SECURITY ANALYSIS OF NETWORK PROTOCOLS: COMPOSITIONAL REASONING AND COMPLEXITY-THEORETIC FOUNDATIONS

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

Anupam Datta
September 2005

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

This dissertation addresses two central problems associated with the design and security analysis of network protocols that use cryptographic primitives. The first problem pertains to the *secure composition of protocols*, where the goal is to develop methods for proving properties of complex protocols by combining independent proofs of their parts. In order to address this problem, we have developed a framework consisting of two formal systems: *Protocol Derivation System (PDS)* and *Protocol Composition Logic (PCL)*. PDS supports syntactic derivations of complex protocols, starting from basic components, and combining or extending them using a sequence of composition, refinement, and transformation operations. PCL is a Floyd-Hoare style logic that supports axiomatic proofs of protocol properties. The eventual goal is to develop proof methods for PCL for every derivation operation in PDS, thereby enabling the parallel development of protocols and their security proofs. In this dissertation, we present proof methods for reasoning about protocol composition and a class of protocol refinements. The composition theorems are formulated and proved by adapting ideas from the assume-guarantee paradigm for reasoning about distributed systems. PDS and PCL have been successfully applied to a number of industrial network security protocols, in several instances identifying serious security vulnerabilities.

The second problem pertains to the *computational soundness of symbolic protocol analysis*. At a high-level, this means that a logical method for protocol analysis should have an associated soundness theorem, which guarantees that a completely symbolic analysis or proof has an interpretation in the standard complexity-theoretic model of modern cryptography. Our approach to this problem involves defining complexity-theoretic semantics and proving a soundness theorem for a variant of PCL which we call *Computational PCL*.

While the basic form of the logic remains unchanged, there are certain important differences involving the interpretation of implication in terms of conditional probability and the semantics of the predicates used to capture secrecy properties.

The final result in the dissertation spans both the problems. An alternative way of specifying and reasoning about protocol composition is through equivalence or simulation between the real protocol and an ideal protocol, which is secure by construction. We prove that, under reasonable assumptions about the communication model, three simulation-based definitions for protocol security–*universal composability*, *black-box simulatability*, and *process observational equivalence*–express the same properties of a protocol. The proofs are axiomatic and are carried out using process calculus equational principles. Since these equational principles are rather general, the proofs carry over to a number of process calculi, in particular, the *probabilistic poly-time process calculus*, whose execution model is consistent with the complexity-theoretic model of cryptography. We also observe certain important differences between the composition paradigm of universal composability, and the assume-guarantee paradigm of PCL.

Acknowledgements

I would like to thank my advisor, John Mitchell, for valuable guidance. I have learnt a lot from him. I would also like to thank the faculty members on my reading and orals committees: Dan Boneh, David Dill, Rajeev Motwani, and Stanley Peters for providing feedback on the research results. Additional thanks to Dan and David for teaching an excellent set of courses which helped me greatly in my research. I would also like to express my gratitude towards Dusko Pavlovic and Andre Scedrov for providing direction and advice. Much of the work on the logic and derivation system presented in this dissertation was carried out in collaboration with Dusko.

I was fortunate to be part of a very active research group here at Stanford. I have worked closely with and learnt a lot from several students, postdocs and visitors in our group. Special thanks to Ante Derek and Ajith Ramanathan for the excellent brain-storming sessions and the fun times. Each chapter in this dissertation is joint work with at least one of them. Thanks also to Andrei Aron, Dan Auerbach, Michael Backes, Adam Barth, Changhua He, Cary Kempston, Ralf Küsters, Mathieu Turuani, Arnab Roy, Vitaly Shmatikov, Mukund Sundararajan, and Bogdan Warinschi for fruitful collaborations and interesting conversations.

I would also like to express my gratitude towards several faculty members at IIT Kharagpur who got me excited about research and who continue to provide encouragement. Finally, I would like to thank my friends for their company, and my (extended) family for unconditional support and understanding over the years.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Protocols that enable secure communication over an untrusted network constitute an important part of the current computing infrastructure. Common examples of such protocols are SSL [53], TLS [44], Kerberos [106], and the IPSec [73] and IEEE 802.11i [1] protocol suites. SSL and TLS are used by internet browsers and web servers to allow secure transactions in applications like online banking. The IPSec protocol suite provides confidentiality and integrity at the IP layer and is widely used to secure corporate VPNs. IEEE 802.11i provides data protection and integrity in wireless local area networks, while Kerberos is used for network authentication.

The design and security analysis of such network protocols presents a difficult problem. In several instances, serious security vulnerabilities were uncovered in protocols many years after they were first published or deployed [105, 59, 1, 16, 104, 68]. While some of these attacks rely on subtle properties of cryptographic primitives, a large fraction can be traced to intricacies in designing protocols that are robust in a concurrent execution setting. To further elaborate this point, let us consider the concrete example of the SSL protocol. In SSL, a client typically sets up a key with a web server. That key is then used to protect all data exchanged between them. A single client can simultaneously engage in sessions with multiple servers and a single server concurrently serves many clients. Let us consider a scenario in which all network traffic is under the control of the adversary. In addition, the adversary may also control some of the clients and servers. The protocol should guarantee certain security properties for honest agents even in such an adversarial environment.

Specifically, if an honest client executes an SSL session with an honest server, the attacker should not be able to recover the exchanged key. This is called the *key secrecy* property. Furthermore, an attacker should not be able to fool an honest client into believing that she has completed a session with an honest server unless that is indeed the case. This property is called *authentication*. The security proof that SSL does indeed provide these guarantees, even when the cryptography is perfect, turns out to be far from trivial [108, 60]. The central problem is ensuring that the attacker cannot combine data acquired from a possibly unbounded number of concurrent sessions to subvert the protocol goals.

Over the last two decades, 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 [24, 121, 55], process calculi [6, 3, 77, 116] and tools [89, 119], as well as theorem-proving [110, 109] and model-checking methods [79, 102, 117, 118] using general purpose tools. (The cited papers are representative but not exhaustive; see [91] for a more comprehensive survey.) There are several points of difference among these approaches. While most model-checking tools can only analyze a finite number of concurrent sessions of a protocol, some of the logics, process calculi, and theorem-proving techniques yield protocol security proofs without bounding the number of sessions. With the exception of the BAN family of logics [24], 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 [105], Dolev-Yao [47], and much subsequent work by others. In this model, the adversary is allowed to choose non-deterministically among the set of possible actions; messages are represented as abstract terms, not sequences of bits; and encryption and other cryptographic primitives are modelled in an abstract black-box manner. This idealization has been a major enabling factor in the development of the above mentioned array of tools and techniques.

As this research area comes of age, several important open problems have been identified (cf. [91]). One significant problem has to do with *secure composition of protocols*.

Many modern protocols like IKEv2 [71], IEEE 802.11i [1], and Kerberos [106] comprise of several different sub-protocols and modes of operation. The challenge is to develop proof methods that allow security proofs of such composite protocols to be built up by combining independent proofs of their parts. Composition is a difficult problem in security since a component may reveal information that does not affect its own security but may degrade the security of some other component in the system. A second important problem pertains to the *model of protocol execution and attack* used while carrying out the security analysis task. As mentioned before, almost all extant approaches for symbolic protocol analysis use an idealized model where cryptography is assumed to be perfect. This idealization makes the protocol analysis problem more amenable to automation. However, the abstraction detracts from the fidelity of the analysis since attacks arising from the interaction between the cryptosystem and the protocol lie outside the scope of this model. The goal then is to develop logical methods for protocol analysis with associated soundness theorems, which guarantee that a completely symbolic analysis or proof has an interpretation in the standard complexity-theoretic model of modern cryptography. At an informal level, this means that a machine-checkable or generated proof should carry the same meaning as a hand-proof done by a cryptographer. This turns out to be a difficult problem since the security definitions of cryptographic primitives and protocols involve complex probability spaces and quantification over all probabilistic polynomial time attackers. In this dissertation, we initiate a program and take several steps towards solving these two problems. In the following paragraphs, we summarize our main results and sketch several directions for future work.

We have developed a framework consisting of two formal systems: *Protocol Derivation System (PDS)* and *Protocol Composition Logic (PCL)*. Within the protocol analysis spectrum, this work can be placed in the category of specialized logics. PDS supports syntactic derivations of complex protocols, starting from basic components, and combining or extending them using a sequence of composition, refinement, and transformation operations. PCL is a Floyd-Hoare style logic [52, 64] that supports axiomatic proofs of protocol properties. The eventual goal is to develop proof methods for PCL for every derivation operation in PDS, thereby enabling the parallel development of protocols and their security proofs. In this dissertation, we present proof methods for reasoning about protocol composition and a class of protocol refinements. The composition theorems are formulated and proved by adapting ideas from the assume-guarantee paradigm for reasoning about distributed systems. PDS and PCL have been successfully applied to a number of industrial network security protocols, in several instances identifying serious security vulnerabilities. Specifically, PCL has been applied to the IEEE 802.11i protocol suite (which includes TLS as a component) [60] and to the IETF GDOI protocol for secure group communication [92]. The second case study identified a previously undiscovered flaw in the protocol. In ongoing work, PCL is being used to analyze IKEv2 [71], IEEE 802.16e [2], Kerberos [106], and Mobile IPv6 [70] protocols. While the semantics of PCL is defined with respect to the idealized, symbolic model, we also present complexity-theoretic foundations and prove a soundness theorem for a variant of PCL which we call *Computational PCL*. The soundness proof uses standard proof techniques from cryptography, in particular, complexity-theoretic reductions. Although the basic form of the logic remains unchanged, there are certain important differences involving the interpretation of implication in terms of conditional probability and the semantics of the predicates used to capture secrecy properties.

The final result in the dissertation spans both the problems. An alternative way of specifying and reasoning about protocol composition is through equivalence or simulation between the real protocol and an ideal protocol, which is secure by construction. We prove that, under reasonable assumptions about the communication model, three simulation-based definitions for protocol security–*universal composability*, *black-box simulatability*, and *process observational equivalence*–express the same properties of a protocol. The proofs are axiomatic and are carried out using process calculus equational principles. Since these equational principles are rather general, the proofs carry over to a number of process calculi, in particular, the *probabilistic poly-time process calculus*, whose execution model is consistent with the complexity-theoretic model of cryptography. We also observe certain important differences between the composition paradigm of universal composability, and the assume-guarantee paradigm of PCL.

Although these results represent significant advances in the state-of-the-art, we expect that it will take several people a number of years to fully accomplish the goals of this program. One current effort seeks to extend and further refine PCL [10]. A second direction is to extend PCL to reason about security in different threat models. Threat models can differ on several respects, in particular, the computational capabilities of protocol principals and adversaries, and the degree of control the adversary has over the network. For example, in the symbolic model the adversary's computational abilities are restricted to a fixed set of actions while in the complexity-theoretic model of cryptography, she can carry out any probabilistic polynomial time computation. Also, in applications like Mobile IPv6, it might be reasonable to assume that the adversary does not have access to all communication paths over the network. A third direction is to continue the work on formalizing PDS to develop a systematic theory of protocol design and to apply it during the design phase of a standards-track protocol. Tool implementation efforts for PCL and PDS are also underway. Finally, the results in this dissertation provide a good starting point for a deeper investigation of the composition problem in computer security and cryptography. The importance of compositional methods in the design and analysis of secure systems is now widely recognized (cf. [126]). However, there is no comprehensive foundational theory for secure composition of secure systems and software. We believe that the assume-guarantee paradigm developed for PCL might be applicable to these other kinds of security mechanisms. In the field of cryptography also, the composition problem has received significant attention. One current approach to this problem is the framework of *universal composability* [26, 114]. The universal composability condition provides strong composition guarantees: a primitive or protocol that satisfies this condition retains its security guarantees in any environment in which it is used. In contrast, the assume-guarantee paradigm of PCL only allows conditional composability: a protocol is secure only in an environment which satisfies a certain set of invariant assumptions associated with the protocol. The ability to reason about protocols under assumptions about the way they will be used offers greater flexibility and appears essential for developing modular proofs about certain classes of protocols. In addition, a number of impossibility results about the realizability of UC-secure primitives and protocols [26, 29, 40] indicates that the UC condition may be too stringent to apply to certain protocols of interest.

The rest of the dissertation is organized as follows. Chapter 2 presents the protocol derivation system. Chapter 3 describes the syntax, semantics, and proof system of PCL. Chapter 4 presents the proof methods in PCL associated with the protocol derivation operations in PDS. The composition theorems are presented in Section 4.1 while the proof method for reasoning about protocol refinements is presented in Section 4.2. Chapter 5 presents complexity-theoretic semantics for a variant of PCL and an associated soundness theorem for its proof system. Chapter 6 presents the results on equivalence between various simulation based security definitions. Related work is discussed in Section 7. Final conclusions and directions for future work are presented in Section 8.

Chapter 2

Protocol Derivation System

Many researchers and practitioners working in the field of protocol security recognize that common authentication and key exchange protocols are built using an accepted set of standard concepts. The common building blocks include Diffie-Hellman key exchange, nonces to avoid replay, certificates from an accepted authority to validate public keys, and encrypted or signed messages that can only be created or read by identifiable parties. An informal practice of presenting protocols incrementally, starting from simple components and extending them by features and functions, is used in [46], with efforts to formalize the practice appearing in [23]. More recently, Bellare, Canetti and Krawczyk [19], for example, have studied protocol transformations that add authentication to a protocol scheme. However, there is no comprehensive theory about how each standard protocol part works, and how properties of a compound protocol can be derived from properties of its parts. In this chapter, we summarize some steps we have taken towards developing such a theory.

Our framework for deriving security protocols consists of a set of basic building blocks called *components* and a set of operations for constructing new protocols from old ones. These operations may be divided into three different types: *composition*, *refinement* and *transformation*. A *component* is a basic protocol step or steps, used as a building block for larger protocols. Diffie-Hellman key exchange and challenge-response are examples of basic components. A *composition* operation combines two protocols. Parallel composition and sequential composition are two examples of composition operations. A *refinement* operation acts on message components of a single protocol. For example, replacing a plaintext nonce by an encrypted nonce is a refinement. A refinement does not change the number of messages or the basic structure of a protocol. A *transformation* operates on a single protocol, and may modify several steps of a protocol by moving data from one message to another, combining steps, or inserting one or more additional steps. For example, moving data from one protocol message to an earlier message (between the same parties) is a transformation.

In principle, there may be many possible protocol refinements and transformations. Our goal in this chapter is to show how protocol composition, refinement, and transformation may be used by working out some examples. In the next section, we examine the structure of a set of key exchange protocols (which we call the STS family) to illustrate the use of this method. Among the derived protocols are STS [46], the standard signature-based challenge-response protocol [93], JFKi, JFKr, ISO-9798-3 [8], and the core of the IKE protocol [59].

2.1 Derivation of the STS Family

The STS family includes protocols like IKE which have been deployed on the Internet and JFKi and JFKr which were considered by IETF as replacements for IKE. The security properties relevant to the STS family of protocols include key secrecy, mutual authentication, denial-of-service protection, identity protection and computational efficiency. Computational efficiency is achieved by reusing Diffie-Hellman exponentials across multiple sessions.

We begin by describing the basic components, and the composition, refinement and transformation operations used in deriving the STS family of key exchange protocols. The components and operations are presented tersely, with additional intuition and explanation given where they are used.

In informally describing the derivation system, we use a standard messages-and-arrows notation for protocol steps. Experience suggests that this simple notation is useful for conveying some of the central ideas. However, the reader should bear in mind that, in addition, a protocol involves initial conditions, communication steps, and internal actions. When we derive a protocol, the derivation step may act on the initial conditions, network messages, or internal actions.

2.1.1 Components

A protocol component consists of a set of roles (e.g., initiator, responder, server), where each role has a sequence of inputs, outputs and protocol actions. Intuitively, a principal executing a role of the protocol starts in a state where it knows the inputs (e.g. its private signing key), executes the prescribed actions (e.g., generates nonces, sends or receives messages) and then produces the outputs (e.g., a shared key if the protocol is a key exchange protocol). In this derivation, we use Diffie-Hellman key exchange and a signature-based authenticator as basic components.

Diffie-Hellman component, C_1

The Diffie-Hellman protocol [45] provides a way for two parties to set up a shared key (g^{ir}) which a passive attacker cannot recover. There is no authentication guarantee: the secret is shared between two parties, but neither can be sure of the identity of the other. Our component C_1 contains only the internal computation steps of the Diffie-Hellman protocol. The initiator and responder role actions are given below.

I: generates random value i and computes g^i (for previously agreed base b)
R: generates random value r and computes g^r (for previously agreed base b)

In this component no messages are sent; the exponentials are considered to be the output of this protocol fragment.

Signature-based authenticator, C_2

The signature-based challenge-response protocol shown below is a standard mechanism for one-way authentication (see Section 10.3.3 of [93])

$$I \rightarrow R : m$$
$$R \rightarrow I : SIG_R(m)$$

It is assumed that m is a fresh value or nonce and that the initiator, I, possesses the public key certificate of responder, R, and can therefore verify the signature.

2.1.2 Composition

The composition operation used is sequential composition of two protocol components with term substitution. The precise definition of this operation is in Section 4.1.1. Intuitively, the roles of the composed protocol have the following structure: the input sequence is the same as that of the first component and the output is the same as that of the second component; the actions are obtained by concatenating the actions of the first component with those of the second (sequential composition) with an appropriate term substitution—the outputs of the first component are substituted for the inputs of the second.

2.1.3 Refinements

While defining refinements, we use the notation $a \Rightarrow b$ to indicate that some instance of message component a in the protocol should be replaced by b.

Refinement R_1 $\quad SIG_X(m) \Rightarrow E_K(SIG_X(m))$, where K is a key shared with the peer. The purpose of this refinement is to provide identity protection against passive attackers. In all the protocols that we consider here, everything signed is public. So, an attacker can verify guesses at identities of a principal if the signature is not encrypted.

Refinement R_2 $\quad SIG_X(m) \Rightarrow SIG_X(HMAC_K(m, ID_X))$, where K is a key shared with the peer. While the signature by itself proves that this term was generated by X, the keyed hash in addition proves that X possesses the key K. This additional property is crucial for mutual authentication guaranteed by IKE. It is further elaborated in the derivation below.

Refinement R_3 $\quad SIG_X(m) \Rightarrow SIG_X(m), HMAC_K(m, ID_X)$, where K is a key shared with the peer. This refinement serves the same purpose as R_2 and is used to derive the core of the JFKr protocol.

Refinement R_4 $\quad SIG_X(m) \Rightarrow SIG_X(m, ID_Y)$, where Y is the peer. It is assumed that X possesses the requisite identifying information for Y, e.g., $Y's$ public key certificate, before the protocol is executed. This assumption can be discharged if X receives $Y's$ identity in an earlier message of the protocol. In public-key based challenge-response protocols, the authenticator should identify both the sender and the intended recipient. Otherwise, the protocol is susceptible to a person-in-the-middle attack. Here, the signature identifies the sender and the identity inside the signature identifies the intended recipient. In an encryption-based challenge-response protocol (e.g., Needham-Schroeder [105]), since the public encryption key identifies the intended recipient, the sender's identity needs to be included inside the encryption. The original protocol did not do so, resulting in the property discovered nearly twenty years later by Lowe [78].

Refinement R_5 $\quad g^x \Rightarrow g^x, n_x$, where n_x is a fresh value. In many Diffie-Hellman based key exchange protocols, the Diffie-Hellman exponentials serve two purposes: (a) they provide the material to derive secret keys; (b) they provide the freshness guarantee for runs required in order to prevent replay attacks. However, Diffie-Hellman exponentials are expensive to compute. This refinement makes participants exchange nonces in addition to Diffie-Hellman exponentials, thereby offloading function (b) onto the nonces. The use of nonces enables the reuse of exponentials across multiple sessions resulting in a more efficient protocol. On the other hand, when exponents are reused, perfect forward secrecy is lost. This tradeoff is offered both by JFKi and JFKr.

Refinement R_6 $\quad SIG_X(m) \Rightarrow SIG_X(m), ID_X$, where ID_X denotes the public key certificate of X. Since the other party may not possess the signature-verification key, it is necessary to include the certificate along with the signature. Unlike refinements R_1 and R_5 above, which add properties to a protocol (identity protection and efficiency respectively), this is an example of a refinement which discharges the assumption that the principals possess each other's public key certificates before the session.

Refinement R_7 $\quad E_K(m) \Rightarrow E_K(m), HMAC_{K'}(role, E_K(m))$, where K and K' are keys shared with the peer and $role$ identifies the protocol role in which this term was produced (initiator or responder). This refinement is used in the derivation of JFKr. Here, each party includes a keyed hash of the encrypted signature and its own role (i.e., initiator or responder) in addition to the signature. The hash serves the same purpose as in refinements R_2, R_3. The protocol role is included inside the hash to prevent reflection attacks.

2.1.4 Transformations

Message component move, T_1

This transformation moves a top-level field t of a message m to an earlier message m', where m and m' have the same sender and receiver, and if t does not contain any data freshly generated or received between the two messages. One reason for using this transformation is to reduce the total number of messages in the protocol.

Binding, T_2

Binding transformations generally add information from one part of a protocol to another in order to "bind" the two parts in some meaningful way. The specific instance of this general concept that we use here adds a nonce from an earlier message into the signed portion of a later message, as illustrated in Figure 2.1.

$$\begin{array}{lcl} I \to R : m & & I \to R : m \\ R \to I : n, SIG_R(m) & \Longrightarrow & R \to I : n, SIG_R(n, m) \\ I \to R : SIG_I(n) & & I \to R : SIG_I(m, n) \end{array}$$

Figure 2.1: An example of a binding transformation

We can understand the value of this transformation by considering the signature-based authenticator, C_2, described above. Protocol C_2 provides one-sided authentication: after executing the protocol, I is assured that the second message was generated by R in response to the first message. However, R does not know the identity of I. Since the goal of a mutual authentication protocol is to provide the authentication guarantee to both parties, it seems likely that we can construct a mutual authentication protocol from two instances (executed in opposite directions) of C_2. However, the sequential composition of two runs of C_2 does not quite do the job, since neither party can be sure that the other participated in one of the runs. If we take the protocol obtained by sequential composition of two instances of C_2, apply transformation T_1 on nonce n to obtain the protocol on the left side of Figure 2.1, and then apply the binding transformation to obtain the one on the right, the resulting protocol with both nonces inside the signatures ensures that m and n belong to the same session.

We note, however, that the protocol on the right side of Figure 2.1 does not guarantee mutual authentication in the conventional sense. Specifically, after I completes a session with R, initiator I cannot be sure that R knows she has completed the same session with I. The stronger guarantee may be achieved by including the peer's identity inside the signatures, as discussed further in Section 2.1.5. Also, note that our formal model does not allow type confusion attacks, which is essential for the soundness of this transformation.

Cookie, T_3

The purpose of the cookie transformation is to make a protocol resistant to blind Denial-of-Service (DoS) attacks. Under certain assumptions, it guarantees that the responder does not have to create state or perform expensive computation before a round-trip communication is established with the initiator. The cookie transformation is described in detail in [43], where it is derived using more primitive operations. Here, we only touch on the main idea.

$$
\begin{array}{lcl}
I \rightarrow R : m_1 & & I \rightarrow R : m_1 \\
R \rightarrow I : m_2 & & R \rightarrow I : m_2^c, HMAC_{HK_R}(m_1, m_2^c) \\
I \rightarrow R : m_3 & \Longrightarrow & I \rightarrow R : m_3, m_1, m_2^c, \\
 & & \qquad\quad HMAC_{HK_R}(m_1, m_2^c) \\
\ldots & & R \rightarrow I : m_2^e \\
 & & \ldots
\end{array}
$$

Figure 2.2: An example of a cookie transformation

An example of a cookie transformation is shown in Figure 2.2. The protocol on the left hand side is a standard three message protocol in which after receiving message m_1, R creates state and replies with message m_2. Clearly, this protocol is vulnerable to both computation and memory DoS attacks. Now assume that the components of message m_2 can be divided into two sets: those that can be computed without performing any expensive operation (denoted by m_2^c) and those that require expensive operations (denoted by m_2^e). In the transformed protocol, upon receiving the first message, the responder R does not create local state and does not perform any expensive computation. Instead, R sends an unforgeable token (cookie) back to I which captures the local state, and resumes the protocol only after the cookie is returned by I. Here the cookie is a keyed hash of message m_1 and m_2^c. The key used for this purpose, HK_R, is known only to R. Since expensive computation and creation of state is deferred till it is established that the initiator can receive messages at the IP address which it claimed as its own, the resulting protocol is resistant to blind DoS attacks.

2.1.5 The Derivation

We now use the components and operations of the derivation system defined above to systematically derive the protocols in the STS family. The complete derivation graph is shown in Figure 2.3. In what follows, we trace the derivations of the various protocols in the graph. At each derivation step, we explain what property that step helps achieve.

