hat{X}$ picks one of two messages at random and executes the secure sessions protocol with $\hat{Y}$, then the attacker cannot distinguish,

which of the two messages was transmitted.

As a concrete example, we consider the secure session protocol with the following initiator program. The responder simply decrypts the message.

$$\mathbf{InitS}(\hat{Y}, m, k) \equiv \left[e := \texttt{enc}\ m, k; \texttt{send}\ \hat{Y}, e;\right]_X$$

Using the proof system we can prove that this protocol provides the secure-session property between threads X and Y assuming that the key k is a shared key between X and Y, formally:

$$\begin{aligned}&\mathsf{SharedKey}(X, Y, k)\\ &[(m, b) = \texttt{pick}\ m_0, m_1; \mathbf{InitS}(\mathbf{Y}, \mathbf{m}, \mathbf{k})]_X\\ &\mathsf{Honest}(\hat{X}, \hat{Y}) \wedge Z \neq X \wedge Z \neq Y \implies \mathsf{Indist}(Z, b)\end{aligned}$$

The security property of an IND-CPA secure encryption scheme (expressed by axiom **ENC**) is central to this proof. This is a point of difference between the logic and the approaches which relate the symbolic and computational models [77, 10] and require stronger cryptographic assumptions such as IND-CCA-2.

4.4.3 Composition

We prove that the sequential composition of ISO-9798-3 and the secure sessions protocol described above is also a good secure session protocol, when the key generated in the first part is used in the second part. We use the general sequential theorem of [36]. This involves two steps: a) the property guaranteed by ISO (that k is a shared key between X and Y) is precisely the assumption of the secure sessions protocol b) two protocols satisfy each other's invariants (e.g. line (6) of Table 4.1). This step guarantees that one protocol does not provide an oracle that can be used to break the security of the other protocol (see [36] for further elaboration and examples of composition). The composition would not have worked if we used key indistinguishability instead of the weaker shared-key property.

Consider a protocol in which both parties execute session of the Q_{ISO} protocol followed by a session of protocol Q_{SS} protocol. Moreover, key used in the secure

session protocol is exactly the key established in the key-exchange phase, while the message transmitted in the secure session phase in an arbitrary message obtained via input interface of a resulting protocol. Formally, $Q = \{\mathbf{InitQ}, \mathbf{RespQ}\}$ where the initiators program is given by (responders program is analogous):

$$\mathbf{InitQ}(\hat{Y}, m) \equiv [\mathbf{Init}(\hat{Y}); \mathbf{InitS}(\hat{Y}, k, m)]_X$$

To formally prove that the combined protocol also provides secret session property we use composition theorems and the sequencing rules in steps as follows: First we need to ensure that protocols satisfy each others invariants, i.e. $Q_{SS} \vdash \Gamma_{ISO}$ and $Q_{ISO} \vdash \Gamma_{SS}$, where Γ_{ISO} and Γ_{SS} are invariants used in the proof of the key exchange and the secure session protocol respectively. Informally, Γ_{ISO} requires that honest agents treat their Diffie-Hellman exponents correctly, while Γ_{SS} requires that no party possessing the shared key acts as a decryption oracle.

$$\begin{aligned}
\Gamma_{ISO} &= \mathsf{Honest}(\hat{Y}) \wedge \mathsf{Sign}(Y, (z, gx, \hat{X})) \supset \\
&\quad \exists y.\ z = g^y \wedge \mathsf{DHSource}(Y, y) \\
\Gamma_{SS} &= \mathsf{Honest}(\hat{X}, \hat{Y}) \wedge \mathsf{SharedKey}(X, Y, k) \wedge \\
&\quad (\mathsf{Decrypts}(Y, m, k) \wedge \mathsf{Contains}(m, m') \wedge \\
&\quad \mathsf{Send}(Y, m'') \supset \neg\mathsf{Contains}(m'', m')) \\
&\quad (\mathsf{Decrypts}(X, m, k) \wedge \mathsf{Contains}(m, m') \wedge \\
&\quad \mathsf{Send}(X, m'') \supset \neg\mathsf{Contains}(m'', m'))
\end{aligned}$$

When this is established, both proofs are still valid, using the sequencing rule and the fact that the Q_{SS} protocol does not invalidate DHSource predicate we establish that the shared-key property persists from the end of the Q_{ISO} protocol to the end of the Q_{SS} protocol. Finally using the properties of the Q_{SS} protocol, we establish that the combined protocol provides secure session property.