Protocol P_1 Obtained by sequential composition of two symmetric copies of component C_2.

$$\begin{aligned}
&I \rightarrow R : m \\
&R \rightarrow I : SIG_R(m) \\
&R \rightarrow I : n \\
&I \rightarrow R : SIG_I(n)
\end{aligned}$$

This is the first step in constructing a mutual authentication protocol from two instances of an unilateral authentication protocol. Here, it is assumed that m and n are fresh values and that I and R possess each other's public key certificates and so can verify the signatures.

Protocol P_2 Obtained from protocol P_1 by using transformation T_1: the component of message 3 is moved up to message 2.

$$\begin{aligned}
&I \rightarrow R : m \\
&R \rightarrow I : n, SIG_R(m) \\
&I \rightarrow R : SIG_I(n)
\end{aligned}$$

This refinement serves to reduce the number of messages in the protocol from 4 to 3.

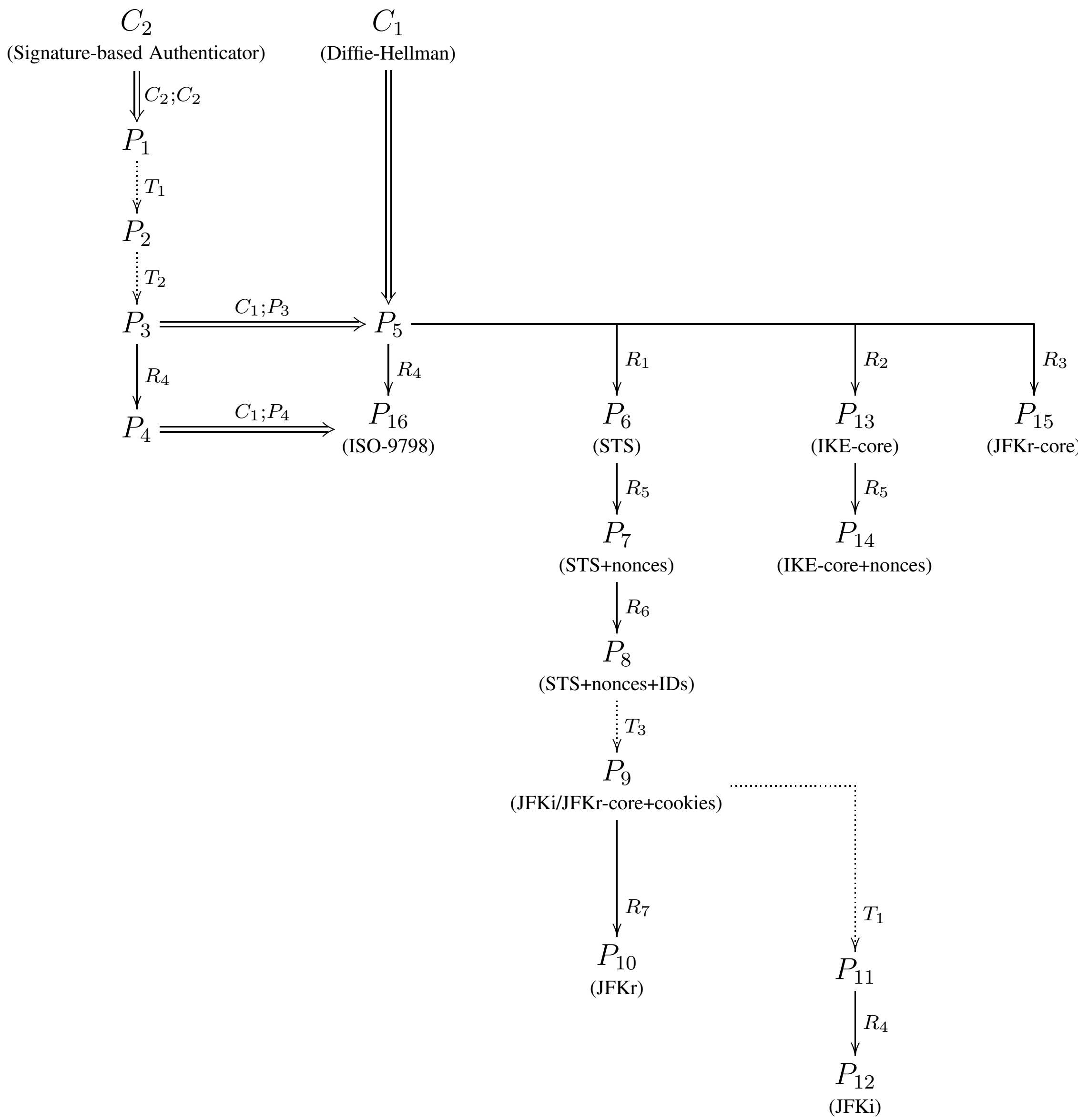

Figure 2.3: Derivation graph of the STS protocol family

Protocol P_3 Obtained from protocol P_2 by using the binding transformation, T_2.

$$\begin{aligned}
&I \rightarrow R : m\\
&R \rightarrow I : n, SIG_R(n, m)\\
&I \rightarrow R : SIG_I(m, n)
\end{aligned}$$

After executing this protocol, I is assured that R generated the second message and moreover that the message was freshly generated. However, as elaborated below, it would be incorrect of I to conclude that R believes that she was talking to I. The source of the problem is that the authenticator does not indicate who the message was meant for. One way to get around it is by applying refinement R_4 mentioned in the previous section. There are other ways too as we will see while proceeding with the derivation.

The following attack describes a scenario in which R and I hold different beliefs about who they completed the session with. Attacker M intercepts and then forwards the first two messages, obtaining nonces m and n. Then M blocks the final message from I and substitutes $SIG_M(m, n)$. After these steps, I believes nonces m and n were exchanged with R, but R believes the nonce m was generated by imposter M.

Protocol P_5 Obtained by composing component C_1 with protocol P_3.

$$\begin{aligned}
&I \rightarrow R : g^i\\
&R \rightarrow I : g^r, SIG_R(g^r, g^i)\\
&I \rightarrow R : SIG_I(g^i, g^r)
\end{aligned}$$

The nonces m and n were instantiated to Diffie-Hellman exponents g^i and g^r. The assumption that m and n are fresh values is still valid as long as i and r are fresh. This is an example of composition by term substitution. Intuitively, the actions that any principal carries out in P_5 is the sequential composition of the actions that she carries out in C_1 and in P_3, except that instead of sending and receiving nonces, she sends and receives Diffie-Hellman exponentials. That is why it makes sense to regard term substitution as a composition operation. Protocol P_5 possesses all the properties of protocol P_3. In addition, whenever I completes a session supposedly with R, then if R is honest, then I and R share a secret, g^{ir}. Note that since the person-in-the-middle attack described above is still possible, R may not believe that she has a shared secret with I.

After protocol P_5, four different derivation paths can be seen in Figure 2.3. The first path includes STS, JFKi and JFKr; the second path includes the core of IKE; the third path includes a protocol that forms the core of IKE-sigma [75] and JFKr; the fourth path includes the ISO-9798-3 protocol. We now describe these derivation paths one by one.

Path 1: STS, JFKi and JFKr

Protocol P_6 Obtained by applying refinement R_1 to protocol P_5, where K is a key derived from the Diffie-Hellman secret. This is the STS protocol.

$$\begin{aligned}&I \rightarrow R : g^i\\&R \rightarrow I : g^r, E_K\left(SIG_R(g^r, g^i)\right)\\&I \rightarrow R : E_K\left(SIG_I(g^i, g^r)\right)\end{aligned}$$

In addition to the properties of P_5, P_6 provides identity protection against passive attackers. As mentioned before, refinement R_1 is geared towards adding this property to the protocol on which it is applied. P_6 also provides a mutually authenticated shared secret. The person-in-the-middle attack described while presenting protocol P_3 (and which is applicable to protocol P_5 too) does not work anymore since an attacker cannot compute the encryption key, K, which depends on the Diffie-Hellman secret, g^{ir}, and hence cannot replace I's signature in the third message by her own. However, Lowe describes another attack on this protocol in [79]. It is not quite clear whether that attack breaks mutual authentication.

Protocol P_7 Obtained by applying refinement R_5 to protocol P_6.

$$\begin{aligned}&I \rightarrow R : g^i, n_i\\&R \rightarrow I : g^r, n_r, E_K\left(SIG_R(g^r, n_r, g^i, n_i)\right)\\&I \rightarrow R : E_K\left(SIG_I(g^i, n_i, g^r, n_r)\right)\end{aligned}$$

P_7 retains all the properties of P_6 except perfect forward secrecy. As mentioned while describing refinement R_5, the use of fresh nonces enables the reuse of Diffie-Hellman exponentials across multiple sessions resulting in a more computationally efficient protocol.

Protocol P_8 Obtained by applying refinement R_6 to protocol P_7.

$$
\begin{aligned}
&I \rightarrow R : g^i, n_i \\
&R \rightarrow I : g^r, n_r, \\
&\qquad\qquad E_K\left(SIG_R(g^r, n_r, g^i, n_i), ID_R\right) \\
&I \rightarrow R : E_K\left(SIG_I(g^i, n_i, g^r, n_r), ID_I\right)
\end{aligned}
$$

By applying refinement R_6 to P_7, no new properties are introduced. Instead, the assumption that the protocol principals possessed each other's public key certificates apriori is discharged by explicitly exchanging certificates alongside the signatures.

Protocol P_9 Obtained by applying the cookie transformation, T_3, to protocol P_8.

$$
\begin{aligned}
&I \rightarrow R : g^i, n_i \\
&R \rightarrow I : g^r, n_r, HMAC_{HK_R}(g^r, n_r, g^i, n_i) \\
&I \rightarrow R : g^i, n_i, g^r, n_r, HMAC_{HK_R}(g^r, n_r, g^i, n_i), \\
&\qquad\qquad E_K\left(SIG_I(g^i, n_i, g^r, n_r), ID_I\right) \\
&R \rightarrow I : E_K\left(SIG_R(g^r, n_r, g^i, n_i), ID_R\right)
\end{aligned}
$$

The cookie transformation ensures that in addition to the properties of protocol P_8, this protocol also possesses the additional property of resistance to blind Denial-of-Service attacks.

At this point, we have derived a protocol that provides key secrecy, mutual authentication, identity protection (for initiator against passive attackers and for responder against active attackers), DoS protection and computational efficiency, i.e., all the stated security properties for this family of protocols. Both JFKi and JFKr are obtained from P_9 and only differ in the form of identity protection that they offer.

Path 1.1: JFKr

Protocol P_{10} Obtained by applying refinement R_7 to P_9. This is essentially JFKr. We ignore some of the message fields (e.g., the security association and the group identifying information) which can be added using two more refinements.

$$
\begin{aligned}
I \rightarrow R &: g^i, n_i \\
R \rightarrow I &: g^r, n_r, HMAC_{HK_R}(g^r, n_r, g^i, n_i) \\
I \rightarrow R &: g^i, n_i, g^r, n_r, HMAC_{HK_R}(g^r, n_r, g^i, n_i), \\
&\quad E_K\left(SIG_I(g^i, n_i, g^r, n_r), ID_I\right), \\
&\quad HMAC_{K'}\left(I, E_K\left(SIG_I(g^i, n_i, g^r, n_r), ID_I\right)\right) \\
R \rightarrow I &: E_K\left(SIG_R(g^r, n_r, g^i, n_i), ID_R\right), \\
&\quad HMAC_{K'}\left(R, E_K\left(SIG_R(g^r, n_r, g^i, n_i), ID_R\right)\right)
\end{aligned}
$$

P_{10} retains all the properties of P_9. The keyed hash of the encrypted signature appears to serve the same purpose as the encryption of the signature in protocol P_6. It guarantees that since the computation of the keys K and K' requires knowledge of g^{ir}, the adversary cannot launch the person-in-the-middle attack described while presenting protocol P_3, since she cannot compute the encrypted signature and the keyed hash.

Path 1.2: JFKi

Protocol P_{11} Obtained by applying transformation T_1 to protocol P_9.

$$
\begin{aligned}
I \rightarrow R &: g^i, n_i \\
R \rightarrow I &: g^r, n_r, ID_R, HMAC_{HK_R}(g^r, n_r, g^i, n_i) \\
I \rightarrow R &: g^i, n_i, g^r, n_r, HMAC_{HK_R}(g^r, n_r, g^i, n_i), \\
&\quad E_K\left(SIG_I(g^i, n_i, g^r, n_r), ID_I\right) \\
R \rightarrow I &: E_K\left(SIG_R(g^r, n_r, g^i, n_i)\right)
\end{aligned}
$$

The message component ID_R is moved from message 4 in P_9 to message 2 here. The reason for applying this transformation becomes clear in the next step when the principals include the peer's identity inside the signatures. Since I's signature is part of the third message of the protocol, she must possess R's identity before she sends out that message. This protocol retains all the properties of P_9 except for the fact that the form of identity protection is different. Unlike P_9, here the responder's identity is not protected. The initiator's identity is still protected against active attackers.

Protocol P_{12} Obtained by applying refinement R_4 to protocol P_{11}. This is JFKi (except for one additional signature in the second message which can be added using one more transformation). As with JFKr, some of the message fields which do not contribute to the core security property are ignored.

$$
\begin{aligned}
& I \rightarrow R : g^i, n_i \\
& R \rightarrow I : g^r, n_r, ID_R, HMAC_{HK_R}(g^r, n_r, g^i, n_i) \\
& I \rightarrow R : g^i, n_i, g^r, n_r, HMAC_{HK_R}(g^r, n_r, g^i, n_i), \\
& \qquad\quad E_K\left(SIG_I(g^i, n_i, g^r, n_r, ID_R), ID_I\right) \\
& R \rightarrow I : E_K\left(SIG_R(g^r, n_r, g^i, n_i, ID_I)\right)
\end{aligned}
$$

The refinement added the peer's identities inside the signatures. ID_R and ID_I are added inside I's and R's signatures in message 3 and message 4 respectively. Including the identities inside the signatures obviates the attack described while presenting protocol P_3 and Lowe's attack on STS [79]. P_{12} retains all the properties of P_{11}.

Path 2: IKE

We now consider the second path starting from protocol P_5. This path includes two protocols closely related to IKE [59].

Protocol P_{13} Obtained by applying refinement R_2 to protocol P_5. This protocol has been described as the core for IKE in [8].

$$
\begin{aligned}
& I \rightarrow R : g^i \\
& R \rightarrow I : g^r, SIG_R\left(HMAC_K(g^r, g^i, ID_R)\right) \\
& I \rightarrow R : SIG_I\left(HMAC_K(g^i, g^r, ID_I)\right)
\end{aligned}
$$

Instead of just signing the Diffie-Hellman exponentials, each principal now signs a keyed hash of the exponentials and their own identities. Since the key used is derived from the Diffie-Hellman secret, g^{ir}, which is known only to I and R, an adversary cannot launch the person-in-the-middle attack described while presenting P_3 and to which P_5 is also susceptible. This protocol therefore provides both mutual authentication and a shared secret between I and R.

Protocol P_{14} Obtained by applying refinement R_5 to protocol P_{13}.

$$\begin{aligned}
&I \rightarrow R : g^i, n_i \\
&R \rightarrow I : g^r, n_r, SIG_R\left(HMAC_K(g^r, n_r, g^i, n_i, ID_R)\right) \\
&I \rightarrow R : SIG_I\left(HMAC_K(g^i, n_i, g^r, n_r, ID_I)\right)
\end{aligned}$$

This step in the derivation exactly parallels the step in the derivation of JFKi and JFKr where, in addition to Diffie-Hellman exponentials, nonces where exchanged. The purpose, as before, is to allow reuse of Diffie-Hellman exponentials across multiple sessions resulting in a more efficient protocol. The tradeoff is that perfect forward secrecy is lost in the process. Note that the original IKE specification did not stipulate the reuse of Diffie-Hellman exponentials across sessions.

Path 3: JFKr/SIGMA-core

The third path starting from protocol P_5 consists of a protocol that has been described as the core for JFKr and IKE-SIGMA in [8].

Protocol P_{15} Obtained by applying refinement R_3 to protocol P_5.

$$\begin{aligned}
&I \rightarrow R : g^i \\
&R \rightarrow I : g^r, SIG_R(g^r, g^i), HMAC_K(g^r, g^i, ID_R) \\
&I \rightarrow R : SIG_I(g^i, g^r), HMAC_K(g^i, g^r, ID_I)
\end{aligned}$$

This protocol is very similar to protocol P_{13} and possesses exactly the same properties (mutual authentication and shared secret). The only difference is that instead of signing the keyed hash, the principals send the hash separately. Since computation of the hash requires possession of the Diffie-Hellman secret, g^{ir}, which is known only to I and R, an adversary cannot launch the person-in-the-middle attack described while presenting P_3 and to which P_5 is also susceptible.

Path 4: ISO-9798-3

Protocol P_{16} Obtained by applying refinement R_4 to protocol P_5. This is the ISO-9798-3 protocol.

$$\begin{aligned}
&I \to R : g^i \\
&R \to I : g^r, SIG_R(g^r, g^i, ID_I) \\
&I \to R : SIG_I(g^i, g^r, ID_R)
\end{aligned}$$

This protocol provides a means for I and R to set up a mutually authenticated shared secret. The person-in-the-middle attack possible on protocol P_5 (and described while presenting protocol P_3) is not possible in this protocol since the principals indicate who the authenticated message is intended for by including the identity of the intended recipient inside the signature. Thus the attacker M cannot forward the second message of the protocol that R sends to her to I since the signature will contain ID_M and not ID_I.

Alternative Derivation of the ISO-9798-3 Protocol

Now we present a derivation of protocol P_{16}, ISO-9798-3 .

Protocol P_4 Obtained by applying refinement R_4 to protocol P_3. This is the standard challenge-response protocol.

$$\begin{aligned}
&I \to R : m \\
&R \to I : n, SIG_R(n, m, ID_I) \\
&I \to R : SIG_I(m, n, ID_R)
\end{aligned}$$

P_3 is refined so that the peer's identity is included inside the signatures. Consequently, the person-in-the-middle attack on P_3 doesn't succeed against P_4. P_4 therefore provides mutual authentication. Protocol P_{16} is now derived by composing component C_1 with protocol P_4 in exactly the same way that P_5 was derived.

2.1.6 Other Issues

Commutativity of Rules

As suggested by protocol P_{16} above, many protocols have several different derivations, obtained by applying compositions, refinements and transformations in different orders. Such *commutativities* of the derivation steps are usually justified by the fact that the properties that they realize are logically independent. For instance, the refinements R_1 (encrypting the signatures) and R_5 (adjoining nonces to the exponentials) commute, because the corresponding properties - identity protection and reusability of exponentials - are logically independent.

Generalization of Refinements

In this introductory presentation, we often selected the refinements leading to the desired properties by a shortest path. Building a library of reusable derivations of a wider family of protocols would justify more general rules. For example, refinement R_1 is a special case of a general refinement: $m \Rightarrow E_K(m)$, where m is any term and K is a shared key. The purpose of this refinement would be to remove the term m from the set of publicly known values.

Chapter 3

Protocol Composition Logic

Protocol Composition Logic (PCL) is a logic for proving security properties of network protocols. A preliminary version of PCL was presented in [49, 50]. In subsequent work [35, 36, 39, 37, 60], we have significantly extended the logic and developed new proof methods. Currently, we are able to prove authentication and secrecy properties of common security protocols by derivations of twenty to sixty lines of proof. The reason for this succinctness is that the proof rules of the logic state general properties of protocol traces that can be reused for many different protocols. The logic is different from previous "belief" logics like BAN [24] and from explicit reasoning about protocol participants and the intruder as in Paulson's Inductive Method [110]. In a sense, the goal of this work was to retain the readability and ease of use of BAN logic while providing the same degree of assurance in the security of protocols as Paulson's Inductive Method.

The logic is designed around a process calculus with actions for each protocol step. Protocol actions are annotated with assertions in a manner resembling dynamic logic for sequential imperative programs. The semantics of our logic is based on sets of traces of protocol executions, following the standard symbolic model of protocol execution and attack. Security proofs involve local reasoning about properties guaranteed by individual actions and global reasoning about actions of honest principals who faithfully follow the protocol. One central idea is that assertions associated with an action will hold in any protocol execution that contains this action. This observation gives us the power to reason about all possible runs of the protocol without explicitly reasoning about possible steps carried out by the adversary. The soundness of most of the axioms and inference rules are based on this observation. A second important insight is that since honest principals faithfully follow the protocol, invariants for the programs for the roles of the protocol also hold in all runs, irrespective of the intruder actions. This idea is codified in the proof system in the *honesty rule*.

The rest of this section is organized as follows. Section 3.1 describes *cord calculus*, the process language for representing protocols. The syntax and semantics of PCL are presented in Section 3.2. The proof system and soundness theorem are presented in Section 3.3.

3.1 Cord Calculus

One important part of security analysis involves understanding the way honest agents running a protocol will respond to messages from a malicious attacker. The common informal arrows-and-messages notation is generally insufficient, since it only presents the executions (or traces) of the protocol that occur when there is no attack. 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 agent performing each protocol role so that we know not only which actions occur in a run, but why they occur.

As explained in [49], we used a form of process calculus that we call cords. *Cords* form an action structure [96, 97, 111], based on π-calculus [99], and related to spi-calculus [4]. The cords formalism is also similar to the approach of the Chemical Abstract Machine formalism [22], in that the communication actions can be viewed as reactions between "molecules". Cord calculus serves as a simple "protocol programming language" which supports our Floyd-Hoare style logical annotations, and verifications in an axiomatic semantics. Cord calculus is summarized in Appendix A.

In this section, we show how protocols are represented in cord calculus with an example. Figure 3.1 shows the ISO-9798-3 protocol [66] in the informal arrows-and-messages notation. The roles of the same protocol are written out as cords in Figure 3.2, writing $\hat{X}$ and $\hat{Y}$ for the agents executing cords $\mathbf{Init}$ and $\mathbf{Resp}$, respectively. The arrows between the

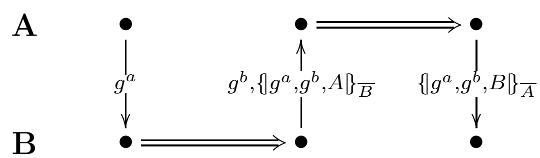

Figure 3.1: ISO-9798-3 as arrows-and-messages

$$\mathbf{Init} = [\ (\nu x)\langle \hat{X}, \hat{Y}, g^x\rangle(\hat{Y}, \hat{X}, y, z)(z/\{|g^x, y, \hat{X}|\}_{\overline{Y}}) \quad \langle \hat{X}, \hat{Y}, \{|g^x, y, \hat{Y}|\}_{\overline{X}}\rangle \quad]$$

$$\mathbf{Resp} = [\ (\hat{X}, \hat{Y}, x)(\nu y) \quad \langle \hat{Y}, \hat{X}, g^y, \{|x, g^y, \hat{X}|\}_{\overline{Y}}\rangle \quad (\hat{X}, \hat{Y}, z)(z/\{|x, g^y, \hat{Y}|\}_{\overline{X}})]$$

Figure 3.2: Cords for ISO-9798-3

cords in the figure are meant to show how messages sent by one cord may be received by the other, but they are not part of the cords formalism. In this example, the protocol consists of two roles, the initiator role and the responder role. The sequence of actions in the initiator role are given by the cord **Init** in Figure 3.2. In words, the actions of a principal executing cord **Init** are: generate a fresh random number; send a message with the Diffie-Hellman exponential of that number to the peer, $\hat{Y}$; receive a message with source address $\hat{Y}$; verify that the message contains $\hat{Y}$'s signature over data in the expected format; and finally, send another message to $\hat{Y}$ with the initiator's signature over the Diffie-Hellman exponential that she sent in the first message, the data she received from $\hat{Y}$ (which should be a Diffie-Hellman exponential generated by $\hat{Y}$) and $\hat{Y}$'s identity. The notations (νx), $\langle t\rangle$, (x) refer respectively to the actions of nonce generation, sending a term and receiving a message. Formally, a *protocol* is given by a finite set of closed cords, one for each role of the protocol. In addition to the sequence of actions, a cord has static input and output parameters (see Appendix A for detailed definitions and Section 4.1.2 for a complete example).

3.2 Protocol Logic

3.2.1 Syntax

The formulas of the logic are given by the grammar in Table 3.1, where ρ may be any role, written using the notation of cord calculus. Here, t and P denote a term and a *thread*, respectively. A thread is a sequence of actions by a principal executing an instance of a role, e.g., Alice executing the initiator role of a protocol. As a notational convention, we use $\hat{X}$ to refer to the principal executing the thread X. We use ϕ and ψ to indicate predicate formulas, and m to indicate a generic term we call a "message". A message has the form (source, destination, protocol-identifier, content), giving each message source and destination fields and a unique protocol identifier in addition to the message contents. The source field of a message may not identify the actual sender of the message since the intruder can spoof the source address. Similarly, the 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 basis for defining precise semantics discussed in the next section.

The formula $\mathsf{Has}(X, x)$ means that principal $\hat{X}$ possesses information x in the thread X. This is "possesses" in the limited sense of having either generated the data or received it in the clear or received it under encryption where the decryption key is known. The formula $\mathsf{Send}(X, m)$ means that the last action in a run of the protocol corresponds to principal $\hat{X}$ sending message m in the thread X. $\mathsf{Receive}(X, m)$, $\mathsf{New}(X, t)$, $\mathsf{Decrypt}(X, t)$, and $\mathsf{Verify}(X, t)$ are similarly associated with the receive, new, decrypt and signature verification actions of a protocol. $\mathsf{Fresh}(X, t)$ means that the term t generated in X is "fresh" in the sense that no one else has seen any term containing t as a subterm. Typically, a fresh term will be a nonce and freshness will be used to reason about the temporal ordering of actions in runs of a protocol. This form of reasoning is useful in proving authentication

Action formulas

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

Formulas

$\phi \quad ::= \quad \mathsf{a} \mid \mathsf{Has}(P, t) \mid \mathsf{Fresh}(P, t) \mid \mathsf{Honest}(N) \mid \mathsf{Contains}(t_1, t_2) \mid \phi \wedge \phi \mid \neg\phi \mid \exists x.\phi \mid \diamondminus\, \phi \mid \ominus\, \phi \mid \mathsf{Start}(P)$

Modal formulas

$\Psi \quad ::= \quad \phi\ \rho\ \phi$

Table 3.1: Syntax of the logic

properties of protocols. The formula $\mathsf{Honest}(\hat{X})$ means that the actions of principal $\hat{X}$ in the current run are precisely an interleaving of initial segments of traces of a set of roles of the protocol. In other words, $\hat{X}$ assumes some set of roles and does exactly the actions prescribed by them. $\mathsf{Contains}(t_1, t_2)$ means that t_2 is a subterm of t_1. This predicate helps us identify the components of a message. The two temporal operators $\diamondminus$ and $\ominus$ have the same meaning as in Linear Temporal Logic [81]. Since we view a run as a linear sequence of states, $\diamondminus\ \phi$ means that in some state in the past ϕ holds, whereas $\ominus\ \phi$ means that in the previous state ϕ holds. $\mathsf{Start}(X)$ means that thread X did not preform any actions in the past.

We note here that the temporal operator $\diamondminus$ and some of the predicates (Send, Receive) bear semblance to those used in NPATRL [120], the temporal requirements language for the NRL Protocol Analyzer [88, 89]. However, while NPATRL is used for specifying protocol requirements, our logic is also used to infer properties of protocols.

Our formalization of authentication is based on the notion of matching records of runs [46] which requires that whenever $\hat{A}$ and $\hat{B}$ accept each other's identities at the end of a run, their records of the run should match, i.e., each message that $\hat{A}$ sent was received by $\hat{B}$ and vice versa, each send event happened before the corresponding receive event, and moreover the messages sent by each principal ($\hat{A}$ or $\hat{B}$) appear in the same order in both the records. Including the source and destination fields in the message allows us to match up send-receive actions. Since we reason about correctness of a protocol in an environment in which other protocols may be executing concurrently, it is important that when $\hat{A}$ and

$\hat{B}$ accept each other's identities, they also agree on which protocol they have successfully completed with the other. One way to extend the matching histories characterization to capture this requirement is by adding protocol identifiers to messages. Now if $\hat{A}$ and $\hat{B}$ have matching histories at the end of a run, not only do they agree on the source, destination and content of each message, but also on which protocol this run is an instance of.

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

The inductive definition of $Q, R \models \phi$ is given in Appendix B. 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 principal in some role. Associating that action with the state that the system ends up in as a consequence, allows us to use the well-understood terminology of LTL in our logic. A formula is true in a run if it is true in the last state of that run. An action formula a is therefore true in a run if it is the last action in that run. On the other hand, a past formula $\diamondminus \mathsf{a}$ is true if in the past the action formula a was true in some state, i.e., if the action had occurred in the past.

AA1	$\phi[a]_X \diamondminus \mathsf{a}$
AA2	$\mathsf{Fresh}(X,t)[a]_X \diamondminus (\mathsf{a} \wedge \ominus\mathsf{Fresh}(X,t))$
AN2	$\phi[(\nu n)]_X\ \mathsf{Has}(Y,n) \supset (Y = X)$
AN3	$\phi[(\nu n)]_X\ \mathsf{Fresh}(X,n)$
ARP	$\diamondminus\ \mathsf{Receive}(X,p(x))[(q(x)/q(t))]_X \diamondminus \mathsf{Receive}(X,p(t))$

Table 3.2: Axioms for protocol actions

3.3 Proof System

The proof system contains 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. The axioms and inference rules specific to reasoning about protocols are presented briefly here, with additional explanation given in Appendix C.

3.3.1 Axioms for Protocol Actions

The axioms about protocol actions are listed in Table 3.2. All the axioms state properties that hold in the state reached by executing one of the actions from a state in which a precondition related to the action is assumed. Note that the a in axioms $\mathbf{AA1}$ and $\mathbf{AA2}$ is any one of the 5 actions and a is the corresponding predicate in the logic. $\mathbf{AA1}$ 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. $\mathbf{AA2}$ states that if a term t is fresh in some state, then it remains fresh until the corresponding thread executes an action. If thread X generates a new value n and does no further actions, then $\mathbf{AN2}$ says that no one else knows n, and $\mathbf{AN3}$ says that n is fresh.

3.3.2 Axioms relating Atomic Predicates

Table 3.3 lists axioms relating various propositional properties, most of which follow naturally from the semantics of atomic formulas. The *possession axioms* characterize the terms that a principal can derive if it possesses certain other terms. $\mathbf{ORIG}$ and $\mathbf{REC}$ state respectively that a principal possesses a term if she freshly generated it (a nonce) or if she received it in some message. $\mathbf{TUP}$ and $\mathbf{ENC}$ enable construction of tuples and encrypted terms if the parts are known. $\mathbf{PROJ}$ and $\mathbf{DEC}$ allow decomposition of a tuple into its components and decryption of an encrypted term if the key is known. The next two axioms are aimed at capturing the *black-box model of encryption and signature*. $\mathbf{VER}$ refers to the unforgeability of signatures while $\mathbf{SEC}$ stipulates the need to possess the private key in order to decrypt a message encrypted with the corresponding public key. The additional condition requiring principal $\hat{X}$ to be honest guarantees that the intruder is not in possession of the private keys. An important axiom is $\mathbf{N1}$ which states that if a thread X has generated a value n, then that value is distinct from all other values generated in all other roles. $\mathbf{N2}$ states that freshly generated values within the same thread are distinct from each other (here $\mathsf{After}(\mathsf{a}, \mathsf{b})$ is a shorthand for $\diamondminus(\mathsf{b} \wedge \ominus\diamondminus\mathsf{a})$). $\mathbf{F1}$ states that fresh values generated in different threads are distinct. $\mathbf{N1}$, $\mathbf{N2}$, and $\mathbf{F1}$ together capture the intuition that fresh nonces and Diffie-Hellman exponentials are unique. Finally, $\mathbf{CON}$ states that a term contains its subterms.

3.3.3 Modal Axioms and Rules

Table 3.4 collects the inference rules and some additional axioms. The generic inference rules follow naturally from the semantics. $\mathbf{G2}$ is exactly of the same form as the rule of consequence in Hoare Logic. It is clear that most predicates are preserved by additional actions. For example, if in some state $\mathsf{Has}(X, n)$ holds, then it continues to hold, when X executes additional actions. Intuitively, if a thread possesses some information at a point in a run, then she remembers it for the rest of the run. Note, however, that the Fresh predicate is preserved only if the fresh term t is not sent out in a message (see $\mathbf{P2}$). Sequencing rule $\mathbf{S1}$ gives us a way of sequentially composing two cords P and P' when post-condition of P, matches the pre-condition or P'

Possession Axioms:

$$
\begin{array}{rl}
\textbf{ORIG} & \diamond\!\!\!-\, \mathsf{New}(X, n) \supset \mathsf{Has}(X, n) \\
\textbf{REC} & \diamond\!\!\!-\, \mathsf{Receive}(X, x) \supset \mathsf{Has}(X, x) \\
\textbf{TUP} & \mathsf{Has}(X, x) \wedge \mathsf{Has}(X, y) \supset \mathsf{Has}(X, (x, y)) \\
\textbf{ENC} & \mathsf{Has}(X, x) \wedge \mathsf{Has}(X, K) \supset \mathsf{Has}(X, \{|x|\}_K) \\
\textbf{PROJ} & \mathsf{Has}(X, (x, y)) \supset \mathsf{Has}(X, x) \wedge \mathsf{Has}(X, y) \\
\textbf{DEC} & \mathsf{Has}(X, \{|x|\}_K) \wedge \mathsf{Has}(X, K) \supset \mathsf{Has}(X, x)
\end{array}
$$

Encryption and Signature:

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

Uniqueness of Nonces:

$$
\begin{array}{rl}
\textbf{N1} & \diamond\!\!\!-\, \mathsf{New}(X, n) \wedge \diamond\!\!\!-\, \mathsf{New}(Y, n) \supset (X = Y) \\
\textbf{N2} & \mathsf{After}(\mathsf{New}(X, n_1), \mathsf{New}(X, n_2)) \supset (n_1 \neq n_2) \\
\textbf{F1} & \diamond\!\!\!-\, \mathsf{Fresh}(X, t) \wedge \diamond\!\!\!-\, \mathsf{Fresh}(Y, t) \supset (X = Y)
\end{array}
$$

Subterm Relation:

$$
\begin{array}{rl}
\textbf{CON} & \mathsf{Contains}((x, y), x) \wedge \mathsf{Contains}((x, y), y)
\end{array}
$$

Table 3.3: Basic Axioms

Generic Rules:

$$\frac{\theta[P]_X\phi \quad \theta[P]_X\psi}{\theta[P]_X\phi \wedge \psi}\,\mathbf{G1} \qquad \frac{\theta[P]_X\phi \quad \theta' \supset \theta \quad \phi \supset \phi'}{\theta'[P]_X\phi'}\,\mathbf{G2} \qquad \frac{\phi}{\theta[P]_X\phi}\,\mathbf{G3}$$

Sequencing rule:

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

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

P1 $\mathsf{Persist}(X,t)[a]_X\mathsf{Persist}(X,t)$
P2 $\mathsf{Fresh}(X,t)[a]_X\mathsf{Fresh}(X,t)$, where $t \not\subseteq a$ or $a \neq \langle m \rangle$
P3 $\mathsf{HasAlone}(X,n)[a]_X\mathsf{HasAlone}(X,n)$, where $n \not\subseteq_v a$ or $a \neq \langle m \rangle$

$$\mathsf{HasAlone}(X,t) \equiv \mathsf{Has}(X,t) \wedge (\mathsf{Has}(Y,t) \supset X = Y)$$

Freshness Loss Axiom:

F $\theta[\langle m \rangle]_X\neg\mathsf{Fresh}(X,t)$, where $(t \subseteq m)$

Table 3.4: Modal Axioms and Rules

3.3.4 Axioms and Rules for Temporal Ordering

In order to prove mutual authentication, we need to reason about the temporal ordering of actions carried out by different threads. For this purpose, we use a fragment of the proof system for Propositional Linear Temporal Logic, PLTL (Table 3.5). See [112] for a complete axiomatization of PLTL. The axioms and rules specific to the temporal ordering of actions are presented in Table 3.5. We use $\mathsf{a}_1, \ldots, \mathsf{a}_\mathsf{n}$, to denote action formulas corresponding to actions $a_1, \ldots, a_n$. Similarly, b_1 and b_2 stand for any action predicates. The rules are fairly straightforward. $\mathbf{AF0}$ simply states that before a thread X executes any action, it is true that X did not execute any actions in the past. $\mathbf{AF1}$ orders the actions within a role. This is consistent with the way we view a role as an ordered sequence of actions. $\mathbf{AF2}$ uses the freshness of terms to reason about the ordering of actions carried out by different threads. Intuitively, $\mathbf{AF2}$ states that if a thread X has a fresh value t at some point in the run and than executes action $\mathsf{b}_1(X, t_1)$, then any action $\mathsf{b}_2(Y, t_2)$ carried out by any other thread which involves t (e.g. if Y receives a message containing t inside a signature), happens after the action b_1.

3.3.5 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 [81] 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 [49, 50], 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

PLTL Axioms:

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

Temporal Generalization Rule:

$$\frac{\phi}{\neg\diamondminus\neg\phi}\,\mathbf{TGEN}$$

Temporal Ordering of actions:

$$\begin{array}{rcl}
\mathsf{After}(\mathsf{a},\mathsf{b}) & \equiv & \diamondminus(\mathsf{b} \wedge \ominus\diamondminus\,\mathsf{a}) \\
\mathsf{ActionsInOrder}(\mathsf{a_1},\ldots,\mathsf{a_n}) & \equiv & \mathsf{After}(\mathsf{a_1},\mathsf{a_2}) \wedge \cdots \wedge \mathsf{After}(\mathsf{a_{n-1}},\mathsf{a_n})
\end{array}$$

$$\begin{array}{ll}
\mathbf{AF0} & \mathsf{Start}(X)[\,]_X\ \neg\diamondminus\,\mathsf{a}(\mathsf{X},\mathsf{t}) \\
\mathbf{AF1} & \theta[a_1 \ldots a_n]_X\ \mathsf{After}(\mathsf{a_1},\mathsf{a_2}) \wedge \ldots \wedge \mathsf{After}(\mathsf{a_{n-1}},\mathsf{a_n}) \\
\mathbf{AF2} & (\diamondminus\,(\mathsf{b_1}(X,t_1) \wedge \ominus\mathsf{Fresh}(X,t)) \wedge \diamondminus\mathsf{b_2}(Y,t_2)) \supset \\
 & \quad \mathsf{After}(\mathsf{b_1}(X,t_1),\mathsf{b_2}(Y,t_2)), \text{ where } t \subseteq t_2 \text{ and } X \neq Y
\end{array}$$

Table 3.5: Axioms and rules for temporal ordering

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 $\mathcal{Q} = \{\rho_1, \rho_2, \ldots, \rho_k\}$ is a set of roles, 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. Using $\rho \in \mathcal{Q}$ to indicate that ρ is a role of $\mathcal{Q}$, and the notation for basic sequences just introduced, the honesty rule for protocol $\mathcal{Q}$ is written as follows.

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

In words, if ϕ holds at the beginning of every role of $\mathcal{Q}$ and is preserved by all its basic sequences, then every honest principal executing protocol $\mathcal{Q}$ must satisfy ϕ. The side condition prevents free variables in the conclusion $\mathsf{Honest}(\hat{X}) \supset \phi$ from becoming bound in any hypothesis. As explained in [49, 50], this is a finitary rule, expressed in a slightly unusual way. For each protocol $\mathcal{Q}$, the corresponding instance of the honesty rule has a finite number of formulas in the antecedent, the exact number and form of each depending on the roles of $\mathcal{Q}$ and their basic sequences.

3.3.6 Soundness Theorem

The soundness theorem for this proof system is proved, by induction on the length of proofs, in Appendix C. We write $\Gamma \vdash \gamma$ if γ is provable from the formulas in Γ and any axiom or inference rule of the proof system except the honesty rule ($\mathbf{HON_Q}$ for any protocol $\mathcal{Q}$). We write $\Gamma \vdash_Q \gamma$ if γ is provable from the formulas in Γ, the basic axioms and inference rules of the proof system and the honesty rule for protocol $\mathcal{Q}$ (i.e., $\mathbf{HON_Q}$ but not $\mathbf{HON_{Q'}}$ for any $\mathcal{Q}' \neq \mathcal{Q}$). Here γ is either a modal formula or a basic formula (i.e., of the syntactic form Ψ or ϕ in Table 3.1).

Theorem 3.3.1. *If* $\Gamma \vdash_Q \gamma$*, then* $\Gamma \models_Q \gamma$*. Furthermore, if* $\Gamma \vdash \gamma$*, then* $\Gamma \models \gamma$*.*

Chapter 4

PCL Proof Methods

This chapter describes two methods of proof in PCL, which distinguish it from other approaches for proving security properties of network protocols. Section 4.1 discusses a method for compositional reasoning about protocols, while Section 4.2 presents an abstraction-instantiation method for reasoning about a class of protocol refinements.

4.1 Compositional Proof Method

In this section, we present a method for reasoning about compound protocols from their parts. In general terms, we address two basic problems in compositional security. The first may be called *additive combination* – we wish to combine protocol components in a way that accumulates security properties. For example, we may wish to combine a basic key exchange protocol with an authentication mechanism to produce a protocol for authenticated key exchange. The second basic problem is ensuring *nondestructive combination.* If two mechanisms are combined, each serving a separate purpose, then it is important to be sure that neither one degrades the security properties of the other. For example, if we add an alternative mode of operation to a protocol, then some party may initiate a session in one mode and simultaneously respond to another session in another mode, using the same public key or long-term key in both. Unless the modes are designed not to interfere, there may be an attack on the multi-mode protocol that would not arise if only one mode were possible. An interesting illustration of the significants of nondestructive combination is the construction in [72] which shows that for every security protocol there is another protocol that interacts with it insecurely.

Intuitively, additive combination is captured by a before-after formalism for reasoning about steps in protocol execution. Suppose P is a sequence of protocol steps, and ϕ and ψ are formulas asserting secrecy of some data, past actions of other principals, or other facts about a run of a protocol. The triple $\phi[P]_A\psi$ means that if ϕ is true before principal A does actions P, then ψ will be true afterwards. For example, the precondition might assert that A knows B's public key, the actions P allow A to receive a signed message and verify B's signature, and the postcondition may say that B sent the signed message that A received. The importance of before-after assertions is that we can combine assertions about individual protocol steps to derive properties of a sequence of steps: if $\phi[P]_A\psi$ and $\psi[P']_A\theta$, then $\phi[PP']_A\theta$. For example, an assertion assuming that keys have been successfully distributed can be combined with steps that do key distribution to prove properties of a protocol that distributes keys and uses them.

We ensure nondestructive combination, which is useful for reasoning about running older versions of a protocol concurrently with current versions (e.g., SSL 2.0 and SSL 3.0) and for verifying protocols like IKE [59] which contain a large number of sub-protocols, using invariance assertions. The central assertion in our reasoning system, $\Gamma \vdash \phi[P]_A\psi$, says that in any protocol satisfying the invariant Γ, the before-after assertion $\phi[P]_A\psi$ holds in any run (regardless of any actions by any dishonest attacker). Typically, our invariants are statements about principals that follow the rules of a protocol, as are the final conclusions. For example, an invariant may state that every honest principal maintains secrecy of its keys, where "honest" means simply that the principal only performs actions that are given by the protocol. A conclusion in such a protocol may be that if Bob is honest (so no one else knows his key), then after Alice sends and receives certain messages, Alice knows that she has communicated with Bob. Under the specific conditions described here, nondestructive combination occurs when two protocols are combined and neither violates the invariants of the other.

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

4.1.1 Composition Theorems

In this section, we define 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. For example, as explained in Section 3.1, the STS protocol is defined by two cords, one each for the initiator and responder role of the protocol.

Definition 4.1.1. *(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 3.3 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].

[1] A preliminary version of this result was presented in [36, 39]

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

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

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,

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

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.

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

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 4.1.2 below, we therefore have $\vdash_{Q_1|Q_2} \Gamma_1 \cup \Gamma_2$. From this and the formulas from step 3, we can conclude that the security properties of $\mathcal{Q}_1$ and $\mathcal{Q}_2$ are preserved under their parallel composition.

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

Lemma 4.1.2. *If* $\vdash_{Q_1} \phi$ *and* $\vdash_{Q_2} \phi$*, then* $\vdash_{Q_1|Q_2} \phi$*, where the last step in the proof of* ϕ *in both* $\mathcal{Q}_1$ *and* $\mathcal{Q}_2$ *uses the honesty rule and no previous step uses the honesty rule.*

Theorem 4.1.3. *If* $\vdash_{Q_1} \Gamma$ *and* $\Gamma \vdash \Psi$ *and* $\vdash_{Q_2} \Gamma$*, then* $\vdash_{Q_1|Q_2} \Psi$.

Definition 4.1.4. *(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 $\mathcal{Q}_2$.*

Definition 4.1.5. *(Sequential Composition of Cords) Given closed cords $r = (x_0 \ldots x_{\ell-1})[R]_X\langle u_0 \ldots u_{m-1}\rangle$, $s = (y_0 \ldots y_{m-1})[S]_Y\langle t_0 \ldots t_{n-1}\rangle$, their sequential composition is defined by*

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

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'.

It is clear that the sequential composition of protocols does not yield an 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. Further explanation of the sequential composition of two cords is given in Appendix A. We now illustrate the idea with an example. We consider two protocols: DH_0, the initial part of the Diffie-Hellman key exchange protocol, and CR, a signature-based challenge-response protocol. The protocols are written out as cords below.

$$\begin{aligned}
DH_0 &= \{ \quad (X\ Y)[(\nu x)]_X\langle X\ Y\ g^x\rangle \quad \} \\
CR &= \{ \quad (X\ Y\ x)[\langle \hat{X}, \hat{Y}, x\rangle(\hat{Y}, \hat{X}, y, z)(z/\{\!|x, y, \hat{X}|\!\}_{\overline{Y}})\langle \hat{X}, \hat{Y}, \{\!|x, y, \hat{Y}|\!\}_{\overline{X}}\rangle]_X\langle\ \rangle, \\
&\qquad (X\ Y\ y)[(\hat{Y}, \hat{X}, x)\langle \hat{X}, \hat{Y}, \{\!|x, y, \hat{Y}|\!\}_{\overline{X}}\rangle(\hat{Y}, \hat{X}, z)(z/\{\!|x, y, \hat{X}|\!\}_{\overline{Y}})]_X\langle\ \rangle \quad \}
\end{aligned}$$

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. 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.

$$\begin{aligned} ISO-9798-3 = \{ \quad & (X\ Y)[(\nu x)\langle \hat{X}, \hat{Y}, g^x\rangle(\hat{Y}, \hat{X}, y, z)(z/\{\!|g^x, y, \hat{X}|\!\}_{\overline{Y}}) \\ & \quad \langle \hat{X}, \hat{Y}, \{\!|g^x, y, \hat{Y}|\!\}_{\overline{X}}\rangle]_X\langle\ \rangle, \\ & (X\ Y)[(\nu y)(\hat{Y}, \hat{X}, x)\langle \hat{X}, \hat{Y}, \{\!|x, g^y, \hat{Y}|\!\}_{\overline{X}}\rangle(\hat{Y}, \hat{X}, z) \\ & \quad (z/\{\!|x, g^y, \hat{X}|\!\}_{\overline{Y}})]_X\langle\ \rangle \quad \} \end{aligned}$$

The sequencing rule, **S1** (see Table 3.4), is the main rule used to construct a modular correctness proof of a protocol that is a sequential composition of several smaller subprotocols. It gives us a way of sequentially composing two roles P and P' when the logical formula guaranteed by the execution of P, i.e., the post-condition of P, matches the pre-condition required in order to ensure that P' achieves some property. In addition, just like in parallel composition, it is essential that the composed protocols respect each other's invariants. Our methodology for proving properties of the sequential composition of two protocols involves the following steps.

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

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

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,

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

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.

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

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 $\mathbf{S1}$. 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 $\theta[P_1]\phi$ and $\phi[P_2]\psi$, we have:

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

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 4.1.6, 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.

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

Lemma 4.1.6. *If $\vdash_{Q_1} \phi$ and $\vdash_{Q_2} \phi$, then $\vdash_{Q_3} \phi$, where $\mathcal{Q}_3$ is a sequential composition of $\mathcal{Q}_1$ and $\mathcal{Q}_2$, and the last step in the proof of ϕ in both $\mathcal{Q}_1$ and $\mathcal{Q}_2$ uses the honesty rule and no previous step uses the honesty rule.*

Theorem 4.1.7. *If $\vdash_{Q_1} \Gamma_1$, $\Gamma_1 \vdash \theta[P_1]\phi$; $\vdash_{Q_2} \Gamma_2$, $\Gamma_2 \vdash \phi[P_2]\psi$; and $\vdash_{Q_1} \Gamma_2$, $\vdash_{Q_2} \Gamma_1$, then $\vdash_{Q_3} \theta[P_1P_2]\psi$, where $\mathcal{Q}_3$ is a sequential composition of $\mathcal{Q}_1$ and $\mathcal{Q}_2$.*

4.1.2 Illustrative Example

In this section, we use the protocol logic to formally prove properties of the ISO-9798-3 protocol from properties of its parts—the signature-based challenge-response protocol (CR) and the protocol based on Diffie-Hellman key exchange (DH_0), presented in the previous section. The logical proof follows the derivation step in Figure 2.3, where P_{16} is derived from C_1 and P_4 by applying a composition operation (note that P_{16} is ISO-9798-3 and C_1 and P_4 are DH_0 and CR respectively). We sketch the outline of the compositional proof in Section 4.1.2 and present the complete formal proofs in Section 4.1.2.