As mentioned in Section 4.1, there are key exchange protocols which satisfy the key usability property but may not compose well with any subsequent secure session. Using the same example, if a key exchange protocol provides a decryption oracle then

Γ_{SS} will not be satisfied and the protocol composition will fail.

Chapter 5

Other Results

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

5.1 PCL Proof Methods

In [34], 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 dissertation focuses on authentication proofs, we have also developed a proof method for establishing secrecy properties [90]. 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" [93] and

work using the strand space execution model [97], 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.

5.2 PCL Applications

PCL has been used to analyze a number of industrial security protocols including the IEEE 802.11i wireless LAN security standard [57] (of which SSL/TLS is a component), Kerberos V5 [90], and the IETF GDOI standard for secure group communication [75].

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 [56]. 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 [56] 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. [98]).

Kerberos [63] 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 [90], 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 [30] 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 [13], and non-formal mathematical proofs have been developed for other abstractions of Kerberos [23], this is the first concise formal logic proof of secrecy and authentication for Kerberos and PKINIT.

Chapter 6

Related Work

6.1 Protocol Composition Logic

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 [21, 96, 52], process calculi [3, 2, 64, 87] and tools [73, 95], as well as theorem-proving [85, 84] and model-checking methods [65, 79, 88, 92] using general purpose tools.

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 [21], 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 [82], Dolev-Yao [41], and much subsequent work by others.

PCL shares several features with BAN [21], 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 [85]. 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.

6.2 Secure Protocol Composition

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 [53] or something very similar to it [58]. 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 [74], 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 [97]. 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 [47]. A somewhat different approach is used by Lynch [66] 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.

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 [68, 69, 70, 71, 72], 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" [24]. 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 [78], 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.

6.3 Computational Soundness

We use the phrase "computational soundness" to refer to a line of work whose goal is to develop symbolic methods for security analysis faithful to the complexity theoretic model of cryptography. Abadi and Rogaway [4] have initiated this line of research. Their main result is a soundness theorem for a logic of encrypted expressions: a symbolic notion of equivalence between such expressions based on Dolev-Yao deducibility [42] implies computational indistinguishability. This work has been extended to the case of (symbolic) static equivalence [11], while other works investigate completeness aspects of the Abadi-Rogaway logic [76, 49, 5]. All these results hold for a passive adversary, while our results are set in the more general and more realistic

framework of active adversaries.

The active setting has been investigated by Backes, Pfitzmann, and Waidner [10] and by Micciancio and Warinschi [77], with further refinements and applications provided in [31, 62, 9]. In these approaches, the core results are emulation theorems that state that the behavior of arbitrary computational adversaries can be emulated by symbolic adversaries. It follows from an emulation theorem that security in the symbolic model implies security in the computational model. However, current emulation theorems require strong cryptographic assumptions (e.g., IND-CCA2 encryption) while the present paper allows weaker assumptions. Our approach appears to offer a higher degree of flexibility and modularity when compared to [10, 77], which requires a new emulation theorem for each added primitive; this may be difficult or impossible in some cases [8]. Similarly, new primitives can be added to the present framework by adding appropriate axioms and proof rules to the logic and proving them sound. However, this appears easier, primarily because it is not necessary to completely axiomatize new primitives, but only to formalize the properties that are needed to prove protocols of interest correct. For instance, our axiom for exponentiation does not explicitly give any algebraic properties (although the soundness proof certainly accounts for them), and only reflects the Decisional Diffie-Hellman assumption.

A complementary line of research is proposed by Impagliazzo and Kapron [60], who provide a logic for reasoning about indistinguishability. Their logic is appropriate for reasoning about security of primitives, but has not been extended to deal with protocols.