Compositional Proof Sketch

As illustrated in Section 4.1.1, the ISO-9798-3 protocol is constructed by a sequential composition of DH_0 and CR. Here, we describe the key secrecy property of DH_0 and the mutual authentication property of CR. 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: Our formulation of authentication is based on the concept of *matching conversations* [20] and is similar to the idea of proving authentication using *correspondence assertions* [127]. The same basic idea is also presented in [46] where it is referred to as *matching records of runs*. Simply put, it requires that whenever $\hat{A}$ and $\hat{B}$ accept each other's identities at the end of a run, their records of the run *match*, i.e., each message that $\hat{A}$ sent was received by $\hat{B}$ and vice versa, each send event happened before the corresponding receive event, and moreover the messages sent by each principal ($\hat{A}$ or $\hat{B}$) appear in the same order in both the records.

A complete proof of the mutual authentication property guaranteed by executing the CR protocol is presented in Table 4.2 in Section 4.1.2. We also discuss there the structure of the proof and identify a method for proving authentication results in the logic. The final property proved about the initiator role is of the form: *precondition [actions] postcondition*, where:

$$\begin{array}{lcl} precondition & = & \mathsf{Fresh}(X,x) \\ actions & = & [\langle \hat{X},\hat{Y},x\rangle(\hat{Y},\hat{X},y,z)(z/\{\!|x,y,\hat{X}|\!\}_{\overline{Y}})\langle \hat{X},\hat{Y},\{\!|x,y,\hat{Y}|\!\}_{\overline{X}}\rangle]_X \\ postcondition & = & \mathsf{Honest}(\hat{Y}) \supset \exists Y.\ \mathsf{ActionsInOrder}(\\ & & \quad \mathsf{Send}(X,\{\hat{X},\hat{Y},x\}), \\ & & \quad \mathsf{Receive}(Y,\{\hat{X},\hat{Y},x\}), \\ & & \quad \mathsf{Send}(Y,\{\hat{Y},\hat{X},\{y,\{\!|x,y,\hat{X}|\!\}_{\overline{Y}}\}\}), \\ & & \quad \mathsf{Receive}(X,\{\hat{Y},\hat{X},\{y,\{\!|x,y,\hat{X}|\!\}_{\overline{Y}}\}\})) \end{array}$$

The *actions* in the formula are the actions of the initiator cord of CR, given in section 4.1.1. There is an implicit universal quantification over the free variables in the formula (X, Y, and x), which correspond to the input parameters of the initiator cord. The *precondition* imposes constraints on the free variables. In this example, the requirement is that x is a fresh term generated in thread X. 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 (referred as ϕ_{auth} henceforth) is a formula capturing the notion of matching conversations. Intuitively, this formula means that after executing the actions in the initiator role purportedly with $\hat{Y}$, $\hat{X}$ is guaranteed that her record of the run matches that of $\hat{Y}$, 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).

The set of invariants used in this proof, Γ_2, contains only one formula (line (7) of Table 4.2).

$$\begin{array}{l} \Gamma_2 = \{\ \mathsf{Honest}(\hat{Y}) \supset (\\ \qquad (\ \diamondminus \mathsf{Send}(Y,x_0) \wedge \mathsf{Contains}(x_0,\{\!|x,y,\hat{X}|\!\}_{\overline{Y}}) \wedge \neg \diamondminus \mathsf{Fresh}(Y,x)\ ) \supset (\\ \qquad\quad x_0 = \{\hat{Y},\hat{X},\{y,\{\!|x,y,\hat{X}|\!\}_{\overline{Y}}\}\} \wedge \\ \qquad\quad \diamondminus (\mathsf{Send}(Y,\{\hat{Y},\hat{X},\{y,\{\!|x,y,\hat{X}|\!\}_{\overline{Y}}\}\}) \wedge \ominus \mathsf{Fresh}(Y,y)) \wedge \\ \qquad\quad \mathsf{After}(\mathsf{Receive}(Y,\{\hat{X},\hat{Y},x\}),\mathsf{Send}(Y,\{\hat{Y},\hat{X},\{y,\{\!|x,y,\hat{X}|\!\}_{\overline{Y}}\}\})) \\ \qquad)\ )\ \} \end{array}$$

Intuitively, this invariant states that whenever honest $\hat{Y}$ signs a term which is a triple with

$$\begin{array}{ll}
\mathbf{DH1} & \mathsf{Computes}(X, g^{ab}) \supset \mathsf{Has}(X, g^{ab}) \\
\mathbf{DH2} & \mathsf{Has}(X, g^{ab}) \supset \\
 & (\mathsf{Computes}(X, g^{ab}) \vee \exists m.(\diamondminus\, \mathsf{Receive}(X, m) \wedge \mathsf{Contains}(m, g^{ab}))) \\
\mathbf{DH3} & (\diamondminus\, \mathsf{Receive}(X, m) \wedge \mathsf{Contains}(m, g^{ab})) \supset \\
 & \exists Y, m'.(\mathsf{Computes}(Y, g^{ab}) \wedge \diamondminus\, \mathsf{Send}(Y, m') \wedge \mathsf{Contains}(m', g^{ab})) \\
\mathbf{DH4} & \mathsf{Fresh}(X, a) \supset \mathsf{Fresh}(X, g^a)
\end{array}$$

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

Table 4.1: Diffie-Hellman Axioms

the third component $\hat{X}$, and the first component was not freshly generated by $\hat{Y}$, then it is the case that this signature was sent as part of the second message of the CR protocol and $\hat{Y}$ must have previously received the first message of the protocol from $\hat{X}$. (Note that each message sent and received has the protocol-id in it. We omit these to improve readability).

Base Diffie Hellman Protocol, DH_0**:** The DH_0 protocol involves generating a fresh random number and computing its Diffie-Hellman exponential. It is therefore the initial part of the standard Diffie-Hellman key exchange protocol. In order to reason about the security property of this protocol, the term language and the protocol logic have to be enriched to allow reasoning about Diffie-Hellman computation. The terms $g(a)$ and $h(a, b)$, respectively representing the Diffie-Hellman exponential $g^a \, mod \, p$ and the Diffie-Hellman secret $g^{ab} \, mod \, p$, are added to the term language. To improve readability, we will use g^a and g^{ab} instead of $g(a)$ and $h(a, b)$. Table 4.1 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. $\mathbf{DH1}$ states that if X can compute the Diffie-Hellman secret, then she also possesses it. $\mathbf{DH2}$ 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. $\mathbf{DH3}$ 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. **DH4** captures the intuition that if a is fresh at some point of a run, then g^a is also fresh at that point. The property of the initiator role of the DH_0 protocol is given by the formula below. It is of the modal form *[actions] postcondition*.

$$[(\nu x)]_X\ \mathsf{HasAlone}(X, x) \wedge \mathsf{Fresh}(X, g^x)$$

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. In doing so, we follow the methodology for proving sequential composition results outlined in Section 4.1.1. Let us go back and look at the form of the logical formulas characterizing the initiator roles of DH_0 and CR. Denoting the initiator role actions of DH_0 and CR by $\mathbf{Init_{DH_0}}$ and $\mathbf{Init_{CR}}$ respectively, we have:

$$\Gamma_1 \vdash \mathsf{Start}(X)\ [\mathbf{Init_{DH_0}}]_X\ \mathsf{Fresh}(X, g^x) \tag{4.1}$$

$$\Gamma_2 \vdash \mathsf{Fresh}(X, x)\ [\mathbf{Init_{CR}}]_X\ \phi_{auth} \tag{4.2}$$

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.

$$\Gamma_2 \vdash \mathsf{Start}(X)\ [\mathbf{Init_{DH_0}}]_X\ \mathsf{Fresh}(X, g^x) \tag{4.3}$$

$$\Gamma_2 \vdash \mathsf{Fresh}(X, x)\ [\mathbf{Init_{CR}}]_X\ \phi_{auth} \tag{4.4}$$

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. This involves substituting g^x for x. We refer to the modified protocol as CR'. Since the validity of formulas is preserved under substitution, the following formula is valid.

$$\Gamma_2[g^x/x] \vdash \mathsf{Fresh}(X, g^x)\ [\mathbf{Init_{CR'}}]_X\ \phi_{auth}[g^x/x] \tag{4.5}$$

Note that the post-condition of (1) matches the pre-condition of (5). We can therefore compose the two formulas by applying the sequencing rule $\mathbf{S1}$. The resulting formula is:

$$\Gamma_2[g^x/x] \vdash \mathsf{Start}(X)\ [\mathbf{Init_{DH_0}}; \mathbf{Init_{CR'}}]_X\ \phi_{auth}[g^x/x] \tag{4.6}$$

The result of composing the two roles is that the freshly generated Diffie-Hellman exponential is substituted for the nonce in the challenge-response cord. The resulting role is precisely the initiator role of the ISO-9798-3 protocol. The formula above states that the mutual authentication property of CR is preserved by the composition process assuming that the invariants in Γ_2 are still satisfied. Finally, using the honesty rule, it is easily proved that DH_0 respects the environmental invariants in Γ_2 (Step 5). Therefore, from Lemma 4.1.6, we conclude that the sequential composition of DH_0 and CR, which is ISO-9798-3 , respects the invariants in Γ_2. This completes the compositional proof for the mutual authentication property.

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

$$\vdash \mathsf{Start}(X)\ [\mathbf{Init_{DH_0}}]_X\ \mathsf{HasAlone}(X, x) \tag{4.7}$$

$$\vdash \mathsf{HasAlone}(X, x)\ [\mathbf{Init_{CR'}}]_X\ \mathsf{HasAlone}(X, x) \tag{4.8}$$

Therefore, by applying the sequencing rule $\mathbf{S1}$ again, we have the secrecy condition for the ISO-9798-3 protocol:

$$\vdash \mathsf{Start}(X)\ [\mathbf{Init_{DH_0}}; \mathbf{Init_{CR'}}]_X\ \mathsf{HasAlone}(X, x) \tag{4.9}$$

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:

$$\begin{aligned}&\mathsf{Start}(X)\,[\mathbf{Init_{DH_0}};\mathbf{Init_{CR'}}]_X\,\mathsf{Honest}(\hat{\mathsf{Y}})\supset\\&\qquad\qquad\exists Y.\,\exists y.\,(\mathsf{Has}(X,g^{xy})\wedge(\mathsf{Has}(Z,g^{xy})\supset(Z=X\vee Z=Y)))\end{aligned}$$

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. The complete proof is presented in Table 4.3 in Section 4.1.2.

Formal Correctness Proofs of Protocols

The proof of the shared secret property of ISO-9798-3 is given in Table 4.3. This proof uses composition ideas and the structure of the proof has been discussed in the previous section. A complete proof of the authentication property for the initiator role of the challenge-response protocol ($\mathbf{Init_{CR}}$) is given in Table 4.2. We discuss below the structure of this proof and provide some insight on the proof technique used in proving authentication properties in this logic.

Proof Structure of Challenge-Response Protocol: The formal proof in Table 4.2 naturally breaks down into three parts:

- Lines (1)–(3) assert what actions were executed by Alice in the initiator role as well as the order in which those actions occurred. Specifically, in this part of the proof, the order in which Alice executed her send-receive actions is proved. Denoting the i-th message of the protocol by msg_i, we prove: ActionsInOrder(Send(A,msg_1), Receive(A,msg_2)). (As an expositional convenience, we refer to $\hat{X}$ and $\hat{Y}$ as Alice and Bob.)

- In lines (4)–(8), we first use the fact that the signatures of honest parties are unforgeable (axiom $\mathbf{VER}$), to conclude that Bob must have sent out some message containing his signature since Alice received Bob's signature in msg_2. The honesty rule is then used to infer that whenever Bob generates a signature of this form, he always sends it to Alice as part of msg_2 of the protocol and must have previously received msg_1 from Alice. Thus, the order in which Bob executed his actions are proved, i.e., ActionsInOrder(Receive(B,msg_1), Send(B,msg_2)).

- Finally, in lines (9)–(11), the temporal ordering rules are used to order the send-receive actions of Alice and Bob. Line (11) concludes that Bob must have received msg_1 after Alice sent it since msg_1 contains a fresh nonce. Line (12) uses the same argument for msg_2 sent by Bob. Finally, line (13) combines these two assertions to conclude that the following formula is true:
  ActionsInOrder(Send(A,msg_1), Receive(B,msg_1), Send(B,msg_2), Receive(A,msg_2)).
  This formula means that Alice and Bob have matching conversations.

This proof is 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.

$$
\begin{array}{rlr}
\mathbf{AA2}, \mathbf{P1} & \mathsf{Fresh}(X, x)[\mathbf{Init_{CR}}]_X & \\
& \quad \diamondminus\, (\mathsf{Send}(X, \{\hat{X}, \hat{Y}, x\}) \wedge \ominus \mathsf{Fresh}(X, x)) & (4.1) \\
\mathbf{AA1}, \mathbf{P1} & \mathsf{Fresh}(X, x)[\mathbf{Init_{CR}}]_X \diamondminus \mathsf{Verify}(X, \{\!|x, y, \hat{X}|\!\}_{\overline{Y}}) & (4.2) \\
\mathbf{AF1}, \mathbf{ARP} & \mathsf{Fresh}(X, x)[\mathbf{Init_{CR}}]_X \quad \mathsf{ActionsInOrder}(& \\
& \quad \mathsf{Send}(X, \{\hat{X}, \hat{Y}, x\}), & \\
& \quad \mathsf{Receive}(X, \{\hat{Y}, \hat{X}, y, \{\!|x, y, \hat{X}|\!\}_{\overline{Y}}\}), & \\
& \quad \mathsf{Send}(X, \{\hat{X}, \hat{Y}, \{\!|x, y, \hat{Y}|\!\}_{\overline{X}}\})) & (4.3) \\
(3), \mathbf{F1}, \mathbf{P1}, \mathbf{G2} & \mathsf{Fresh}(X, x)[\mathbf{Init_{CR}}]_X \neg \diamondminus \mathsf{Fresh}(Y, x) & (4.4) \\
\mathbf{VER} & \mathsf{Honest}(\hat{Y}) \wedge \diamondminus \mathsf{Verify}(X, \{\!|x, y, \hat{X}|\!\}_{\overline{Y}}) \supset & \\
& \; \exists Y. \exists x'. (\diamondminus \mathsf{Send}(Y, x') \wedge \mathsf{Contains}(x', \{\!|x, y, \hat{X}|\!\}_{\overline{Y}})) & (4.5) \\
(2), (5), \mathbf{P1}, \mathbf{G1-3} & \mathsf{Fresh}(X, x)[\mathbf{Init_{CR}}]_X \mathsf{Honest}(\hat{Y}) \supset & \\
& \; \exists Y. \exists x'. (\diamondminus \mathsf{Send}(Y, x') \wedge \mathsf{Contains}(x', \{\!|x, y, \hat{X}|\!\}_{\overline{Y}})) & (4.6) \\
\mathbf{HON} & \mathsf{Honest}(\hat{Y}) \supset (((\diamondminus \mathsf{Send}(Y, x_0) \wedge & \\
& \; \mathsf{Contains}(x_0, \{\!|x, y, \hat{X}|\!\}_{\overline{Y}}) \wedge \neg \diamondminus \mathsf{Fresh}(Y, x)) \supset & \\
& \quad (\; x_0 = \{\hat{Y}, \hat{X}, \{y, \{\!|x, y, \hat{X}|\!\}_{\overline{Y}}\}\} \wedge & \\
& \quad \diamondminus\, (\mathsf{Send}(Y, \{\hat{Y}, \hat{X}, \{y, \{\!|x, y, \hat{X}|\!\}_{\overline{Y}}\}\}) \wedge \ominus \mathsf{Fresh}(Y, y)) \wedge & \\
& \quad \mathsf{ActionsInOrder}(\mathsf{Receive}(Y, \{\hat{X}, \hat{Y}, x\}), & \\
& \qquad \mathsf{Send}(Y, \{\hat{Y}, \hat{X}, \{y, \{\!|x, y, \hat{X}|\!\}_{\overline{Y}}\}\}))))) & (4.7) \\
(4), (6), (7), \mathbf{G1-3} & \mathsf{Fresh}(X, x)[\mathbf{Init_{CR}}]_X \mathsf{Honest}(\hat{Y}) \supset & \\
& \; \exists Y.\; \diamondminus\, (\mathsf{Send}(Y, \{\hat{Y}, \hat{X}, \{y, \{\!|x, y, \hat{X}|\!\}_{\overline{Y}}\}\}) \wedge \ominus \mathsf{Fresh}(Y, y)) \wedge & \\
& \; \mathsf{After}(\mathsf{Receive}(Y, \{\hat{X}, \hat{Y}, x\}), & \\
& \; \mathsf{Send}(Y, \{\hat{Y}, \hat{X}, \{y, \{\!|x, y, \hat{X}|\!\}_{\overline{Y}}\}\})) & (4.8) \\
(1), \mathbf{AF2} & \mathsf{Fresh}(X, x)[\mathbf{Init_{CR}}]_X \diamondminus \mathsf{Receive}(Y, \{\hat{X}, \hat{Y}, x\})) \supset & \\
& \; \mathsf{After}(\mathsf{Send}(X, \{\hat{X}, \hat{Y}, x\}), \mathsf{Receive}(Y, \{\hat{X}, \hat{Y}, x\})) & (4.9) \\
(3), \mathbf{AF2} & \mathsf{Fresh}(X, x)[\mathbf{Init_{CR}}]_X \mathsf{Send}(Y, \{\hat{Y}, \hat{X}, \{y, \{\!|x, y, \hat{X}|\!\}_{\overline{Y}}\}\}) & \\
& \; \wedge \ominus \mathsf{Fresh}(Y, y) \supset \mathsf{After}(\mathsf{Send}(Y, \{\hat{Y}, \hat{X}, \{y, \{\!|x, y, \hat{X}|\!\}_{\overline{Y}}\}\}), & \\
& \quad \mathsf{Receive}(X, \{\hat{Y}, \hat{X}, y, \{\!|x, y, \hat{X}|\!\}_{\overline{Y}}\}) & (4.10) \\
(8), (9), (10), \mathbf{AF2} & \mathsf{Fresh}(X, x)[\mathbf{Init_{CR}}]_X \mathsf{Honest}(\hat{Y}) \supset & \\
& \; \exists Y.\; \mathsf{ActionsInOrder}(\mathsf{Send}(X, \{\hat{X}, \hat{Y}, x\}), \mathsf{Receive}(Y, \{\hat{X}, \hat{Y}, x\}), & \\
& \quad \mathsf{Send}(Y, \{\hat{Y}, \hat{X}, \{y, \{\!|x, y, \hat{X}|\!\}_{\overline{Y}}\}\}), & \\
& \quad \mathsf{Receive}(X, \{\hat{Y}, \hat{X}, y, \{\!|x, y, \hat{X}|\!\}_{\overline{Y}}\})) & (4.11)
\end{array}
$$

Table 4.2: Deductions of $\hat{X}$ executing $\mathbf{Init}$ role of Challenge-Response Protocol

$\mathbf{P3}$	$\mathsf{HasAlone}(X,x) \wedge \mathsf{Fresh}(X,g^x)[\mathbf{Init_{CR'}}]_X$ $\mathsf{HasAlone}(X,x)$	(4.1)
CR	$\mathsf{HasAlone}(X,x) \wedge \mathsf{Fresh}(X,g^x)[\mathbf{Init_{CR'}}]_X \mathsf{Honest}(\hat{Y}) \supset$ $\exists Y.\ \mathsf{ActionsInOrder}($ $\mathsf{Send}(X,\{\hat{X},\hat{Y},g^x\}),$ $\mathsf{Receive}(Y,\{\hat{X},\hat{Y},g^x\}),$ $\mathsf{Send}(Y,\{\hat{Y},\hat{X},\{n,\{\lvert g^x,n,\hat{X}\rvert\}_{\overline{Y}}\}\}),$ $\mathsf{Receive}(X,\{\hat{Y},\hat{X},n,\{\lvert g^x,n,\hat{X}\rvert\}_{\overline{Y}}\}))$	(4.2)
$\mathbf{HON}$	$\mathsf{Honest}(\hat{Y}) \wedge \diamondminus \mathsf{Send}(Y,\{\hat{Y},\hat{X},\{n,\{\lvert g^x,n,\hat{X}\rvert\}_{\overline{Y}}\}\})$ $\supset \exists y'.(n = g^{y'} \wedge \mathsf{HasAlone}(Y,y'))$	(4.3)
$(2),(3)$	$\mathsf{HasAlone}(X,x) \wedge \mathsf{Fresh}(X,g^x)[\mathbf{Init_{CR'}}]_X \mathsf{Honest}(\hat{Y}) \supset$ $\exists Y.\ \exists y.(n = g^y \wedge \mathsf{HasAlone}(Y,y))$	(4.4)
$\mathbf{AA1}, \mathbf{REC}, \mathbf{PROJ}, \mathbf{P1}$	$\mathsf{HasAlone}(X,x) \wedge \mathsf{Fresh}(X,g^x)[\mathbf{Init_{CR'}}]_X \mathsf{Has}(X,n)$	(4.5)
$(1),(4),(5),\mathbf{Computes}$	$\mathsf{HasAlone}(X,x) \wedge \mathsf{Fresh}(X,g^x)[\mathbf{Init_{CR'}}]_X \mathsf{Honest}(\hat{Y}) \supset$ $\exists Y.\ \exists y.(n = g^y \wedge \mathsf{Computes}(X,g^{xy}))$	(4.6)
$(1),(4),\mathbf{Computes}$	$\mathsf{HasAlone}(X,x) \wedge \mathsf{Fresh}(X,g^x)[\mathbf{Init_{CR'}}]_X \mathsf{Honest}(\hat{Y}) \supset$ $\exists Y.\ \exists y.(n = g^y \wedge$ $(\mathsf{Computes}(Z,g^{xy}) \supset (Z = X \vee Z = Y))))$	(4.7)
$(6),(7)$	$\mathsf{HasAlone}(X,x) \wedge \mathsf{Fresh}(X,g^x)[\mathbf{Init_{CR'}}]_X \mathsf{Honest}(\hat{Y}) \supset$ $\exists Y.\ \exists y.(n = g^y \wedge \mathsf{Computes}(X,g^{xy}) \wedge$ $(\mathsf{Computes}(Z,g^{xy}) \supset (Z = X \vee Z = Y))))$	(4.8)
$\mathbf{DH2}, \mathbf{DH3}$	$\mathsf{Has}(X,g^{xy}) \supset (\mathsf{Computes}(X,g^{xy}) \vee \exists Y,m'.$ $(\mathsf{Computes}(Y,g^{xy}) \wedge \diamondminus$ $\mathsf{Send}(Y,m') \wedge \mathsf{Contains}(m',g^{xy}))$	(4.9)
$\mathbf{HON}$	$\mathsf{Honest}(\hat{Y}) \supset (\mathsf{Computes}(Y,g^{xy}) \supset$ $\neg\exists m'.(\diamondminus\ \mathsf{Send}(Y,m') \wedge \mathsf{Contains}(m',g^{xy})))$	(4.10)
$(8),(9),(10),\mathbf{DH1}$	$\mathsf{HasAlone}(X,x) \wedge \mathsf{Fresh}(X,g^x)[\mathbf{Init_{CR'}}]_X \mathsf{Honest}(\hat{Y}) \supset$ $\exists Y.\ \exists y.(\mathsf{Has}(X,g^{xy}) \wedge$ $(\mathsf{Has}(Z,g^{xy}) \supset (Z = X \vee Z = Y)))$	(4.11)

Table 4.3: Deductions of $\hat{X}$ executing **Init** role of CR' protocol

4.2 Abstraction-Refinement Proof Method

In this section, we present proof methods for reasoning about protocol refinements. Recall that in a protocol refinement, a message or portion of a message is systematically refined by, for example, adding additional data or otherwise changing the data contained in one or more messages. For example, replacing a plaintext nonce by an encrypted nonce, in both the sending and receiving protocol roles, is a protocol refinement. While refinements seem to arise naturally in contemporary practical protocols [8, 74], they provide a challenge for formal reasoning. One reason is that refinements may involve replacement, and replacement of one expression by another does not have a clean formulation in standard mathematical logic. This immediate problem is solved by introducing protocol templates and decomposing term replacement into an abstraction step of selecting an appropriate template and an instantiation step that replaces template variables with protocol expressions. Another issue, addressed by associating hypotheses with a proof about a template, is that a refinement may not apply to all protocols, but only to protocols that satisfy certain hypotheses.

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.

After describing the formal framework, we illustrate the use of protocol templates with several examples. As an example of multiple instantiations of a single template, we prove an authentication property of a generic challenge-response protocol, and then show how to instantiate the template to ISO-9798-2, ISO-9798-3, or SKID3 [93]. As an example of one protocol that is an instance of two templates, we show how to reason about an identity-protection refinement using an authentication template and an encryption template. The third example compares two key exchange protocol templates, one that can be instantiated to the ISO-9798 family of protocols, and one that can be instantiated to STS [46] and SIGMA [74]. The first reflects the key exchange mechanism used in JFKi [8], while the second corresponds to that of IKE [59], JFKr [8], and IKEv2 [71]. While there has been considerable debate and discussion in the IETF community about the tradeoffs offered by these two protocols, previous analyses are relatively low-level and do not illustrate the design principles involved. However, it is possible to compare the authentication and non-repudiation properties of the two approaches by comparing the templates.

4.2.1 Cords and Protocol Logic with Function Variables

Like a program module containing functions that are not defined in the module, a cord may contain functions that are not given a specific meaning in the cord calculus. When a cord contains undefined functions, the cord cannot be executed as is, but can be used to define a set of runs if the function is replaced by a combination of defined operations. Since cords contain functions such as encryption and pairing, it is a simple matter to extend the syntax with additional function names. Since we will apply substitution for these function names, and implicitly quantify over their possible interpretations in the protocol logic, we refer to these function names as function variables. The mechanism for substituting an expression for a function variable, in a manner that treats function arguments correctly, is standard in higher-order logic. A simple explanation that does not involve lambda calculus or related machinery is given at the beginning of [54].

In a judgement

$$\mathcal{Q}, \Gamma \vdash \phi_1[P]_A\phi_2$$

where $\mathcal{Q}$ is a protocol containing function variables, P is one role or initial segment of a role of the protocol, and Γ denotes the set of assumed properties and invariants. The formulas in Γ may also contain function variables. The meaning of this judgement is that for every substitution that eliminates all function variables, any execution of the resulting protocol $\mathcal{Q}'$ respecting the resulting invariants Γ' satisfies the resulting formula $\phi_1'[P']_A\phi_2'$.

Theorem 4.2.1. (Soundness Theorem) *Protocol Composition Logic [49, 35, 50, 36] is sound for protocols and assertions containing function variables. Furthermore, substitution preserves semantic entailment and validity of formulas.*

The proof of this theorem follows the structure of the soundness proof of PCL without function variables (see Appendix C). The differences arise from the fact that the term language includes function variables which are universally quantified in the protocol logic. Therefore, the soundness proof for any axiom that contains an arbitrary term involves an additional step where the universal quantifier over a function variable is instantiated.

As a technical note for logicians, we also observe that since we do not have any comprehension principle for our logic, it is actually reducible to first-order logic by the standard method of treating function variables as first-order variables via an Apply function. Consequently, our higher-order protocol logic is no less tractable for automated theorem proving than the logic without function variables.

4.2.2 Abstraction and Refinement Methodology

Protocol Templates: A *protocol template* is a protocol that uses function variables. An example of a challenge-response protocol template using the informal trace notation is given below.

$$\begin{aligned}
&A \to B : m \\
&B \to A : n, F(B, A, n, m) \\
&A \to B : G(A, B, m, n)
\end{aligned}$$

Here, m and n are fresh nonces and F and G are function variables. Substituting cryptographic functions for F and G with the parameters appropriately filled in yields concrete protocols. For example, instantiating F and G to signatures yields the standard signature-based challenge-response protocol from the ISO-9798-3 family, whereas instantiating F and G to a keyed hash yields the SKID3 protocol.

Characterizing protocol concepts: Protocol templates provide a useful method for formally characterizing design concepts. Our methodology for formal proofs involves the following two steps.

1. Assuming properties of the function variables and some invariants, prove properties of the protocol templates. Formally,

$$\mathcal{Q}, \Gamma \vdash \phi_1[P]_A\phi_2$$

   Here, $\mathcal{Q}$ is an abstract protocol and P is a program for one role of the protocol. Γ denotes the set of assumed properties and invariants.

2. Instantiate the function variables to cryptographic functions and prove that the assumed properties and invariants are satisfied by the obtained protocol. Hence conclude that this protocol possesses the security property characterized by the protocol template.

$$\text{If } \mathcal{Q}' \vdash \Gamma', \text{ then } \mathcal{Q}' \vdash \phi_1'[P']_A\phi_2'$$

   Here, the primed versions of the protocol, hypotheses, etc. are obtained by applying the substitution σ used in the instantiation.

The correctness of the method is an immediate corollary of Theorem 4.2.1.

Combining protocol templates: Protocol templates can also be used to formalize the informal practice of protocol design by combining different mechanisms. The key observation is that if a concrete protocol is an instantiation of two different protocol templates, each instantiation respecting the assumed invariants associated with the template, then the concrete protocol has the security properties of both templates. Our methodology involves the following three steps.

1. Identify two protocol templates which guarantee certain security properties under some assumptions.

$$\mathcal{Q}_1, \Gamma_1 \vdash \phi_{11}[P_1]_A\phi_{21} \text{ and } \mathcal{Q}_2, \Gamma_2 \vdash \phi_{12}[P_2]_A\phi_{22}$$

   Here, $\mathcal{Q}_1$ and $\mathcal{Q}_2$ are protocol templates; P_1 and P_2 are respectively the programs corresponding to a specific role; and Γ_1 and Γ_2 denote the sets of assumed properties and invariants.

2. Find substitutions σ_1 and σ_2 such that the two instantiated protocols and roles are identical, i.e.,

$$\sigma_1\mathcal{Q}_1 = \sigma_2\mathcal{Q}_2 = \mathcal{Q}' \text{ and } \sigma_1 P_1 = \sigma_2 P_2 = P'$$

3. Prove that the instantiated protocol satisfies the hypotheses of both the protocol templates. Hence conclude that it inherits the security properties of both.

$$\text{If } \mathcal{Q}' \vdash \Gamma_1' \cup \Gamma_2', \text{then } \mathcal{Q}' \vdash (\phi_{11}' \wedge \phi_{12}')[P']_A(\phi_{21}' \wedge \phi_{22}')$$

   Here, the primed versions of the protocol, hypotheses, etc. are obtained by applying the substitutions σ_1 and σ_2 used in the instantiations.

4.2.3 Illustrative Examples

In this section, we present several examples illustrating the abstraction-instantiation methodology. The protocols considered include real-world protocols from the ISO and IKE families.

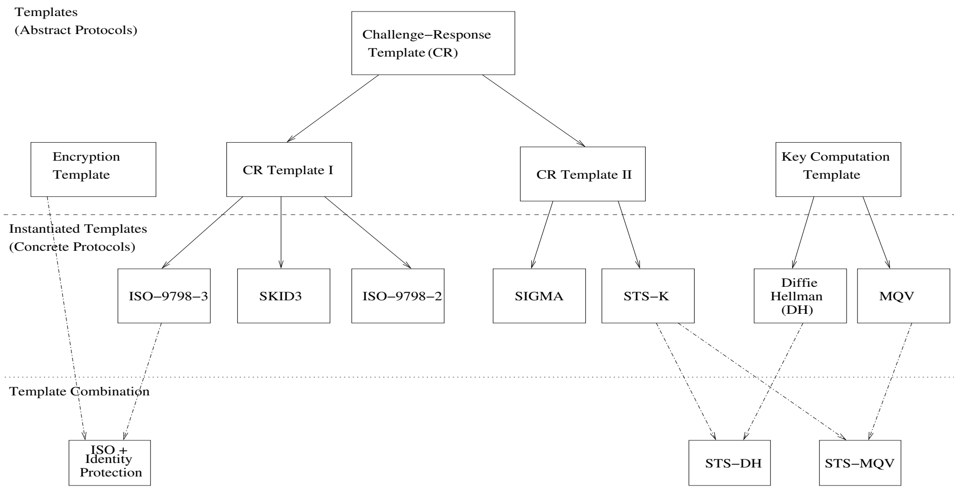

Figure 4.1: Illustrating the Methodology

Characterizing Protocol Concepts

A protocol template can be instantiated to multiple protocols. Security proofs of instances of a template follow from the proof of the template plus a (usually much simpler) proof that the instances satisfy the assumed hypotheses. In what follows, we work through an example demonstrating the approach.

Example: Challenge-Response Template In our first example, we characterize a challenge-response protocol template and then obtain three protocols: ISO-9798-2, ISO-9798-3, and SKID3 by appropriate substitutions. In doing so, we follow the two step methodology outlined in Section 4.2.2.

Step 1**:** The first step is to precisely define and characterize the protocol template. This involves defining the template (denoted Q_{CR} in the sequel) as a cord space, expressing the security property achieved, and identifying the set of assumptions under which the property holds. The programs for the initiator and responder roles of Q_{CR} is written out below in the notation of cords.

$$
\begin{aligned}
\mathbf{Init_{CR}} \;=\; & (\nu m)\ \langle\{\hat{A}, \hat{B}, m\}\rangle \\
& (\{\hat{B}, \hat{A}, n, F(\hat{B}, \hat{A}, n, m)\}) \\
& \langle\{\hat{A}, \hat{B}, G(\hat{A}, \hat{B}, m, n)\}\rangle
\end{aligned}
$$

$$
\begin{aligned}
\mathbf{Resp_{CR}} \;=\; & (\{\hat{Y}, \hat{B}, y\}) \\
& (\nu x)\ \langle\{\hat{B}, \hat{Y}, x, F(\hat{B}, \hat{Y}, x, y)\}\rangle \\
& (\{\hat{Y}, \hat{B}, G(\hat{Y}, \hat{B}, y, x)\})
\end{aligned}
$$

Here, F and G are function variables. Under a set of assumptions (Γ_{CR}) about these variables, we prove an authentication property for the initiator of the protocol using the logic (see Table 4.4 for the complete formal proof).

$$Q_{CR}, \Gamma_{CR} \vdash [\mathbf{Init_{CR}}]_A\ \mathsf{Honest}(\hat{A}) \wedge \mathsf{Honest}(\hat{B}) \supset \phi_{auth}$$

Intuitively, this formula means that if A executed a session of Q_{CR} supposedly with B and both of them are honest (implying that they strictly follow the protocol and do not, for example, reveal their private keys), then the authentication property expressed by the formula ϕ_{auth} holds in the resulting state. ϕ_{auth} specifies an authentication property for the initiator based on the concept of *matching conversations* [46]. Simply put, it requires that whenever A completes a session supposedly with B, both A and B have consistent views of the run, i.e., they agree on the content and order of the messages exchanged. Formally,

$$
\begin{aligned}
\phi_{auth} \;\equiv\; & \exists B.(\mathsf{ActionsInOrder}(\\
& \quad \mathsf{Send}(A, \{\hat{A}, \hat{B}, m\}), \\
& \quad \mathsf{Receive}(B, \{\hat{A}, \hat{B}, m\}), \\
& \quad \mathsf{Send}(B, \{\hat{B}, \hat{A}, n, F(\hat{B}, \hat{A}, n, m)\}), \\
& \quad \mathsf{Receive}(A, \{\hat{B}, \hat{A}, n, F(\hat{B}, \hat{A}, n, m)\}), \\
& \quad \mathsf{Send}(A, \{\hat{A}, \hat{B}, G(\hat{A}, \hat{B}, m, n)\}), \\
& \quad \mathsf{Receive}(B, \{\hat{A}, \hat{B}, G(\hat{A}, \hat{B}, m, n)\})))
\end{aligned}
$$

The set of assumptions Γ_{CR} used to prove the authentication property consists of the following four logical formulas:

$$
\begin{aligned}
\gamma_1 \ \equiv\ & \mathsf{Computes}(X, F(\hat{B}, \hat{A}, n, m)) \supset \\
& \quad (\exists A'.X = A') \vee (\exists B'.X = B') \\
\gamma_2 \ \equiv\ & \diamondminus\mathsf{Fresh}(Z, x) \supset \\
& \quad \neg\mathsf{Contains}(\{\hat{X}, \hat{Y}, x\}, F(\hat{B}, \hat{A}, n, m)) \\
\gamma_3 \ \equiv\ & \diamondminus\mathsf{Fresh}(Z, x) \wedge \diamondminus\mathsf{Fresh}(W, y) \wedge \\
& \mathsf{Contains}(\{\hat{X}, \hat{Y}, G(\hat{X}, \hat{Y}, x, y)\}, F(\hat{B}, \hat{A}, n, m)) \supset \\
& \quad \hat{X} = \hat{B} \wedge \hat{Y} = \hat{A} \wedge n = y \wedge m = x \\
\gamma_4 \ \equiv\ & \diamondminus\mathsf{Fresh}(Z, x) \wedge \diamondminus\mathsf{Fresh}(W, y) \wedge \\
& \mathsf{Contains}(\{\hat{X}, \hat{Y}, x, F(\hat{X}, \hat{Y}, x, y)\}, F(\hat{B}, \hat{A}, n, m)) \supset \\
& \quad \hat{X} = \hat{B} \wedge \hat{Y} = \hat{A} \wedge n = x \wedge m = y
\end{aligned}
$$

Informally, assumption γ_1 states that the function F is hard to compute: only agents $\hat{A}$ and $\hat{B}$ can compute $F(\hat{B}, \hat{A}, n, m)$ (more precisely, if some session X has enough information to compute $F(\hat{B}, \hat{A}, n, m)$ then X is either a session of agent $\hat{A}$ or agent $\hat{B}$). The honesty of $\hat{A}$ and $\hat{B}$ is a part of the premise and have been omitted to improve readability. In a concrete protocol, this assumption can be satisfied by, for example, instantiating F to a signature. The other assumptions impose syntactic constraints on F and G. For example, γ_2 implies that $F(\hat{B}, \hat{A}, n, m)$ cannot be mistaken for a nonce. This obviates certain type confusion attacks. γ_4 implies that F depends on the value of all four parameters.

Proof Structure of Challenge-Response Template: A complete proof of the authentication property for the initiator role of the challenge-response template is given in Table 4.4. The leftmost column identifies the axioms and rules used in the corresponding step, where the naming convention follows [36] and Section 4.2.4. The proof naturally breaks down into four parts:

- Line (1) asserts what actions were executed by Alice in the initiator role. Specifically, we can conclude that Alice has received a message msg containing $F(\hat{B}, \hat{A}, n, m)$.

$F(X,Y,x,y) \equiv E_{K_{XY}}(x,y,X)$	$H_{K_{XY}}(x,y,X)$	$SIG_X(x,y,Y)$
$G(X,Y,x,y) \equiv E_{K_{XY}}(y,x)$	$H_{K_{XY}}(y,x,X)$	$SIG_X(y,x,Y)$
$A \rightarrow B : m$	$A \rightarrow B : m$	$A \rightarrow B : m$
$B \rightarrow A : n, E_{K_{AB}}(n,m,B)$	$B \rightarrow A : n, H_{K_{AB}}(n,m,B)$	$B \rightarrow A : n, SIG_B(n,m,A)$
$A \rightarrow B : E_{K_{AB}}(n,m)$	$A \rightarrow B : H_{K_{AB}}(n,m,A)$	$A \rightarrow B : SIG_A(n,m,B)$
ISO 9798-2	SKID3	ISO 9798-3

Figure 4.2: Instantiations of the Challenge-Response template

- In lines (2)–(9), we track the source of term $F(\hat{B}, \hat{A}, n, m)$ received by Alice. Since Alice received a message containing $F(\hat{B}, \hat{A}, n, m)$, there must be a process which computed that term and sent it out. Using the assumption γ_1, we can conclude that only Alice or Bob could have computed $F(\hat{B}, \hat{A}, n, m)$. From assumptions $\gamma_2, \gamma_3, \gamma_4$, we can deduce that Alice did not send $F(\hat{B}, \hat{A}, n, m)$. Therefore, Bob must have sent a message msg' containing $F(\hat{B}, \hat{A}, n, m)$.

- In lines (10)–(13), we use the honesty rule, and assumptions $\gamma_2, \gamma_3, \gamma_4$ to conclude that Bob must have sent $F(\hat{B}, \hat{A}, n, m)$ as part of the second message of the responder role. Therefore, Bob must have received a corresponding first message in the past. Also, using γ_4, we can conclude that Bob is in a session with Alice.

- Finally, in lines (14)–(19), the temporal ordering rules are used to establish a total ordering among the send-receive actions of Alice and Bob. Line (17) concludes that Bob must have received $msg1$ after Alice sent it since $msg1$ contains a fresh nonce. Line (18) uses the same argument for $msg2$ sent by Bob. Finally, line (19) uses the transitivity axiom to conclude that the authentication formula ϕ_{auth} is true.

This completes the characterization of the protocol template. We are now ready to move on to Step 2.

Step 2: In this step, we instantiate the protocol template to three well known protocols from the ISO family. The substitutions for the function variables and the resulting protocols are shown in Figure 4.2. ISO 9798-2, SKID3, and ISO 9798-3 [93] respectively use symmetric key encryption with a pre-shared key, keyed hash and signatures to instantiate F and G. These substitutions respect the assumed invariants in Γ_{CR}. For example, γ_1 is satisfied by signatures since the signature can be computed only by an agent who has the corresponding private key. (The formal proofs follow immediately from logical axioms and are omitted.) We can therefore conclude that all three protocols guarantee the authentication property characterized by the protocol template.

Note that we have only proved the authentication property for the initiator in the protocol. To complete the proof of the mutual authentication property, we need to prove a formula analogous to ϕ_{auth} for the responder. This is achieved using symmetric assumptions about function variable G. All three instantiations satisfy these additional assumptions.

Combining Protocol Templates

A refinement operation, when applied to a protocol, adds an additional security property while preserving the original properties. Examples of refinement operations considered in [35] include replacing signatures by encrypted signatures to provide identity protection and replacing fresh Diffie-Hellman exponentials by a pair consisting of a stale exponential and a fresh nonce, thereby enabling reuse of exponentials and hence greater computational efficiency. The methodology for combining protocol templates, described in Section 4.2.2, provides a way to formally reason about a broad class of refinements including the two just mentioned. Below we illustrate the general method by examining the identity protection refinement in some detail.

Example: Identity Protection Refinement In this example, we start with a signature based protocol, ISO-9798-3, that provides mutual authentication. We apply the identity protection refinement to it, which involves replacing the signatures by encrypted signatures using a shared key. The intention is to prevent adversaries from observing signatures since they can reveal identities of communicating peers. Our goal is to prove that this refinement step is correct, i.e., it does indeed guarantee that the resulting protocol provides identity protection, while preserving the mutual authentication property of the original protocol. We identify two templates: Q_{CR}, the challenge-response template described in the previous section, and Q_{ENC} described below, which provides a form of secrecy. The aim now is to prove that the protocol obtained after the refinement step is an invariant respecting instance of both these templates and the terms protected by the secrecy template are precisely the signatures.

Step 1**:** The Q_{CR} template has been defined and characterized in an earlier section. Here, we do the same for the Q_{ENC} template. Using the informal arrows-and-messages diagram, the template can be described as follows.

$$\begin{array}{l} A \rightarrow B : m \\ B \rightarrow A : n, E_{K_{AB}}(H(B, A, n, m)) \\ A \rightarrow B : E_{K_{AB}}(I(A, B, m, n)) \end{array}$$

The programs for the initiator and responder roles of Q_{ENC} is written out below in the notation of cords.

$$\begin{aligned} \mathbf{Init_{ENC}} \;=\; & (\nu m)\, \langle\{\hat{A}, \hat{B}, m\}\rangle \\ & (\{\hat{B}, \hat{A}, n, E_{K_{AB}}(H(\hat{B}, \hat{A}, n, m))\}) \\ & \langle\{\hat{A}, \hat{B}, E_{K_{AB}}(I(\hat{A}, \hat{B}, m, n))\}\rangle \end{aligned}$$

$$\begin{aligned} \mathbf{Resp_{ENC}} \;=\; & (\{\hat{Y}, \hat{B}, y\}) \\ & (\nu x)\, \langle\{\hat{B}, \hat{Y}, x, E_{K_{BY}}(H(\hat{B}, \hat{Y}, x, y))\}\rangle \\ & (\{\hat{Y}, \hat{B}, E_{K_{BY}}(I(\hat{Y}, \hat{B}, y, x))\}) \end{aligned}$$

Here, H and I are function variables. Under a set of assumptions (Γ_{ENC}) about these variables, we prove that the term $H(\hat{B}, \hat{A}, n, m)$ remains secret: it is known only to A and B. Formally,

$$\begin{aligned} Q_{ENC}, \Gamma_{ENC} \vdash \; & [\mathbf{Init_{ENC}}]_A \; \mathsf{Honest}(\hat{A}) \wedge \mathsf{Honest}(\hat{B}) \supset \\ & \phi_{secret} \end{aligned}$$

Intuitively, this formula means that if A executed a session of Q_{ENC} supposedly with

$F(X,Y,x,y) \equiv E_{K_{XY}}(SIG_X(x,y))$
$G(X,Y,x,y) \equiv E_{K_{XY}}(SIG_X(y,x))$

$H(X,Y,x,y) \equiv SIG_X(x,y)$
$I(X,Y,x,y) \equiv SIG_X(y,x)$

$A \rightarrow B : m$
$B \rightarrow A : n, E_{K_{AB}}(SIG_B(n,m))$
$A \rightarrow B : E_{K_{AB}}(SIG_A(n,m))$

Figure 4.3: Protocol that is an instantiation of both CR and ENC templates

B and both of them are honest, then the secrecy property expressed by the formula ϕ_{secret} holds in the resulting state. ϕ_{secret} specifies the secrecy property for the term $H(\hat{B},\hat{A},n,m)$. Formally,

$$\begin{aligned} \phi_{secret} \;\equiv\; & \exists B.(\mathsf{Has}(X, H(\hat{B},\hat{A},n,m)) \supset \\ & (X = A \vee X = B)) \end{aligned}$$

The set of assumptions Γ_{ENC} used to prove the authentication property consists of the following four logical formulas:

$$\begin{aligned}
\delta_1 \equiv\; & \mathsf{Computes}(X, H(\hat{B},\hat{A},n,m)) \supset \exists B.X = B \\
\delta_2 \equiv\; & \diamondminus\mathsf{Fresh}(Z,x) \supset \\
& \quad \neg\mathsf{Contains}(\{\hat{X},\hat{Y},x\}, H(\hat{B},\hat{A},n,m)) \\
\delta_3 \equiv\; & \diamondminus\mathsf{Fresh}(Z,x) \wedge \diamondminus\mathsf{Fresh}(W,y) \wedge \\
& \mathsf{Contains}(\{\hat{X},\hat{Y},E_{K_{XY}}(I(\hat{X},\hat{Y},x,y))\}, H(\hat{B},\hat{A},n,m)) \\
& \supset \quad (\hat{X} = \hat{B} \wedge n = y \wedge m = x) \\
\delta_4 \equiv\; & \diamondminus\mathsf{Fresh}(Z,x) \wedge \diamondminus\mathsf{Fresh}(W,y) \wedge \\
& \mathsf{Contains}(\{\hat{X},\hat{Y},x,E_{K_{XY}}(H(\hat{X},\hat{Y},x,y))\}, \\
& \qquad H(\hat{B},\hat{A},n,m)) \supset \quad (\hat{X} = \hat{B} \wedge n = x \wedge m = y)
\end{aligned}$$

These formulas capture simple ideas, e.g., $H(\hat{B},\hat{A},n,m)$ (e.g. $B's$ signature) can be computed only by B and certain syntactic constraints, e.g., a signature is not a subterm of a nonce.

Step 2**:** The second step is to find substitutions σ_1 and σ_2 such that both the templates (Q_{CR} and Q_{ENC}) instantiate to the same real protocol. The desired substitutions are shown in Figure 4.3. σ_1 is on the left, σ_2 is on the right and the instantiated protocol is in the middle of the figure.

Step 3**:** The final step is to verify that the instantiated protocol satisfies the union of the hypotheses in Γ_{CR} and Γ_{ENC}. This follows easily from the properties of signature and encryption under symmetric key as expressed in the logic and the syntactic structure of the protocol. We can therefore conclude that the identity protection refinement operation as applied here is correct, i.e., it adds the identity protection property while preserving the original properties of the protocol (which in this case is mutual authentication).

Authenticated Key-Exchange Templates

An important use of protocol templates is to underpin basic principles used in designing classes of protocols and to bring out subtle tradeoffs offered by various protocol families. In this section, we examine two families of authenticated key exchange protocols. The first template, AKE1, generalizes a family of protocols in which authentication is achieved by explicitly embedding the intended recipient's identity inside authenticators in messages. This family includes the ISO-9798-3 key exchange protocol and related protocols including the core JFKi protocol. The second template, AKE2, generalizes a family of protocols where agents authenticate each other using a combination of signatures and a proof of possession of the Diffie-Hellman shared secret computed during the execution of the protocol. This family includes STS, SIGMA, and the core of the IKE and JFKr protocols. Part of the reason these two families are interesting is that they were both candidates for the recently proposed IKEv2 protocol and there has been considerable discussion and debate in the IETF community about the tradeoffs offered by the two designs. The use of templates to characterize the two families sheds light on the subtle difference between the authentication, non-repudiation and identity protection guarantees associated with the two sets of protocols.