An approach similar to the one presented here is taken by Gupta and Shmatikov [54] who extend the logic of [37] with signatures and Diffie-Hellman keys, and then use the resulting logic to express security properties of key exchange protocols. The main result is a proof that protocols that satisfy their security requirement are secure with respect to a computational model for secure key exchange due to Shoup [94]. Their logical characterization of secure keys is based on indistinguishability, and unlike our notion is not composable.

6.4 Compositional Notions of Key Exchange

In previous work, three different approaches have been used to define security of key-exchange protocols: the indistinguishability-based approach [17], the simulation-based security paradigm [94], and universal composability [24] or reactive simulatability [86].

The indistinguishability-based approach was proposed by Bellare and Rogaway [17]. A central aspect of this definition is the notion of *key indistinguishability*, which states that an attacker cannot distinguish between the real key and one chosen at random. This model was refined and extended by Bellare, Petrank, Rackoff and Rogaway (in unpublished work) and later by Canetti and Krawczyk [26]. The approach of Canetti and Krawczyk also offers a limited form of composition guarantees. Specifically, they prove that a key exchange protocol which satisfies their definition can be securely composed with a specific secure sessions protocol, which uses the exchanged key. However, key indistinguishability is not generally preserved once the key is used. While [26] provides for a *specific* composition, their theorem would not apply, for example, to IEEE 802.11i, where the key exchange protocol (TLS [39]) is composed with a protocol that uses the exchanged key to set up other fresh keys for securing data transmission.

Bellare, Canetti, Krawzyck [15] and Shoup [94] provide simulation-based alternatives. This line of research is grounded in foundational work on secure multi-party computation. Here, security of a real protocol is asserted by comparing it with an ideal protocol, which is secure by construction. As usual with this approach, while the resulting definitions are quite appealing to intuition, security proofs may be quite involved. Moreover, the basic framework of secure multi-party computation does not have built-in compositionality guarantees, and neither of these two models offers improvements with respect to this important aspect.

Finally, the universal composability framework of Canetti [24] has the explicit goal of providing a framework where demonstrated security is preserved under arbitrary composition. Working in this setting Canetti and Krawczyk [27] prove an equivalence between single-session UC-security of key exchange protocols and the

indistinguishability-based notion introduced in [26], and their result implies that indistinguishability-based notion may be composable. However, the general compositionality properties offered by the basic UC framework only apply when primitives do not share state. A partial solution is offered in [28], which allows multiple sessions of a protocol to share state under some very specific conditions.

Chapter 7

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.

In order to bridge the gap between the symbolic and computational models of security, we propose a computational semantics of a subset of the symbolic Protocol Composition Logic. The associated soundness theorem indicates that it is possible to reason symbolically and at a high level, about security properties defined in terms of probabilistic polynomial-time adversaries. Using CPCL we propose and formalize an approach for proving the security of the key exchange protocols in the computational

model.

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.

Bibliography

[1] IEEE P802.11i/D10.0. Medium Access Control (MAC) security enhancements, amendment 6 to IEEE Standard for local and metropolitan area networks part 11: Wireless Medium Access Control (MAC) and Physical Layer (PHY) specifications., April 2004.

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

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

[4] M. Abadi and P. Rogaway. Reconciling two views of cryptography (the computational soundness of formal encryption). *Journal of Cryptology*, 15(2):103–127, 2002.

[5] P. Adão, G. Bana, and A. Scedrov. Computational and information-theoretic soundness and completeness of formal encryption. In *Proc. of the 18th IEEE Computer Security Foudnations Workshop*, pages 170–184, 2005.

[6] R. Alur and T. A. Henzinger. Computer-aided verification. an introduction to model building and model checking for concurrent systems. Draft, 1998.

[7] M. Backes, 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*. IEEE, 2005.

[8] M. Backes and B. Pfitzmann. Limits of the cryptographic realization of XOR. In *Proc. of the 10th European Symposium on Research in Computer Security*. Springer-Verlag, 2005.