Template AKE1: Using the informal arrows-and-messages diagram, the authenticated key-exchange template AKE1 can be described as follows.

$$A \rightarrow B : A, g^a$$
$$B \rightarrow A : g^b, F(B, A, g^b, g^a)$$
$$A \rightarrow B : G(A, B, g^a, g^b)$$

For this template, we are able to prove both secrecy and authentication for the initiator role:

$$Q_{AKE1}, \Gamma_{AKE1} \vdash [\mathbf{Init_{AKE1}}]_A \mathsf{Honest}(\hat{A}) \wedge \mathsf{Honest}(B) \supset \phi_{auth} \wedge \phi_{shared-secret}$$

The formula ϕ_{auth} describes an authentication property for the initiator based on matching conversations, while formula $\phi_{shared-secret}$ states that A and B are the only two sessions which know the Diffie-Hellman secret g^{ab}. The set of assumptions Γ_{AKE1} is similar to Γ_{CR} in Section 4.2.3:

$$\begin{aligned}
\epsilon_1 \equiv\, & \mathsf{Computes}(X, F(\hat{B}, \hat{A}, g^b, g^a)) \supset \exists B'. X = B' \\
\epsilon_2 \equiv\, & \diamondminus \mathsf{Fresh}(Z, x) \supset \\
& \quad \neg \mathsf{Contains}(\{\hat{X}, \hat{Y}, g^x\}, F(\hat{B}, \hat{A}, g^b, g^a)) \\
\epsilon_3 \equiv\, & \diamondminus \mathsf{Fresh}(Z, x) \wedge \diamondminus \mathsf{Fresh}(W, y) \wedge \\
& \mathsf{Contains}(\{\hat{X}, \hat{Y}, G(\hat{X}, \hat{Y}, g^x, g^y)\}, F(\hat{B}, \hat{A}, g^b, g^a)) \supset \\
& \quad (\hat{X} = \hat{B} \wedge \hat{Y} = \hat{A} \wedge g^b = g^y \wedge g^a = g^x) \\
\epsilon_4 \equiv\, & \diamondminus \mathsf{Fresh}(Z, x) \wedge \diamondminus \mathsf{Fresh}(W, y) \wedge \\
& \mathsf{Contains}(\{\hat{X}, \hat{Y}, g^y, F(\hat{X}, \hat{Y}, g^x, g^y)\}, F(\hat{B}, \hat{A}, g^b, g^a)) \supset \\
& \quad (\hat{X} = \hat{B} \wedge \hat{Y} = \hat{A} \wedge g^b = g^x \wedge g^a = g^y)
\end{aligned}$$

We prove that the ISO-9798-3 key exchange protocol satisfies the set of assumptions, Γ_{AKE1}, and therefore provides similar authentication and secrecy guarantees. However, STS, SIGMA and their variants do not satisfy Γ_{AKE1}. Specifically, the assumption ϵ_4 fails since the intended recipient's identity is not embedded inside an authenticator in the second message of the protocol. The way the proof fails leads us to a run which provides a

$A \rightarrow B : g^a$	g^a	g^a
$B \rightarrow A : g^b, SIG_B(g^b, g^a, A)$	$g^b, E_{g^{ab}}(SIG_B(g^a, g^b))$	$g^b, SIG_B(g^b, g^a), H_{g^{ab}}(B)$
$A \rightarrow B : SIG_A(g^a, g^b, B)$	$E_{g^{ab}}(SIG_A(g^b, g^a))$	$SIG_A(g^b, g^a), H_{g^{ab}}(A)$
ISO 9798-3 Key exchange	STS	Basic SIGMA

Figure 4.4: Instantiations of authenticated key-exchange templates

counterexample to the strong authentication property. This works for both STS and SIGMA and the run is essentially similar to the "attack" on STS first demonstrated by Lowe in [79].

Template AKE2: Using the informal arrows-and-messages diagram, AKE2 can be described as follows.

$$
\begin{aligned}
&A \rightarrow B : g^a \\
&B \rightarrow A : g^b, F(B, g^b, g^a), F'(B, g^{ab}) \\
&A \rightarrow B : G(A, g^a, g^b), G'(A, g^{ab})
\end{aligned}
$$

Informally, the purpose of F is to ensure that B is a session with parameters g^a and g^b, while F' proves that B has the shared secret g^{ab}. More precisely, using the set of assumptions about the function variables Γ_{AKE2}, it is possible to prove that this protocol template provides a form of authentication – matching conversations for the responder:

$$
Q_{AKE2}, \Gamma_{AKE2} \vdash [\mathbf{Init_{AKE2}}]_B \mathsf{Honest}(\hat{A}) \wedge \mathsf{Honest}(\hat{B}) \supset
$$

$$
\phi_{auth} \wedge \phi_{shared-secret}
$$

The set of assumptions Γ_{AKE2} is:

$$
\begin{aligned}
\eta_1 \equiv\ & \mathsf{Computes}(X, G(\hat{A}, g^a, g^b)) \supset \exists A'.X = A' \\
\eta_2 \equiv\ & \mathsf{Computes}(X, F'(\hat{Z}, g^{ab})) \supset \mathsf{Has}(X, g^{ab}) \\
\eta_3 \equiv\ & \diamondminus\mathsf{Fresh}(Z, x) \supset \\
& \quad \neg\mathsf{Contains}(\{\hat{X}, \hat{Y}, g^x\}, G(\hat{A}, g^a, g^b)) \\
\eta_4 \equiv\ & \diamondminus\mathsf{Fresh}(Z, x) \wedge \diamondminus\mathsf{Fresh}(W, y) \supset \\
& \quad \neg\mathsf{Contains}(\{\hat{X}, \hat{Y}, g^y, F(\hat{X}, g^x, g^y), F'(\hat{X}, g^{xy})\}, \\
& \qquad\qquad G(\hat{A}, g^a, g^b)) \\
\eta_5 \equiv\ & \diamondminus\mathsf{Fresh}(Z, x) \wedge \diamondminus\mathsf{Fresh}(W, y) \wedge \\
& \mathsf{Contains}(\{\hat{X}, \hat{Y}, G(\hat{X}, g^x, g^y), G'(\hat{X}, g^{xy})\}, G(\hat{A}, g^a, g^b)) \\
& \quad \supset \quad (\hat{X} = \hat{A} \wedge \wedge g^b = g^x \wedge g^a = g^y)
\end{aligned}
$$

However, as mentioned before, this class of protocols does not have the matching conversations based authentication property for the initiator.

Design Tradeoffs Template AKE1 provides a stronger form of authentication: matching conversations for both initiator and responder whereas AKE2 only provides matching conversations for responder. Therefore, in AKE1, the initiator is required to reveal his identity in the first message of the protocol, and hence instances of this template cannot provide identity protection against active attackers for the initiator. Additionally, assumptions ϵ_1 and ϵ_3 together imply that when the function variable F is instantiated to the signature function, A has a non-repudiable proof of communication with B. In the logic, such a proof just uses terms that A possesses (given by the Has predicate) and the honesty of B.

One of the design goals for instances of the AKE2 template such as SIGMA [74] was to provide identity protection for the initiator. Since the initiator A can only reveal his identity in the third message of the protocol, and cannot be sure that the responder B knows he is talking to A until B receives the last message, template AKE2 does not provide the strong authentication property for the initiator. A counterexample run is shown below:

$$\begin{aligned}
&A \rightarrow I(B) : g^a \\
&I(C) \rightarrow B : g^a \\
&B \rightarrow I(C) : g^b, F(B, g^b, g^a), F'(B, g^{ab}) \\
&I(B) \rightarrow A : g^b, F(B, g^b, g^a), F'(B, g^{ab}) \\
&A \rightarrow I(B) : G(A, g^a, g^b), G'(A, g^{ab})
\end{aligned}$$

In this scenario, A believes he has completed a session with B, while B is waiting for the third message, thinking that he is engaged in a session with C. This counterexample also shows that A's transcript of the run cannot be used to prove that B was involved in the protocol. Hence this template does not provide non-repudiation even when F is instantiated to the signature function.

4.2.4 Protocol Logic Extensions

Most of the predicates and axioms of protocol logic used in the formal proofs in this chapter are presented in Chapter 3. The main extensions include the Computes predicate and some axioms for reasoning about symmetric encryption and cryptographic hash. The definition of Computes in terms of Has and the axioms it satisfies is presented in Table 4.5. Intuitively, $\mathbf{CP2}$ says that there are two ways an agent can possess some term: she can construct it from its components or she can receive it as a part of some message. Axiom $\mathbf{CP3}$ says that every term that appears on the network has a source: it originated from some process that actually computed the term. One use of Computes is to reason about the source of a term. This is useful for reasoning about authentication properties of protocols. A second use is to capture hardness assumptions about cryptographic primitives, which is important for reasoning about secrecy. For example, we postulate that the only way to compute a Diffie-Hellman secret is to have one exponent and the other exponential. Similarly, in order to compute an encrypted message, it is essential to possess the key and the plaintext.

$\mathbf{AA1}, \mathbf{T1}, \mathbf{P1}$	$[\mathbf{Init_{CR}}]_A \diamondminus \mathsf{Receive}(A, msg) \wedge \mathsf{Contains}(msg, F(\hat{B}, \hat{A}, n, m))$	(4.1)
$\mathbf{CP3}, (1)$	$[\mathbf{Init_{CR}}]_A\ \exists X.\exists msg'.(\mathsf{Computes}(X, F(\hat{B}, \hat{A}, n, m)) \wedge$ $\diamondminus \mathsf{Send}(X, msg') \wedge \mathsf{Contains}(msg', F(\hat{B}, \hat{A}, n, m)))$	(4.2)
Γ_{CR}	$\mathsf{Computes}(X, F(\hat{B}, \hat{A}, n, m)) \supset (\exists A'.X = A') \vee (\exists B'.X = B')$	(4.3)
$\mathbf{HON}$	$\mathsf{Honest}(\hat{Y}) \wedge \diamondminus \mathsf{Send}(Y, msg') \supset \exists X.\exists x.\exists y.(\diamondminus \mathsf{Fresh}(Y, y) \wedge$ $(msg' = \{\hat{Y}, \hat{X}, y\} \vee$ $msg' = \{\hat{Y}, \hat{X}, y, F(\hat{Y}, \hat{X}, y, x)\} \vee$ $msg' = \{\hat{Y}, \hat{X}, G(\hat{Y}, \hat{X}, y, x)\})$	(4.4)
Γ_{CR}	$\neg\mathsf{Contains}(\{\hat{A}, \hat{X}, m'\}, F(\hat{B}, \hat{A}, n, m))$	(4.5)
Γ_{CR}	$\mathsf{Contains}(\{\hat{A}, \hat{X}, G(\hat{A}, \hat{X}, m', x)\}, F(\hat{B}, \hat{A}, n, m)) \supset \hat{A} = \hat{B}$	(4.6)
Γ_{CR}	$\mathsf{Contains}(\{\hat{A}, \hat{X}, m', F(\hat{A}, \hat{X}, m', x)\}, F(\hat{B}, \hat{A}, n, m)) \supset \hat{A} = \hat{B}$	(4.7)
$(4-7)$	$\mathsf{Honest}(\hat{A}) \wedge \diamondminus \mathsf{Send}(A', msg') \wedge \mathsf{Contains}(msg', F(\hat{B}, \hat{A}, n, m)) \supset$ $\hat{A} = \hat{B}$	(4.8)
$(2-3), (8)$	$[\mathbf{Init_{CR}}]_A\ \mathsf{Honest}(\hat{A}) \supset \exists B.\exists msg'.(\mathsf{Computes}(B, F(\hat{B}, \hat{A}, n, m)) \wedge$ $\diamondminus \mathsf{Send}(B, msg') \wedge \mathsf{Contains}(msg', F(\hat{B}, \hat{A}, n, m)))$	(4.9)
Γ_{CR}	$\neg\mathsf{Contains}(\{\hat{B}, \hat{X}, n'\}, F(\hat{B}, \hat{A}, n, m))$	(4.10)
Γ_{CR}	$\mathsf{Contains}(\{\hat{B}, \hat{X}, G(\hat{B}, \hat{X}, n', x)\}, F(\hat{B}, \hat{A}, n, m)) \supset m = n'$	(4.11)
Γ_{CR}	$\mathsf{Contains}(\{\hat{B}, \hat{X}, n', F(\hat{B}, \hat{X}, n', x)\}, F(\hat{B}, \hat{A}, n, m)) \supset$ $\hat{A} = \hat{B} \wedge n = n' \wedge x = m$	(4.12)
$(4), (9-12)$	$[\mathbf{Init_{CR}}]_A\ \mathsf{Honest}(\hat{A}) \wedge \mathsf{Honest}(\hat{B}) \supset \exists B.$ $(\diamondminus \mathsf{Send}(B, \{\hat{B}, \hat{X}, G(\hat{B}, \hat{X}, n', x)\}) \wedge \diamondminus \mathsf{Fresh}(B, n') \wedge n' = m) \vee$ $(\diamondminus \mathsf{Send}(B, \{\hat{B}, \hat{A}, n, F(\hat{B}, \hat{A}, n, m)\}))$	(4.13)
$\mathbf{AN3}, \mathbf{P1}$	$[\mathbf{Init_{CR}}]_A \diamondminus \mathsf{Fresh}(A, m)$	(4.14)
$(13-14)$	$[\mathbf{Init_{CR}}]_A\ \mathsf{Honest}(\hat{A}) \wedge \mathsf{Honest}(\hat{B}) \supset$ $\diamondminus \mathsf{Send}(B, \{\hat{B}, \hat{A}, n, F(\hat{B}, \hat{A}, n, m)\})$	(4.15)
$(15), \mathbf{HON}$	$[\mathbf{Init_{CR}}]_A\ \mathsf{Honest}(A) \wedge \mathsf{Honest}(\hat{B}) \supset \mathsf{ActionsInOrder}($ $\mathsf{Receive}(B, \{\hat{A}, \hat{B}, n\}), \mathsf{Send}(B, \{\hat{A}, \hat{B}, n, F(\hat{B}, \hat{A}, n, m)\}))$	(4.16)
$\mathbf{F}, \mathbf{AF3}$	$[\mathbf{Init_{CR}}]_A\ \mathsf{After}(\mathsf{Send}(A, \{\hat{A}, \hat{B}, n\}), \mathsf{Receive}(B, \{\hat{A}, \hat{B}, n\}))$	(4.17)
$\mathbf{F}, \mathbf{AF3}, \mathbf{HON}$	$[\mathbf{Init_{CR}}]_A\ \mathsf{Honest}(\hat{B}) \supset \mathsf{After}(\mathsf{Send}(B, \{\hat{A}, \hat{B}, n, F(\hat{B}, \hat{A}, n, m)\}),$ $\mathsf{Receive}(A, \{\hat{A}, \hat{B}, n, F(\hat{B}, \hat{A}, n, m)\}))$	(4.18)
$\mathbf{AF1}, \mathbf{AF2}$	$[\mathbf{Init_{CR}}]_A\ \mathsf{Honest}(\hat{A}) \wedge \mathsf{Honest}(\hat{B}) \supset \phi_{auth}$	(4.19)

Table 4.4: Deductions of $\hat{A}$ executing $\mathbf{Init_{CR}}$ role

$$
\begin{array}{rl}
\mathbf{CP1} & \mathsf{Computes}(X,t) \supset \mathsf{Has}(X,t) \\
\mathbf{CP2} & \mathsf{Has}(X,t) \supset (\mathsf{Computes}(X,t) \vee \\
 & \exists m.(\diamondminus\ \mathsf{Receive}(X,m) \wedge \mathsf{Contains}(m,t))) \\
\mathbf{CP3} & (\diamondminus\ \mathsf{Receive}(X,m) \wedge \mathsf{Contains}(m,t)) \supset \\
 & \exists Y.\exists m'.(\mathsf{Computes}(Y,t) \wedge \diamondminus\ \mathsf{Send}(Y,m') \wedge \mathsf{Contains}(m',t)) \\
\mathsf{Computes}(X,t) \equiv & (t = g^{ab} \wedge \mathsf{Computes}_{\mathsf{DH}}(X, g^a b)) \vee \\
 & (t = H(a) \wedge \mathsf{Computes}_{\mathsf{HASH}}(X, H(a))) \vee \\
 & (t = E_a(b) \wedge \mathsf{Computes}_{\mathsf{ENC}}(X, E_a(b))) \\
\mathsf{Computes}_{\mathsf{DH}}(X, g^{ab}) \equiv & ((\mathsf{Has}(X,a) \wedge \mathsf{Has}(X,g^b)) \vee (\mathsf{Has}(X,b) \wedge \mathsf{Has}(X,g^a))) \\
\mathsf{Computes}_{\mathsf{ENC}}(X, E_a(b)) \equiv & \mathsf{Has}(X,a) \wedge \mathsf{Has}(X,b) \\
\mathsf{Computes}_{\mathsf{HASH}}(X, H(a)) \equiv & \mathsf{Has}(X,a)
\end{array}
$$

Table 4.5: Computes Axioms

Chapter 5

Complexity-Theoretic Foundations for PCL

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. The purpose of this chapter is to suggest that formal reasoning, based on an abstract treatment of cryptographic primitives, can be used to reason about probabilistic polynomial-time protocols in the face of probabilistic polynomial-time attacks. We do this by proposing a new semantics for a variant of PCL. The new semantics brings forward some interesting distinctions that were not available in the coarser symbolic model, and also raises some apparently fundamental issues about the inherent logic of asymptotic probabilistic properties.

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. In the present paper, we interpret $\varphi \implies \psi$ as the union of $\neg\varphi$ and the composition of ψ with φ; the latter is also 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 [107]. Since the logic does not mention probability explicitly, we consider a formula "true" if it holds with asymptotically overwhelming probability.

In PCL, 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 what is called the Dolev-Yao model, after [48] and much subsequent work by others. 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 5.6 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.

5.1 Protocol Syntax

We use a simple "protocol programming language" based on [50, 35, 38] to represent a protocol by a set of roles, such as "Initiator", "Responder" or "Server", each specifying a sequence of actions to be executed by a honest participant. Since there are some differences from the syntax presented in Chapter 3, the syntax of terms and actions is given in Table 5.1.

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

Terms:				**Actions:**
N	$::=$	$\hat{X}$	$(name)$	$\texttt{a} \; ::=$
K	$::=$	X	(key)	$\mid \; \texttt{new}\, T, n$
S	$::=$	s	$(session)$	$\mid \; V := \texttt{enc}\, T, t, K$
n	$::=$	r	$(nonce)$	$\mid \; V := \texttt{dec}\, T, t, K$
T	$::=$	(N, S)	$(thread)$	$\mid \; [T =,]t/t$
V	$::=$	x	$(term\ variable)$	$\mid \; \texttt{send}\, T, t$
t_B	$::=$	$V \mid K \mid T \mid N \mid n \mid \langle t_B, t_B \rangle$	$(basic\ term)$	$\mid \; \texttt{receive}\, T, V$
t	$::=$	$t_B \mid \{t\}^n_K \mid \langle t, t \rangle$	$(term)$	

Table 5.1: Syntax of protocol terms and actions

notational convenience, we will sometimes write $\tilde{X}$ 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* 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 the notation $[ActionList]_{\tilde{X}}$ to denote a strand executed by thread $\tilde{X}$ and drop the thread identifier from the actions themselves. A *role* is a strand together with a basic term representing the initial knowledge of the thread. A *protocol* is a finite set of $Roles$, together with a basic term representing the initial intruder knowledge.

Action Predicates:
$\mathsf{a} \quad ::= \quad \mathsf{Send}(T, t) \mid \mathsf{Receive}(T, t) \mid \mathsf{New}(T, n)$

Formulas:
$\varphi \quad ::= \quad \mathsf{a} \mid t = t \mid \mathsf{Start}(T) \mid \mathsf{Possess}(T,\ t) \mid \mathsf{Indist}(T,\ t) \mid \mathsf{Fresh}(T,\ t) \mid \mathsf{Honest}(N) \mid$
$\mathsf{Start}(T) \mid \mathsf{Contains}(t,\ t) \mid \mathsf{ContainsOut}(t,\ t,\ t) \mid \mathsf{DecryptsHonest}(T,\ t) \mid$
$\mathsf{Source}(T,\ t,\ t) \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \exists Var.\, \varphi \mid \forall Var.\, \varphi \mid \neg\varphi \mid \varphi \supset \varphi \mid \varphi \Rightarrow \varphi$

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

Table 5.2: Syntax of the logic

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 5.5. 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.

5.2 Logic Syntax

The syntax of formulas is given in Table 5.2. Most formulas have the same intuitive meaning in the computational semantics as in the symbolic model [35, 38], except for predicates Possess and Indist. We summarize the meaning of formulas informally below, with precise semantics in the next section.

For every protocol action, there is a corresponding action predicate which asserts that the action has occurred in the run. For example, $\mathsf{Send}(\tilde{X}, t)$ holds in a run where the thread $\tilde{X}$ has sent the term t. $\mathsf{Fresh}(\tilde{X}, t)$ means that the value of t generated by $\tilde{X}$ 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 $\mathsf{Source}(\tilde{Y}, u, \{m\}^r_X)$ means that the only way for a thread $\tilde{X}$ different from $\tilde{Y}$ to know u is to learn u from the term $\{m\}^r_X$, possibly by some indirect path.

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

In the symbolic model [35, 38],the predicate Has states that a principal can "derive" a message or its contents from the information gathered during protocol execution. We use $\mathsf{Possess}(\tilde{X}, t)$ to state that it is possible to derive t by Dolev-Yao rules from $\tilde{X}$'s view of the run and $\mathsf{Indist}(\tilde{X}, t)$ to state that no probabilistic polynomial-time algorithm, given $\tilde{X}$'s view of the run, can distinguish t from a random value from the same distribution. Typically, we use 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.

5.3 Proof System

The proof system used in this chapter is based on the proof system developed in [35, 38, 11]. Some example axioms and rules are given in Table 5.3. 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 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 $\tilde{X}$ 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

Axioms:

$$\begin{array}{rcl}
\mathbf{AN2} & : & \top[\,\mathtt{new}\ x]_{\tilde{X}} \tilde{Y} \neq \tilde{X} \Rightarrow \mathsf{Indist}(\tilde{Y}, x) \\
\mathbf{AN3} & : & \top[\,\mathtt{new}\ x]_{\tilde{X}} \mathsf{Fresh}(\tilde{X}, x) \\
\mathbf{S1} & : & \mathsf{Source}(\tilde{Y}, u, \{m\}^r_X) \wedge \neg\mathsf{DecryptsHonest}(\hat{X}, \{m\}^r_X) \wedge \hat{Z} \neq \hat{X} \wedge \hat{Z} \neq \hat{Y} \wedge \\
& & \mathsf{Honest}(\hat{X}) \wedge \mathsf{Honest}(\hat{Y}) \Rightarrow \mathsf{Indist}(\tilde{Z}, u)
\end{array}$$

Proof rules:

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

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

Table 5.3: Fragment of the proof system

any partial information about the plaintext. The soundness of this axiom is proved by a complexity-theoretic reduction. Specifically, we show that if an attacker can break this property, then there is another attacker that can break the underlying IND-CCA2 secure encryption scheme [18].

Inference rules: Inference rules include generic rules from modal logics (e.g. $\mathbf{G3}$), sequencing rule $\mathbf{SEQ}$ 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 [35, 38].

First-order axioms and rules: We use two implications: a conditional implication $\Rightarrow$, discussed and defined precisely in section 5.6, 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.

5.4 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 [50, 35, 38] for further explanation and examples. The two protocol roles are:

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

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 $\tilde{X}$ completes the protocol with $\tilde{Y}$, then x will be a shared secret between them, provided both agents are honest. Formally,

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

Since the meaning of $\mathsf{Indist}(\tilde{Z}, x)$ (formally defined in Section 5.6) is that $\tilde{Z}$ cannot distinguish the secret nonce x from a randomly chosen nonce, this formula expresses a standard form of secrecy used in the cryptographic literature.

The axiomatic proof uses $\mathbf{AN2}$, a variant of $\mathbf{S1}$, and modus ponens $\mathbf{MP}$. The proof idea is that at the point the initiator produces the nonce x, by $\mathbf{AN2}$, it is indistinguishable from random to everyone else other than $\tilde{X}$ and $\tilde{Y}$. It continues to remain indistinguishable since it appears on the network under encryption with a public key whose corresponding private key is not available to the attacker. This part of the reasoning is codified by an axiom that is similar to $\mathbf{S1}$ and relies on the fact that the encryption scheme used is IND-CCA2 secure. Modus ponens is used in the general first-order reasoning involved in the proof.

5.5 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 [21].

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 [21], 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 $\texttt{send}$ and $\texttt{receive}$ 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 5.5.1. *(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, $\lambda : Term(e) \rightarrow \{0, 1\}^{p(\eta)}$ 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 5.5.2. *(Participant's View) Given a protocol Q, an adversary A, a security parameter η, a participant $\tilde{X}$ and a trace $t = \langle e, \lambda, O, K, R, R_T, \sigma\rangle \in T_Q(A,\eta)$, $View_t(\tilde{X})$ represents $\tilde{X}'s$ view of the trace. It is defined precisely as follows:*

If $\hat{X}$ is honest, then $View_t(\tilde{X})$ is the initial knowledge of $\tilde{X}$, a representation of $e_{|\tilde{X}}$ and $\lambda(x)$ for any variable x in $e_{|\tilde{X}}$. If $\hat{X}$ is dishonest, then $View_t(\tilde{X})$ is the union of the knowledge of all dishonest participants $\tilde{X}'$ after the trace t (where $View_t(\tilde{X}')$ 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 $\mathsf{Indist}()$. 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 5.5.3. *(LR Oracle) The LR Oracle [18] 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 5.5.4. *(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 5.5.5. *(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 $\tilde{X}$, 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. $Post(T_P)$ 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 5.5.6. *(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 $\tilde{X}$, 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 $\sigma'(P) = \lambda(s_{2\,|\tilde{X}})$. 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 $Post(T_P) = \{t[s \leftarrow s_1; s_2, K \leftarrow K_{\leq p_2}, \sigma \leftarrow \sigma'] \mid t \in T_P\}$.*

5.6 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 $\mathsf{Possess}(\tilde{X}, u)$ 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 "source" axiom **S1**, 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 5.6.1 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.

- $\left[\!\left[\,\mathsf{Send}(\tilde{X}, u)\,\right]\!\right](T, D, \epsilon)$ 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 $\mathtt{send}\ \tilde{Y}, v$ with $\lambda(\tilde{Y}) = \sigma(\tilde{X})$ 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 $\mathsf{Send}(\tilde{X}, u)$.

- $\left[\!\left[\,\mathsf{Honest}(\hat{X})\,\right]\!\right](T, D, \epsilon)$ 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, the resulting set is either the whole set T or the empty set ϕ depending on whether a principal is honest or not.

- $\left[\!\left[\,\mathsf{Start}(\tilde{X})\,\right]\!\right](T, D, \epsilon)$ includes all traces $\langle e, \lambda, O, K, R, R_T, \sigma\rangle \in T$ where $e = InitialState(\mathcal{I}); s$ and $\lambda(s)_{|\sigma(\tilde{X})} = \epsilon$. Intuitively, this set contains traces in which $\tilde{X}$ 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.

- $\llbracket\,\mathsf{ContainsOut}(u,\, v,\, t)\,\rrbracket\,(T, D, \epsilon)$ 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.

- $\llbracket\,\theta \wedge \varphi\,\rrbracket\,(T, D, \epsilon) \;=\; \llbracket\,\theta\,\rrbracket\,(T, D, \epsilon) \,\cap\, \llbracket\,\varphi\,\rrbracket\,(T, D, \epsilon)$.

- $\llbracket\,\theta \vee \varphi\,\rrbracket\,(T, D, \epsilon) \;=\; \llbracket\,\theta\,\rrbracket\,(T, D, \epsilon) \,\cup\, \llbracket\,\varphi\,\rrbracket\,(T, D, \epsilon)$.

- $\llbracket\,\neg\varphi\,\rrbracket\,(T, D, \epsilon) \;=\; T \setminus \llbracket\,\varphi\,\rrbracket\,(T, D, \epsilon)$.

- $\llbracket\,\exists x.\, \varphi\,\rrbracket\,(T, D, \epsilon) \;=\; \bigcup_\beta(\llbracket\,\varphi\,\rrbracket\,(T[x \leftarrow \beta], D, \epsilon)[x \leftarrow \sigma(x)])$
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.

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

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

- $\left\llbracket\,\mathsf{DecryptsHonest}(\tilde{Y},\, \{u\}^r_X)\,\right\rrbracket\,(T, D, \epsilon) \;=\; \llbracket\,\varphi\,\rrbracket\,(T, D, \epsilon)$ with $\varphi = \mathsf{Honest}(\hat{X}) \,\wedge\, \exists v.\, v := \mathsf{dec}\ \tilde{Y},\, \{u\}^r_X$.

- $\left\llbracket\,\mathsf{Source}(\tilde{Y},\, u,\, \{m\}^r_X)\,\right\rrbracket\,(T, D, \epsilon) \;=\; \llbracket\,\exists v.\forall w.\, \varphi\,\rrbracket\,(T, D, \epsilon)$ with :

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

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

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

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

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

- $[\![\, \theta[P]_{\tilde{X}}\varphi \,]\!]\, (T, D, \epsilon) \;=\; T_{\neg P} \cup [\![\, \neg\theta \,]\!]\, (Pre(T_P), D, \epsilon) \cup [\![\, \varphi \,]\!]\, (Post(T_P), D, \epsilon)$ with $T_{\neg P}$, $Pre(T_P)$, and $Post(T_P)$ as given by Definition 5.5.6.

Definition 5.6.1. *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 5.5.1.

Theorem 5.6.2. *(Soundness)* $\quad \forall Q,\ \forall \varphi, \quad Q \vdash \varphi \Rightarrow Q \models \varphi$

The soundness proof for an axiom involves demonstrating that the subset obtained by applying the semantics of the axiom to the set of computational traces nearly covers the whole set (following Definition 5.6.1). The soundness of the axioms and rules are proved using various cryptographic proof techniques. For example, **AN2** is proved sound using an information-theoretic argument, while the soundness of **S1** is proved by reduction to the security of the underlying IND-CCA2 secure encryption scheme.

Chapter 6

Unifying Compositional Protocol Security Definitions

The results presented in this chapter fall within the twin unifying themes of this dissertation—compositional reasoning and complexity-theoretic foundations in the context of security analysis of network protocols. However, the security notions studied here and the framework used to investigate their relation to each other is different from the one presented so far.

One appealing and relatively natural way to specify security properties is through simulation or equivalence. Focusing on protocols and equivalence, we can say what protocol P should achieve by giving an ideal functionality Q and saying that P be equivalent to Q in the face of attack. For example, P may be a key exchange protocol that operates over a public network, and Q an idealized protocol that uses some assumed form of private channel to generate and distribute shared keys. If no adversary can make P behave differently from Q, then since Q is impervious to attack by construction, we are assured that P cannot be successfully attacked. While this intuitive approach may seem clear enough, more precise formulations involve a number of details. For example, we may want to use one form of "ideal key exchange" with few messages to study several competing protocols. This ideal key exchange protocol is distinguishable from key exchange protocols that use different numbers of messages, but we can construct a simulator that uses the ideal key exchange primitive to produce additional messages. Thus a natural variation is not to expect P to be equivalent to Q, but ask that P be equivalent to some extension of Q that simulates P and retains the functionality of Q. Another issue is that we want users of the protocol to have the same positive outcome under all use scenarios.

The main advantage of specification by simulation or equivalence is composability: if protocol P is indistinguishable from ideal behavior Q, and protocol R is similarly indistinguishable from S, then P composed with R is indistinguishable from Q composed with S. Since many forms of security do not compose, the importance of composability should not be underestimated. Another advantage is generality: simulation and equivalence are meaningful when the protocol and the adversary operate in probabilistic polynomial time, and meaningful with nondeterministic computation and idealized cryptography.

We examine three similar specification approaches, and compare the methods over any computational framework satisfying familiar properties of process calculus. In this setting, we prove a very general correspondence: universal composability, black-box simulatability, and process equivalence express the same properties of a protocol, assuming asynchronous communication. Since our proofs hold for any process calculus that satisfies certain equational principles, our results are robust and not dependent on specialized properties of any specific computational setting. However, our results do not immediately apply to Turing machine models [26, 27, 28, 29, 30] or IO Automata models [114, 12] unless the assumed structural properties can be established for these models. If synchronous communication is available, one part of the equivalence becomes weaker because synchronous communication allows processes to detect an intermediate process acting as a buffer.

Although our results may be most useful to researchers concerned with one of the three methods, some high-level points may be understood more broadly. First, rather than finding technical differences between competing approaches, we find that three approaches based on essentially similar intuition are in fact technically equivalent. Someone beginning to study this literature can therefore start with any of the approaches. Second, results proved about one form of specification may be transferred to other forms, simplifying the likely future development of this topic. Third, we believe that the equivalence of three different technical definitions, and the fact that this equivalence holds for a broad range of computational models, strongly suggests that there is a robust, fundamental notion underlying the three definitions.

Universal composability [26, 27, 28, 30, 29] involves a protocol to be evaluated, an ideal functionality, two adversaries, and an environment. The protocol has the ideal functionality if, for every attack on the protocol, there exists an attack on the ideal functionality, such that the observable behavior of the protocol under attack is the same as the observable behavior of the idealized functionality under attack. Each set of observations is performed by the same environment. Black-box simulatability [114, 34, 12] is a formally stronger notion in which the two attacks must be related in a uniform way. Black-box simulatability involves a protocol to be evaluated, an ideal functionality, a simulator, *one* adversary, and an environment. The protocol has the ideal functionality if there exists a simulator such that the protocol and simulation are indistinguishable by any user environment in the face of any network adversary. The difference between universal composability and black-box simulatability is that in the first case, for every attack on the protocol, there must be an attack on the ideal functionality. In the second case, the same is true, but the second attack must be the same as the first attack, interacting with the the ideal functionality through a fixed simulator. An essential difference between the adversary and the environment is that the adversary only has access to network communication, while the environment interacts with the system through input/output connections that are not accessible to the adversary.

While the first two methods were developed using sets of communicating Turing machines and probabilistic I/O automata, the third method was developed using process calculus. In the third method, associated with spi-calculus [5, 6], applied π-calculus [3], and a probabilistic polynomial-time process calculus [101, 76, 103, 115], a protocol P satisfies specification Q if P is observationally equivalent to Q. The specification Q may be the result of combining some ideal process and a simulator. Observational equivalence is a standard notion from the study of programming languages and concurrency theory [95]. Process P is observationally equivalent to Q, written $P \cong Q$ if, for every context $C[\ ]$ consisting of a process with a place to insert P or Q, the observable behavior of $C[P]$ is the same as $C[Q]$. The reason observational equivalence is relevant to security is that we can think of the context as an attack. Then $P \cong Q$ means that any attack on P must succeed equally well on Q, and conversely. In [101, 76, 103, 115], an asymptotic form of process equivalence is used, making observational equivalence the same as asymptotic indistinguishability under probabilistic polynomial-time attack.

Our main results are that with synchronous communication, process equivalence implies black box simulatability, and black box simulatability is equivalent to universal composability. With asynchronous communication, all three notions are equivalent. These results are demonstrated using formal proofs based on standard process calculus properties such as associativity of parallel composition, commutativity, renaming of private channels, scope extrusion, and congruence, together with a few facts about processes that buffer or forward messages from one channel to another. Since our proofs are based on relatively simple axioms, the proofs carry over to any process calculus that satisfies these reasonable and well-accepted equational principles. Although the likely equivalence between universal composability and black-box simulatability has been mentioned in other work [26], we believe this is the first general proof of a precise relationship; an independent proof of the equivalence of black-box simulatability and universal composability is presented for a specific model (I/O automata) in [14]. Previous work on universal composability and black-box simulatability is not situated in process calculus, making the kind of general result we present here, and comparison with process equivalence methods, difficult. In subsequent work [42], we extend our analysis to include communicating Turing machines (as in [26] and other work on universal composability) and I/O automata (as in [114, 12] and related work).

6.1 Process Calculus

Process calculus is a standard language for studying concurrency [95, 124] that has proved useful for reasoning about security protocols [6, 116]. Two main organizing ideas in process calculus are actions and channels. *Actions* occur on channels and are used to model communication flows. *Channels* provide an abstraction of the communication medium. In practice, channels might represent the communication network in a distributed system environment or the shared memory in a parallel processor. In this section, we describe a family of process calculi by giving a sample syntax and a set of equational principles. Two example calculi that satisfy our equational assumptions, spi-calculus [6] and the probabilistic polynomial-time process calculus of [116], are discussed in Section 6.5.

A process calculus provides a syntax and an associated semantics. For concreteness, we will use the syntax defined by the following grammar, although additions to the language or changes in syntactic presentation are not likely to affect our results.

$$
\begin{array}{rcll}
\mathcal{P} & ::= & \mathbf{0} & \text{(termination)} \\
& & \nu_c(\mathcal{P}) & \text{(private channel)} \\
& & \mathtt{in}\,[c,x]\,.(\mathcal{P}) & \text{(input)} \\
& & \mathtt{out}\,[c,T]\,.(\mathcal{P}) & \text{(output)} \\
& & [T_1 = T_2].(\mathcal{P}) & \text{(match)} \\
& & (\mathcal{P} \mid \mathcal{P}) & \text{(parallel composition)} \\
& & !_{f(x)}.\big(\mathcal{P}\big) & \text{(bounded replication)}
\end{array}
$$

Intuitively $\mathbf{0}$ is the *empty process* taking no action. An input operator $\mathtt{in}\,[c,x]\,.\mathcal{P}$ waits until it receives a value on the channel c and then substitutes that value for the free variable x in $\mathcal{P}$. Similarly, an output $\mathtt{out}\,[c,T]\,.\mathcal{P}$ evaluates the term T, transmits that value on the channel c, and then proceeds with $\mathcal{P}$. Channel names that appear in an input or an output operation can be either public or private, with a channel being private if it is bound by a ν-operator and public otherwise. For convenience, we always α-rename channel names so that they are all distinct. The match operator $[T_1 = T_2]$ executes the process following iff T_1 have the T_2 value. The bounded replication operator has bound determined by the function f affixed as a subscript. The expression $!_{f(x)}.\big(\mathcal{P}\big)$ is expanded to the $f(x)$-fold parallel composition $\mathcal{P} \mid \cdots \mid \mathcal{P}$ before evaluation.

Since an output process $\mathtt{out}\,[c,T]\,.(\mathcal{P})$ only proceeds when another process is ready to receive its input, this process calculus has synchronous communication. For maximal generality, we proceed using a synchronous calculus, constructing asynchronous channels when desired by inserting buffer processes. In an asynchronous setting, inserting an additional buffer on a channel would presumably have no effect, and our results would therefore remain valid.

6.1.1 Equational Principles

A process calculus syntax and semantics give rise to an equivalence relation $\cong$ called *observational equivalence*. Informally, two process calculus expressions are observationally equivalent if they produce the same observations, when executed in any context. Traditionally, observations are actions on public channels, with actions on a channel c bound by $\nu_c()$ private and unobservable.

We will assume the standard equational principles collected in Table 6.1. Rules TRN, SYM, and $CONG$ state that observational equivalence is a congruence. Rule $RENAME$ renames bound channels and $SCOPE$ allows us to "extrude" the scope of a private channel. Intuitively, with channels alpha-renamed apart, we can enlarge the scope of a channel binding without changing the observable behavior of the process. Rule $ZERO$ says that the zero process produces no observable activity. Rules COM and ASC reflect the associativity and commutativity of parallel composition.

$$\mathcal{P} \mid \mathcal{Q} \cong \mathcal{Q} \mid \mathcal{P} \tag{COM}$$

$$(\mathcal{P} \mid \mathcal{Q}) \mid \mathcal{R} \cong \mathcal{P} \mid (\mathcal{Q} \mid \mathcal{R}) \tag{ASC}$$

$$\mathbf{0} \mid \mathcal{P} \cong \mathcal{P} \tag{ZERO}$$

$$\frac{\sigma(c) = \sigma(d)}{\nu_c(\mathcal{P}) \cong \nu_d(\mathcal{P}^{[d/c]})} \tag{RENAME}$$

$$\frac{c \notin Channels(\mathcal{C}[\mathbf{0}])}{\nu_c(\mathcal{C}[\mathcal{P}]) \cong \mathcal{C}[\nu_c(\mathcal{P})]} \tag{SCOPE}$$

$$\frac{\mathcal{P} \cong \mathcal{Q}, \mathcal{Q} \cong \mathcal{R}}{\mathcal{P} \cong \mathcal{R}} \tag{TRN}$$

$$\frac{\mathcal{P} \cong \mathcal{Q}}{\mathcal{Q} \cong \mathcal{P}} \tag{SYM}$$

$$\frac{\mathcal{P} \cong \mathcal{Q}}{\forall \mathcal{C}[\ ] \in Con \colon \mathcal{C}[\mathcal{P}] \cong \mathcal{C}[\mathcal{Q}]} \tag{CONG}$$

Table 6.1: Equivalence Principles

For reasons that will become clear in later sections of the chapter, we partition the set of public channel names into two infinite sets: the network channels and the input-output channels. We use the abbreviation net to refer to network channels and io for input-output channels. The difference between these two sets is that network channels will carry communication accessible to the adversary, while io channels allow users (the environment) to provide inputs to and observe the outputs produced by the protocol. We use ν_{net} to indicate binding $\nu_{n1}, \ldots, \nu_{nk}$ of all network channels in a process, and similarly ν_{io} for binding all io channels.

Throughout the chapter, we use $\mathcal{P}$, $\mathcal{F}$, $\mathcal{A}$, and $\mathcal{S}$ (with superscripts if necessary) for processes that represent a real protocol, an ideal functionality, an adversary and a simulator. These may be arbitrary processes, except that we impose restrictions on the names of public channels that each may contain. Specifically, all public channel names in a protocol $\mathcal{P}$, an ideal functionality $\mathcal{F}$, and an adversary $\mathcal{A}$ must be network or input-output channels, while all public channel names in a simulator $\mathcal{S}$ must be network channels. For any given protocol $\mathcal{P}$, the io channels of an adversary $\mathcal{A}$ attacking $\mathcal{P}$ must be disjoint from the io channels of $\mathcal{P}$. The purpose of these restrictions is to allow the adversary, for example, to connect to the network channels of a protocol or ideal functionality, but not to its input-output channels. Also, by making all network channels private when a protocol $\mathcal{P}$ is combined with an adversary $\mathcal{A}$, we ensure that only the io channels are accessible to the environment.

6.1.2 Buffers, dummy adversaries, and asynchronous communication

One of the main differences between process equivalence and the two other relations is that process equivalence only involves one form of context (surrounding processes interacting with the protocol), as opposed to separate adversary and environment contexts in the other two relations. Therefore, while investigating the connection between process equivalence and the other relations, we will replace the adversary in the other definitions by a "dummy adversary" that does nothing but pass messages to the surrounding context. Also, since the underlying process calculus is assumed to be synchronous, we interpose "buffers" between processes to enforce asynchronous communication when desired. Consequently, our proofs require certain equational properties of buffers and simple processes that forward messages from one channel to another.

For any pair a and b of disjoint lists of channel names, both of the same length, we assume two processes $\mathcal{D}_a^b$ and $\mathcal{B}_a^b$, which we will call a dummy adversary and a buffer process, respectively. Intuitively, the axioms about $\mathcal{D}_a^b$ and $\mathcal{B}_a^b$ below state that these processes forward data between channels $a_1, \ldots, a_k$ and $b_1, \ldots, b_k$, respectively. A dummy adversary may need to preserve message order to satisfy Axiom 6.1.1, but a buffer need not preserve message order. We assume that $\mathcal{D}_a^b$ and $\mathcal{B}_a^b$ have the channel names $a_1, \ldots, a_k$ and $b_1, \ldots, b_k$ free, and no other free channel names.

Axiom 6.1.1 (Dummy Adversary (DUMMY)). *Let $\mathcal{P}$ be a protocol and $\mathcal{A}$ be an adversary. Then $\nu_{net}(\mathcal{P} \mid \mathcal{A}) \cong \nu_{net,dummy}(\mathcal{P} \mid \mathcal{D}_{net}^{dummy} \mid \mathcal{A}^{[dummy/net]})$ where $dummy$ is a set of fresh channels of cardinality $|net|$ used to communicate between the dummy adversary and the modified adversary.*

Axiom 6.1.2 (Double Buffering (DBLBUF)). *Let $\mathcal{B}_a^b$, $\mathcal{B}_b^c$ and $\mathcal{B}_a^c$ be three buffers, for disjoint lists of channel names a, b, c of the same length. Then, $\nu_b(\mathcal{B}_a^b \mid \mathcal{B}_b^c) \cong \mathcal{B}_a^c$.*

Axiom 6.1.3 (Dummy and Buffer (DUMBUF)). *Let $\mathcal{B}_a^b$, $\mathcal{B}_b^c$ and $\mathcal{B}_a^c$ be three buffers and let $\mathcal{D}_b^c$ and $\mathcal{D}_a^b$ be dummy adversaries, for disjoint lists of channel names a, b, c of the same length. Then, $\nu_b(\mathcal{B}_a^b \mid \mathcal{D}_b^c) \cong \mathcal{B}_a^c$ and $\nu_b(\mathcal{D}_a^b \mid \mathcal{B}_b^c) \cong \mathcal{B}_a^c$*

Intuitively, Axiom 6.1.1 states that the interaction between a protocol and adversary through the network is indistinguishable from a situation when the communication between the protocol and the adversary is routed through the dummy adversary. Axiom 6.1.2 states that two buffers placed on a channel are indistinguishable from one buffer on that channel and Axiom 6.1.3 states that placing a dummy adversary and a buffer in sequence on a channel is equivalent to just having a buffer on that channel. Specific buffer and dummy adversary processes are presented in Section 6.5.

6.2 Security Definitions

In this section, we define three relations on processes, universal composability, black-box simulatability and process equivalence. These definitions are first presented in the synchronous form, then modified at the end of the section to assume asynchronous communication by placing buffers between process, adversary, and environment.

Definition 6.2.1. *Universal Composability: A protocol $\mathcal{P}$ is said to securely realize an ideal functionality $\mathcal{F}$ if for any adversary $\mathcal{A}$ attacking the protocol, there exists an adversary $\mathcal{A}^*$ attacking the ideal functionality, such that no context can distinguish whether it is interacting with $\mathcal{P}$ and $\mathcal{A}$ or with $\mathcal{F}$ and $\mathcal{A}^*$. Formally,*

$$\forall \mathcal{A}.\ \exists \mathcal{A}^*.\quad \nu_{net}(\mathcal{P} \mid \mathcal{A}) \cong \nu_{net}(\mathcal{F} \mid \mathcal{A}^*)$$

Figure 6.1 provides further intuition. The protocol as well as the ideal functionality communicate with the respective adversary processes over the network channels (denoted net in the figure). These channels are not visible to the context (or "environment" to use the terminology of [26, 114]). However, the context gets to communicate with these processes over the input-output channels (denoted io in the figure). All other channels of $\mathcal{P}$, $\mathcal{A}$, $\mathcal{F}$, and $\mathcal{A}^*$ are private. The intuition behind the distinction between channels is that if you are a user of SSL (Secure Sockets Layer), for example, your browser communicates with the implementation of SSL through io channels, while an attacker on the network has control of traffic on net channels.

Since the two process expressions in the definition of Universal Composability are observationally equivalent, this implies that if there is an attack on the real protocol, then there exists an equivalent attack on the ideal functionality. Hence, if the ideal functionality is impervious to attack by construction, then a real protocol that satisfies the above definition wrt the ideal functionality also cannot be attacked. While [26, 27, 28, 30, 29] discuss an adversary and environment, the environment here is provided by the context used in the definition of $\cong$.

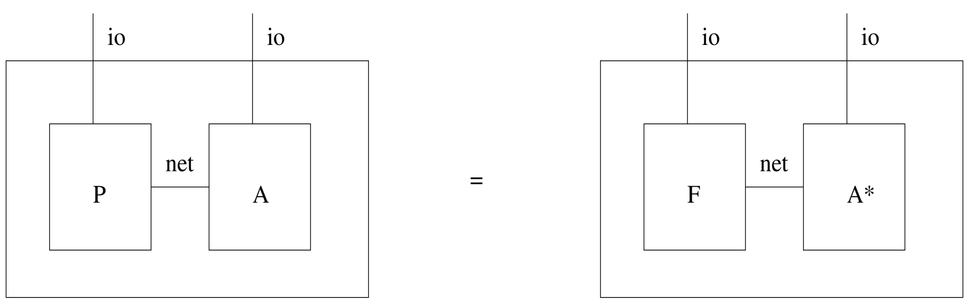

Figure 6.1: Universal Composability

In the definition of black-box simulatability and process equivalence, we use a simulator process whose public channels correspond to the union of the network channels of the ideal functionality (denoted sim below) and the network channels of the adversary (denoted net below).

Definition 6.2.2. *Black-box Simulatability: A protocol $\mathcal{P}$ is said to securely realize an ideal functionality $\mathcal{F}$ if there exists a simulator $\mathcal{S}$ such that for any adversary $\mathcal{A}$, no context can distinguish whether it is interacting with $\mathcal{P}$ and $\mathcal{A}$ or with $\mathcal{F}$, $\mathcal{S}$ and $\mathcal{A}$. Formally,*

$$\exists \mathcal{S}.\ \forall \mathcal{A}. \quad \nu_{net}(\mathcal{P} \mid \mathcal{A}) \cong \nu_{net}(\nu_{sim}(\mathcal{F} \mid \mathcal{S}) \mid \mathcal{A})$$

Figure 6.2 depicts this scenario. In effect, the simulator $\mathcal{S}$ uses the ideal functionality $\mathcal{F}$ to simulate the real protocol $\mathcal{P}$. The difference between universal composability and black-box simulatability is that in the first case, for every attack on the protocol, there must be an attack on the ideal functionality. In the second case, the same is true, but the second attack must be the same as the first attack, carried out on a simulation of the protocol that may rely on the ideal functionality.