[9] M. Backes and B. Pfitzmann. Relating symbolic and cryptographic secrecy. In *Proc. IEEE Symposium on Security and Privacy*, pages 171–182. IEEE, 2005.

[10] M. Backes, B. Pfitzmann, and M. Waidner. A universally composable cryptographic library. Cryptology ePrint Archive, Report 2003/015, 2003.

[11] M. Baudet, V. Cortier, and S. Kremer. Computationally Sound Implementations of Equational Theories against Passive Adversaries. In *Proceedings of the 32nd International Colloquium on Automata, Languages and Programming (ICALP'05)*, volume 3580 of *Lecture Notes in Computer Science*, pages 652–663, Lisboa, Portugal, July 2005. Springer.

[12] R. Beauxis and C. Palamidessi. On the asynchronous nature of the asynchronous π-calculus, 2006. Manuscript.

[13] G. Bella 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*, pages 361–375, Louvain-la-Neuve, Belgium, 1998. Springer-Verlag LNCS 1485.

[14] M. Bellare, A. Boldyreva, and S. Micali. Public-key encryption in a multi-user setting: Security proofs and improvements. In *Advances in Cryptology - EUROCRYPT 2000, Proceedings*, pages 259–274, 2000.

[15] M. Bellare, R. Canetti, and H. Krawczyk. A modular approach to the design and analysis of authenticationand key exchange protocols. In *Proc. of the 30th Annual Symposium on the Theory of Computing*, pages 419–428. ACM, 1998.

[16] M. Bellare and P. Rogaway. Entity authentication and key distribution. In *Advances in Cryprtology - Crypto '93 Proceedings*, pages 232–249. Springer-Verlag, 1994.

[17] M. Bellare and P. Rogaway. Entity authentication and key distribution. In *Proceedings of the 13th Annual International Cryptology Conference on Advances in Cryptology (CRYPTO '93)*, pages 232–249. Springer-Verlag, 1994.

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

[19] N. Borisov, 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*, pages 180–189, 2001.

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

[21] M. Burrows, M. Abadi, and R. Needham. A logic of authentication. *ACM Transactions on Computer Systems*, 8(1):18–36, 1990.

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

[23] F. Butler, 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*, volume 3233 of *Lecture Notes in Computer Science*, pages 1–24. Springer, 2003.

[24] R. Canetti. 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/`.

[25] R. Canetti, L. Cheung, D. Kaynar, M. Liskov, N. Lynch, O. Pereira, and R. Segala. Using probabilistic i/o automata to analyze an oblivious transfer protocol. Technical Report MIT-LCS-TR-1001, MIT CSAIL, 2005.

[26] R. Canetti and H. Krawczyk. Analysis of key-exchange protocols and their use for building secure channels. In *Proc. of EUROCRYPT 2001*, volume 2045 of *LNCS*, pages 453–474, 2001.

[27] R. Canetti and H. Krawczyk. Universally composable notions of key exchange and secure channels. In *EUROCRYPT '02: Proceedings of the International Conference on the Theory and Applications of Cryptographic Techniques*, pages 337–351, London, UK, 2002. Springer-Verlag.

[28] R. Canetti and T. Rabin. Universal composition with joint state. In *Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Proceedings*, volume 2729 of *Lecture Notes in Computer Science*, pages 265–281. Springer-Verlag, 2003.

[29] I. Cervesato, 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.*, December 2006. Preliminary report on http://eprint.iacr.org/2006/009.

[30] I. Cervesato, A. Jaggard, A. Scedrov, J.-K. Tsay, and C. Walstad. Breaking and fixing public-key kerberos. Technical report, 2006.

[31] V. Cortier and B. Warinschi. Computationally sound, automated proofs for security protocols. In *Proceedings of 14th European Symposium on Programming (ESOP'05)*, Lecture Notes in Computer Science, pages 157–171. Springer-Verlag, 2005.

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

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

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

[35] A. Datta, A. Derek, J. C. Mitchell, and D. Pavlovic. Secure protocol composition. In *Proceedings of 19th Annual Conference on Mathematical Foundations of Programming Semantics*. ENTCS, 2004.

[36] A. Datta, A. Derek, J. C. Mitchell, and D. Pavlovic. A derivation system and compositional logic for security protocols. *Journal of Computer Security*, 13(3):423–482, 2005.

[37] A. Datta, 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, pages 16–29. Springer-Verlag, 2005.

[38] A. Datta, 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*, pages 321–334. IEEE, 2006.

[39] T. Dierks and C. Allen. The TLS Protocol — Version 1.0. IETF RFC 2246, January 1999.

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

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

[42] D. Dolev and A. Yao. On the security of public-key protocols. *IEEE Transactions on Information Theory*, 2(29):198–208, 1983.

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

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

[45] S. O. Ehmety and L. C. Paulson. Program composition in isabelle/unity. In *16th International Parallel and Distributed Processing Symposium (IPDPS 2002), Proceedings*. IEEE Computer Society, 2002.

[46] S. O. Ehmety and L. C. Paulson. Mechanizing compositional reasoning for concurrent systems: some lessons. *Formal Aspects of Computing*, 17(1):58–68, 2005.

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

[48] A. Freier, P. Karlton, and P. Kocher. The SSL protocol version 3.0. IETF Internet draft, November 18 1996.

[49] V. Gligor and D. O. Horvitz. Weak Key Authenticity and the Computational Completeness of Formal Encryption. In D. Boneh, editor, *Advances in cryptology - CRYPTO 2003, proceedings of the 23rd annual international cryptology conference*, volume 2729 of *Lecture Notes in Computer Science*, pages 530–547, Santa Barbara, California, USA, Aug. 2003. Springer-Verlag.

[50] O. Goldreich. *Foundations of Cryptography: Basic Applications*. Cambridge University Press, 2004.

[51] S. Goldwasser and S. Micali. Probabilistic encryption. *Journal of Computer and System Science*, 28:270–299, 1984.

[52] L. Gong, R. Needham, and R. Yahalom. Reasoning About Belief in Cryptographic Protocols. In D. Cooper and T. Lunt, editors, *Proceedings 1990 IEEE*

Symposium on Research in Security and Privacy, pages 234–248. IEEE Computer Society, 1990.

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

[54] P. Gupta and V. Shmatikov. Towards computationally sound symbolic analysis of key exchange protocols. In *Proceedings of ACM Workshop on Formal Methods in Security Engineering*, 2005. to appear.

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

[56] C. He and J. C. Mitchell. Security analysis and improvements for IEEE 802.11i. In *Proceedings of the Network and Distributed System Security Symposium, NDSS 2005*. The Internet Society, 2005.

[57] C. He, 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*, pages 2–15, 2005.

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

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

[60] R. Impagliazzo and B. Kapron. Logics for reasoning about cryptographic constructions. In *Prof of 44th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 372–383, 2003.

[61] R. Impagliazzo and B. M. Kapron. Logics for reasoning about cryptographic constructions. In *Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science (FOCS '03)*, pages 372–383. IEEE, 2003.

[62] R. Janvier, L. Mazare, and Y. Lakhnech. Completing the picture: Soundness of formal encryption in the presence of active adversaries. In *Proceedings of 14th European Symposium on Programming (ESOP'05)*, Lecture Notes in Computer Science, pages 172–185. Springer-Verlag, 2005.

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

[64] P. D. Lincoln, 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, pages 776–793. Springer-Verlag, 1999.

[65] G. Lowe. Some new attacks upon security protocols. In *Proceedings of 9th IEEE Computer Security Foundations Workshop*, pages 162–169. IEEE, 1996.

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

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

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

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

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

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

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

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

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

[75] C. Meadows and D. Pavlovic. Deriving, attacking and defending the GDOI protocol. In *Computer Security - ESORICS 2004, 9th European Symposium on Research Computer Security, Proceedings*, volume 3193 of *Lecture Notes in Computer Science*, pages 53–72. Springer, 2004.

[76] D. Micciancio and B. Warinschi. Completeness theorems for the Abadi-Rogaway logic of encrypted expressions. *Journal of Computer Security*, 12(1):99–129, 2004. Preliminary version in WITS 2002.

[77] D. Micciancio and B. Warinschi. Soundness of formal encryption in the presence of active adversaries. In *Theory of Cryptography Conference - Proceedings of TCC 2004*, volume 2951 of *Lecture Notes in Computer Science*, pages 133–151. Springer-Verlag, 2004.

[78] J. Misra and K. M. Chandy. Proofs of networks of processes. *IEEE Transactions on Software Engineering*, 7(4):417–426, 1981.

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

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

[81] J. C. Mitchell, V. Shmatikov, and U. Stern. Finite-state analysis of SSL 3.0. In *Proceedings of Seventh USENIX Security Symposium*, pages 201–216, 1998.

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

[83] N. J. Nilsson. Probabilistic logic. *Artificial Intelligence*, 28(1):71–87, 1986.

[84] L. C. Paulson. Mechanized proofs for a recursive authentication protocol. In *Proceedings of 10th IEEE Computer Security Foundations Workshop*, pages 84–95, 1997.

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

[86] B. Pfitzmann and M. Waidner. A model for asynchronous reactive systems and its application to secure message transmission. In *IEEE Symposium on Security and Privacy*, pages 184–200, Washington, 2001.

[87] A. Ramanathan, 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*, volume 2987 of *Lecture Notes in Computer Science*, pages 468–483. Springer-Verlag, 2004.

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

[89] A. Roy, A. Datta, A. Derek, and J. C. Mitchell. Inductive proof method for computational secrecy, 2006. Manuscript.

[90] A. Roy, 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.

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

[92] S. Schneider. Security properties and CSP. In *IEEE Symp. Security and Privacy*, 1996.

[93] S. Schneider. Verifying authentication protocols with csp. *IEEE Transactions on Software Engineering*, pages 741–58, 1998.

[94] V. Shoup. On formal models for secure key exchange (version 4). Technical Report RZ 3120, IBM Research, 1999.

[95] D. Song. Athena: a new efficient automatic checker for security protocol analysis. In *Proceedings of 12th IEEE Computer Security Foundations Workshop*, pages 192–202. IEEE, 1999.

[96] P. Syverson and P. C. van Oorschot. On unifying some cryptographic protocol logics. In *Proceedings of 7th IEEE Computer Security Foundations Workshop*, pages 14–29, 1994.

[97] F. J. Thayer, J. C. Herzog, and J. D. Guttman. Mixed strand spaces. In *Proceedings of 12th IEEE Computer Security Foundations Workshop*, pages 72–82. IEEE, 1999.

[98] "Verified by Visa" security program. Internet Resource.

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

[100] B. Warinschi. A computational analysis of the Needham-Schroeder(-Lowe) protocol. In *Proceedings of 16th Computer Science Foundation Workshop*, pages 248–262. ACM Press, 2003.

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

Appendix A

Protocol Composition Logic

A.1 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.

Programming Language and the Execution Model Set of terms of PCL (see Table 2.1 in Section 2.1) 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 2.1 in Section 2.1) 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

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

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.

A.2 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 2.3.1. We omit proofs for standard axioms and rules of first order logic.

A.2.1 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 [44]. 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.

A.2.2 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 $Q, R \models \mathbf{ENC}$ 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 $\mathsf{a}(x)[y := \texttt{dec}\ x, K]_X\ \mathsf{a}(ENC_K\{\!|y|\!\})$

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 $\mathcal{Q}, R_0R_1 \models \sigma\mathsf{a}(t)$.

A.2.3 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 [44].

A.2.4 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** $t \not\sqsubseteq a$

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.

A.2.5 Temporal Ordering of Actions

FS1 $\mathsf{Fresh}(X,t)[\mathtt{send}\ t']_X\mathsf{FirstSend}(X,t,t')$ **where** $t \sqsubseteq t'$

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** $t \sqsubseteq t''$

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 2.1.1 and 2.1.3.

A.2.6 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 2.1.3, 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".

A.2.7 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.