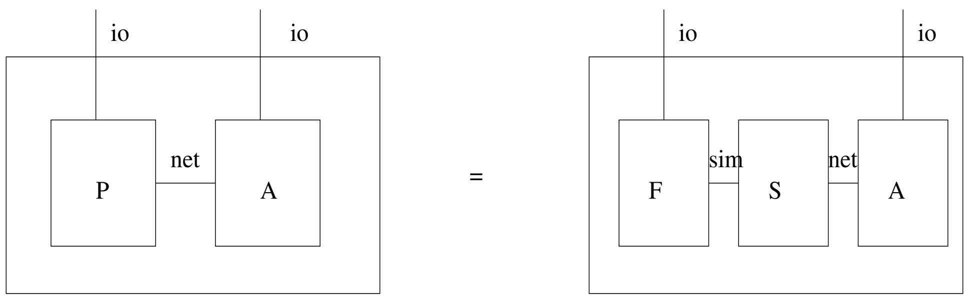

Figure 6.2: Black Box Simulatability

Definition 6.2.3. *Process Equivalence: A protocol $\mathcal{P}$ is said to securely realize an ideal functionality $\mathcal{F}$ if there exists a simulator $\mathcal{S}$ such that no context can distinguish whether it is interacting with $\mathcal{P}$ or with $\mathcal{F}$ and $\mathcal{S}$. Formally,*

$$\exists \mathcal{S}. \quad \mathcal{P} \cong \nu_{sim}(\mathcal{F} \mid \mathcal{S})$$

Figure 6.3 depicts this situation. Note that, unlike the first two definitions, the context has access to both the network and the input-output channels. Intuitively, the context used in the definition of observational equivalence serves the roles of both the adversary and the environment.

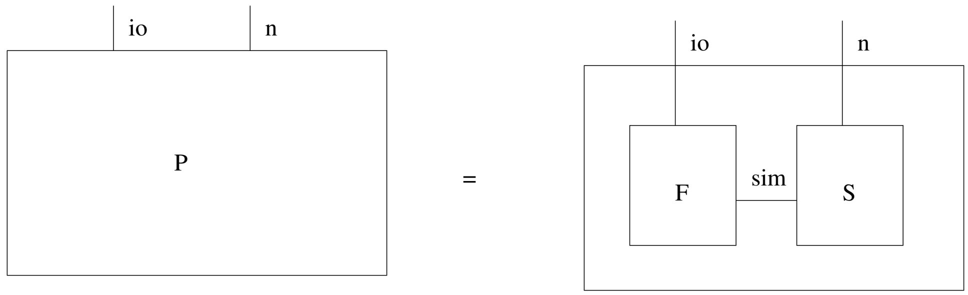

Figure 6.3: Process Equivalence

For each of these three relations, we formulate below corresponding asynchronous conditions by interposing message buffers or "bags" [95] on the network, input-output, and simulation channels. A buffer is any process satisfying the syntactic restrictions and axioms described in Section 6.1.2.

$$\begin{aligned}
UC: \quad & \forall \mathcal{A}.\ \exists \mathcal{A}^*.\ \ \nu_{ph,pn,an,ah}(\mathcal{B}^h_{ph} \mid \mathcal{P} \mid \mathcal{B}^{an}_{pn} \mid \mathcal{A} \mid \mathcal{B}^{h'}_{ah}) \cong \\
& \qquad\qquad\qquad \nu_{fh,fn,sn,sh}(\mathcal{B}^h_{fh} \mid \mathcal{F} \mid \mathcal{B}^{sn}_{fn} \mid \mathcal{A}^* \mid \mathcal{B}^{h'}_{sh}) \\
BB: \quad & \exists \mathcal{S}.\ \forall \mathcal{A}\ \nu_{ph,pn,h',ah'}(\mathcal{B}^h_{ph} \mid \mathcal{P} \mid \mathcal{B}^{h'}_{pn} \mid \mathcal{A} \mid \mathcal{B}^{h''}_{ah'}) \cong \\
& \qquad\qquad\qquad \nu_{fh,fn,sn,sh,h',ah'}(\mathcal{B}^h_{fh} \mid \mathcal{F} \mid \mathcal{B}^{sn}_{fn} \mid \mathcal{S} \mid \mathcal{B}^{h'}_{sh} \mid \mathcal{A} \mid \mathcal{B}^{h''}_{ah'}) \\
PE: \quad & \exists \mathcal{S}.\ \ \nu_{ph,pn}(\mathcal{B}^h_{ph} \mid \mathcal{P} \mid \mathcal{B}^n_{pn}) \cong \nu_{fh,fn,sh,sn}(\mathcal{B}^h_{fh} \mid \mathcal{F} \mid \mathcal{B}^{sh}_{fn} \mid \mathcal{S} \mid \mathcal{B}^n_{sn})
\end{aligned}$$

The binding of channels used in these definitions should be intuitively clear. In the UC condition, for example, $\mathcal{B}^h_{ph}$ buffers messages on $\mathcal{P}'s$ input-output channels; it forwards messages on the channels labelled h to $\mathcal{P}'s$ input-output channels (denoted ph). By binding the channels ph, we ensure that they are not observable by the environmental context. Similarly, $\mathcal{B}^{an}_{pn}$ and $\mathcal{B}^{h'}_{ah}$ buffer messages on the network channels between $\mathcal{P}$ and $\mathcal{A}$ and the io channels of $\mathcal{A}$.

$$\begin{array}{rll}
\mathbf{BB} & \exists \mathcal{S}.\ \forall \mathcal{A}.\ \ \nu_{net}(\mathcal{P} \mid \mathcal{A}) \cong \nu_{net}(\nu_{sim}(\mathcal{F} \mid \mathcal{S}) \mid \mathcal{A}) & (6.1) \\
(1), \mathbf{SCOPE}, \mathbf{ASC} & \exists \mathcal{S}.\ \forall \mathcal{A}.\ \ \nu_{net}(\mathcal{P} \mid \mathcal{A}) \cong \nu_{sim}(\mathcal{F} \mid \nu_{net}(\mathcal{S} \mid \mathcal{A})) & (6.2) \\
(2), \mathbf{RENAME} & \exists \mathcal{S}.\ \forall \mathcal{A}.\ \ \nu_{net}(\mathcal{P} \mid \mathcal{A}) \cong \nu_{net}(\mathcal{F}^R \mid \nu_{sim}(\mathcal{S}^R \mid \mathcal{A}^R)) & (6.3) \\
(3) & \forall \mathcal{A}.\ \exists \mathcal{A}^*.\ \ \nu_{net}(\mathcal{P} \mid \mathcal{A}) \cong \nu_{net}(\mathcal{F}^R \mid \mathcal{A}^*) & (6.4)
\end{array}$$

Table 6.2: Black-Box Simulatability implies Universal Composability (Synchronous Communication)

6.3 Black-box Simulatability and Universal Composability

In this section, we prove that universal composability and black-box simulatability are equivalent for both synchronous and asynchronous communication.

Theorem 6.3.1. *Universal composability is equivalent to black-box simulatability with synchronous communication.*

Proof. $\Leftarrow$: Follows immediately by scope extrusion ($\mathbf{SCOPE}$), associativity of parallel-or ($\mathbf{ASC}$) and renaming of private channels ($\mathbf{RENAME}$). The formal proof is given in Table 6.2. $\mathcal{A}^*$ is simply $\nu_{sim}(\mathcal{S}^R \mid \mathcal{A}^R)$. Thus, the combination of the simulator and the real adversary gives us the ideal process adversary demanded by the universal composability definition.

$\Rightarrow$: The formal proof is in Table 6.3. In Figure 6.4, the same proof is sketched out using intuitive diagrams of the form introduced in Section 6.2. We use standard process calculus proof rules: congruence ($\mathbf{CONG}$), associativity of parallel-or ($\mathbf{ASC}$), renaming of private channels ($\mathbf{RENAME}$), and scope extrusion ($\mathbf{SCOPE}$). The only step in the proof that does not immediately follow from our general equational principles rules is (4). We use the network-specific equivalence rule $\mathbf{DUMMY}$ (see Axiom 6.1.1) here. This rule captures the intuition that the environment cannot distinguish whether it is interacting with a process $\mathcal{P}$ and adversary $\mathcal{A}$ or it is interacting with $\mathcal{P}$ and $\mathcal{A}$ where the communication between them is forwarded through a "dummy adversary", $\mathcal{D}$, which just forwards messages in the order in which it receives them. Naturally, a dummy adversary process has to be defined

$$\begin{array}{rll}
\mathbf{UC} & \exists \mathcal{S}.\ \nu_{net}(\mathcal{P} \mid \mathcal{D}) \cong \nu_{net}(\mathcal{F} \mid \mathcal{S}) & (6.1) \\
(1), \mathbf{CONG} & \exists \mathcal{S}.\, \forall \mathcal{A}.\ \nu_{a'}(\nu_{net}(\mathcal{P} \mid \mathcal{D} \mid \mathcal{A}^R)) \cong \nu_{a'}(\nu_{net}(\mathcal{F} \mid \mathcal{S}) \mid \mathcal{A}^R) & (6.2) \\
(2), \mathbf{SCOPE}, \mathbf{ASC} & \exists \mathcal{S}.\, \forall \mathcal{A}.\ \nu_{net}(\mathcal{P} \mid \nu_{a'}(\mathcal{D} \mid \mathcal{A}^R)) \cong \nu_{a'}(\nu_{net}(\mathcal{F} \mid \mathcal{S}) \mid \mathcal{A}^R) & (6.3) \\
(3), \mathbf{DUMMY} & \exists \mathcal{S}.\, \forall \mathcal{A}.\ \nu_{net}(\mathcal{P} \mid \mathcal{A}) \cong \nu_{a'}(\nu_{net}(\mathcal{F} \mid \mathcal{S}) \mid \mathcal{A}^R) & (6.4) \\
(4), \mathbf{RENAME} & \exists \mathcal{S}.\, \forall \mathcal{A}.\ \nu_{net}(\mathcal{P} \mid \mathcal{A}) \cong \nu_{net}(\nu_{sim}(\mathcal{F}^R \mid \mathcal{S}^R) \mid \mathcal{A}) & (6.5)
\end{array}$$

Table 6.3: Universal Composability implies Black-Box Simulatability (Synchronous Communication)

and the assumed equivalence has to be proven in any concrete calculus in which we wish to apply our general results. □

Theorem 6.3.2. *Universal composability is equivalent to black-box simulatability with asynchronous communication.*

Proof. $\Leftarrow$: Follows immediately by scope extrusion ($\mathbf{SCOPE}$), associativity of parallel-or ($\mathbf{ASC}$) and renaming of private channels ($\mathbf{RENAME}$). The formal proof is exactly the same as the one for the synchronous model. $\mathcal{A}^*$ is simply $\nu_{sn,an}(\mathcal{S} \mid \mathcal{B}_{sn}^{an} \mid \mathcal{A})$.

$\Rightarrow$: The formal proof is given in Table 6.4. The standard process calculus rules used in the proof are congruence ($\mathbf{CONG}$) and scope extrusion ($\mathbf{SCOPE}$). The two non-standard rules used are ($\mathbf{DBLBUF}$) and ($\mathbf{DUMBUF}$). As for ($\mathbf{DUMMY}$) these rules need to be proven in any concrete calculus in which we wish to apply our general results. □

6.4 Process Equivalence and Black-box Simulatability

Process equivalence and black-box simulatability are equivalent with asynchronous communication. With synchronous communication, however, process equivalence implies black-box simulatability but not conversely.

Theorem 6.4.1. *Process equivalence implies black-box simulatability with synchronous communication.*

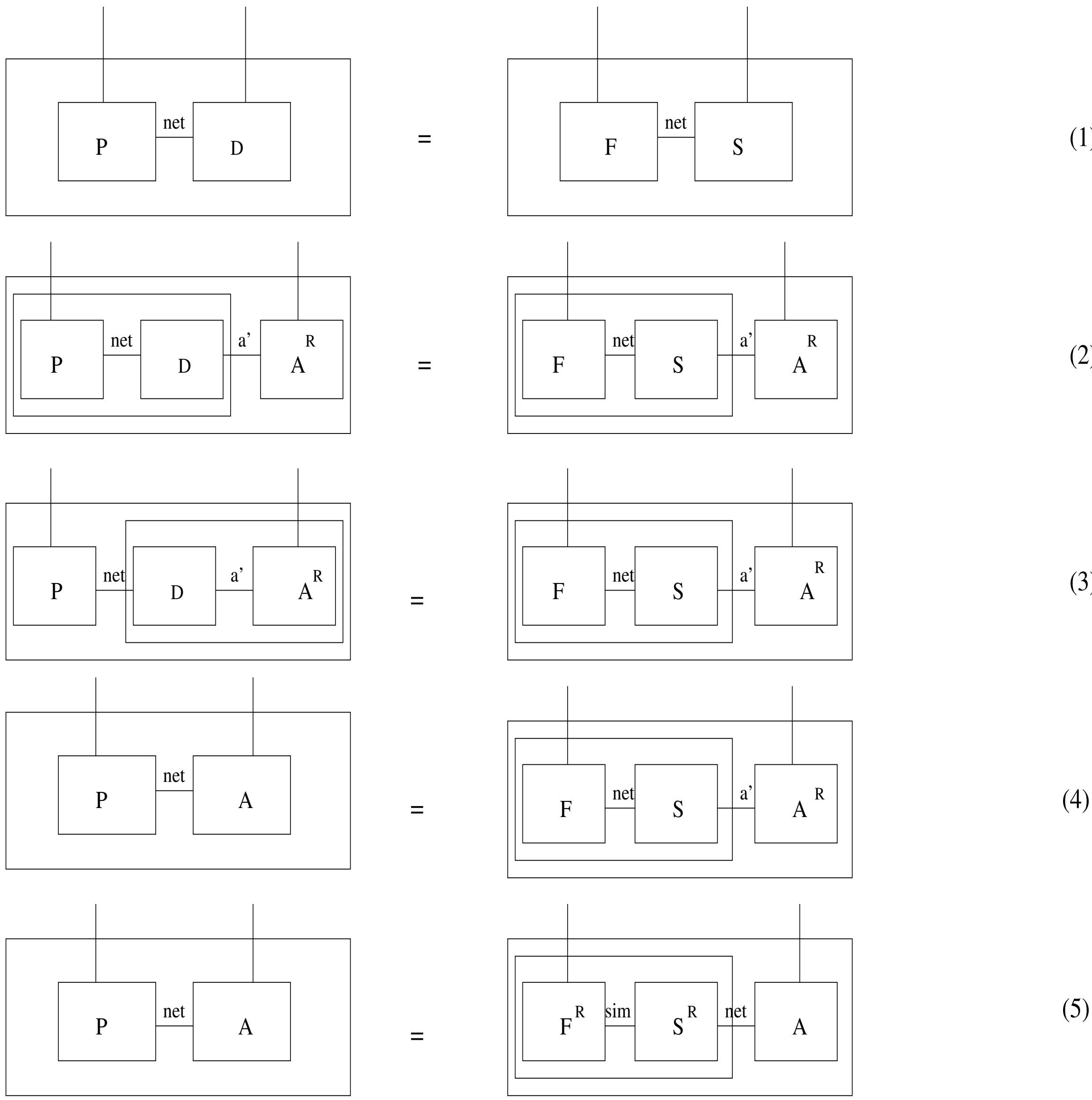

Figure 6.4: Universal Composability implies Black Box Simulatability: Proof Sketch

$$\begin{array}{rl}
\mathbf{UC} & \exists\mathcal{S}.\ \nu_{ph,pn,an,ah}(\mathcal{B}^{h}_{ph} \mid \mathcal{P} \mid \mathcal{B}^{an}_{pn} \mid \mathcal{D}^{ah}_{an} \mid \mathcal{B}^{h'}_{ah}) \cong \\
& \qquad \nu_{fh,fn,sn,sh}(\mathcal{B}^{h}_{fh} \mid \mathcal{F} \mid \mathcal{B}^{sn}_{fn} \mid \mathcal{S} \mid \mathcal{B}^{h'}_{sh}) \qquad (6.1) \\
(1), \mathbf{CONG} & \exists\mathcal{S}.\ \forall\mathcal{A}\ \nu_{ph,pn,an,ah,h',ah'}(\mathcal{B}^{h}_{ph} \mid \mathcal{P} \mid \mathcal{B}^{an}_{pn} \mid \mathcal{D}^{ah}_{an} \mid \mathcal{B}^{h'}_{ah} \mid \mathcal{A} \mid \mathcal{B}^{h''}_{ah'}) \\
& \cong \nu_{fh,fn,sn,sh,h',ah'}(\mathcal{B}^{h}_{fh} \mid \mathcal{F} \mid \mathcal{B}^{sn}_{fn} \mid \mathcal{S} \mid \mathcal{B}^{h'}_{sh} \mid \mathcal{A} \mid \mathcal{B}^{h''}_{ah'}) \qquad (6.2) \\
(2), \mathbf{SCOPE}, \mathbf{DUMBUF} & \exists\mathcal{S}.\ \forall\mathcal{A}\ \nu_{ph,pn,an,h',ah'}(\mathcal{B}^{h}_{ph} \mid \mathcal{P} \mid \mathcal{B}^{an}_{pn} \mid \mathcal{B}^{h'}_{an} \mid \mathcal{A} \mid \mathcal{B}^{h''}_{ah'}) \cong \\
& \nu_{fh,fn,sn,sh,h',ah'}(\mathcal{B}^{h}_{fh} \mid \mathcal{F} \mid \mathcal{B}^{sn}_{fn} \mid \mathcal{S} \mid \mathcal{B}^{h'}_{sh} \mid \mathcal{A} \mid \mathcal{B}^{h''}_{ah'}) \qquad (6.3) \\
(3), \mathbf{SCOPE}, \mathbf{DBLBUF} & \exists\mathcal{S}.\ \forall\mathcal{A}\ \nu_{ph,pn,h',ah'}(\mathcal{B}^{h}_{ph} \mid \mathcal{P} \mid \mathcal{B}^{h'}_{pn} \mid \mathcal{A} \mid \mathcal{B}^{h''}_{ah'}) \cong \\
& \nu_{fh,fn,sn,sh,h',ah'}(\mathcal{B}^{h}_{fh} \mid \mathcal{F} \mid \mathcal{B}^{sn}_{fn} \mid \mathcal{S} \mid \mathcal{B}^{h'}_{sh} \mid \mathcal{A} \mid \mathcal{B}^{h''}_{ah'}) \qquad (6.4)
\end{array}$$

Table 6.4: Universal Composability implies Black-Box Simulatability (Asynchronous Communication)

Proof. By definition, we have $\exists\mathcal{S}.\ \ \mathcal{P} \cong \nu_{sim}(\mathcal{F} \mid \mathcal{S})$. Hence, by the congruence rule, **CONG**, we have that $\exists\mathcal{S}.\ \forall\mathcal{A}.\ \ \nu_{net}(\mathcal{P} \mid \mathcal{A}) \cong \nu_{net}(\nu_{sim}(\mathcal{F} \mid \mathcal{S}) \mid \mathcal{A})$. This is precisely the definition of black-box simulatability in the synchronous communication. $\square$

The reason that process equivalence is strictly stronger than black-box simulatability in the synchronous case is that when the the adversary and environment are combined into one surrounding process context, this context may use the global ordering of events on the net and io channels to distinguish between real and ideal processes. This global ordering is not available when the adversary and the environment are separate processes as in the definition of black-box simulatability. Consider the two processes $\mathcal{P} ::= \texttt{out}\,[io,\alpha]\,.\texttt{out}\,[io,\gamma]\,.\texttt{out}\,[net,\beta]$ and $\mathcal{Q} ::= \texttt{out}\,[io,\alpha]\,.\texttt{out}\,[net,\beta]\,.\texttt{out}\,[net,\gamma]$. These two processes satisfy the definition of black-box simulatability in a non-deterministic process calculus like spi-calculus (using a simulator that just forwards messages to the adversary). However, they do not satisfy the definition of process equivalence since the global ordering of observables on the io and net channels is α, γ, β in one case and α, β, γ in the other.

Theorem 6.4.2. *Process equivalence is equivalent to black-box simulatability with asynchronous communication.*

$$\begin{array}{rl} \mathbf{BB} & \exists \mathcal{S}.\ \nu_{ph,pn,an,ah}(\mathcal{B}^{h}_{ph} \mid \mathcal{P} \mid \mathcal{B}^{an}_{pn} \mid \mathcal{D}^{ah}_{an} \mid \mathcal{B}^{h'}_{ah}) \cong \\ & \nu_{fh,fn,sh,sn,an,ah}(\mathcal{B}^{h}_{fh} \mid \mathcal{F} \mid \mathcal{B}^{sh}_{fn} \mid \mathcal{S} \mid \mathcal{B}^{an}_{sn} \mid \\ & \mathcal{D}^{ah}_{an} \mid \mathcal{B}^{h'}_{ah}) \qquad (6.1) \\ (1), \mathbf{SCOPE}, \mathbf{DUMBUF} & \exists \mathcal{S}.\ \nu_{ph,pn,an}(\mathcal{B}^{h}_{ph} \mid \mathcal{P} \mid \mathcal{B}^{an}_{pn} \mid \mathcal{B}^{h'}_{an}) \cong \\ & \nu_{fh,fn,sh,sn,an}(\mathcal{B}^{h}_{fh} \mid \mathcal{F} \mid \mathcal{B}^{sh}_{fn} \mid \mathcal{S} \mid \mathcal{B}^{an}_{sn} \mid \mathcal{B}^{h'}_{an}) \qquad (6.2) \\ (2), \mathbf{SCOPE}, \mathbf{DBLBUF} & \exists \mathcal{S}.\ \nu_{ph,pn,an}(\mathcal{B}^{h}_{ph} \mid \mathcal{P} \mid \mathcal{B}^{h'}_{pn}) \cong \\ & \nu_{fh,fn,sh,sn,an}(\mathcal{B}^{h}_{fh} \mid \mathcal{F} \mid \mathcal{B}^{sh}_{fn} \mid \mathcal{S} \mid \mathcal{B}^{h'}_{sn}) \qquad (6.3) \end{array}$$

Table 6.5: Black-Box Simulatability implies Process Equivalence (Asynchronous Communication)

Proof. $\Rightarrow$: By definition, $\exists \mathcal{S}.\ \nu_{ph,pn}(\mathcal{B}^{h}_{ph} \mid \mathcal{P} \mid \mathcal{B}^{n}_{pn}) \cong \nu_{fh,fn,sh,sn}(\mathcal{B}^{h}_{fh} \mid \mathcal{F} \mid \mathcal{B}^{sh}_{fn} \mid \mathcal{S} \mid \mathcal{B}^{n}_{sn})$. Hence, by the congruence rule, $\mathbf{CONG}$, we have $\exists \mathcal{S}.\ \forall \mathcal{A}.\ \nu_{ph,pn,an,ah}(\mathcal{B}^{h}_{ph} \mid \mathcal{P} \mid \mathcal{B}^{an}_{pn} \mid \mathcal{A} \mid \mathcal{B}^{h'}_{ah}) \cong \nu_{fh,fn,sh,sn,an,ah}(\mathcal{B}^{h}_{fh} \mid \mathcal{F} \mid \mathcal{B}^{sh}_{fn} \mid \mathcal{S} \mid \mathcal{B}^{an}_{sn} \mid \mathcal{A} \mid \mathcal{B}^{h'}_{ah})$. This is precisely the definition of black-box simulatability when communication is asynchronous. The proof follows the same line of reasoning as the one for synchronous communication.

$\Leftarrow$: The formal proof is in Table 6.5. Besides scope extrusion, it uses ($\mathbf{DBLBUF}$) and ($\mathbf{DUMBUF}$) to replace a dummy adversary and buffer process combination as well as two sequentially connected buffers by a single instance of a buffer process. □

6.5 Applications to specific process calculi

In this section, we demonstrate that several standard process calculi used for reasoning about security protocols (the probabilistic polynomial-time process calculus of [116], the spi-calculus [6], and the applied π-calculus [3]) satisfy the equational principles used in the axiomatic proofs in the previous sections. The proved relations between the various security definitions therefore hold in these calculi.

6.5.1 Probabilistic Poly-time Process Calculus

A probabilistic polynomial-time process calculus (PPC) for security protocols is developed in [101, 76, 103]; the best current presentations are [115, 116]. It consists of a set of *terms* that do not perform any communications, *expressions* that can communicate with other expressions, and, *channels* that are used for communication. Terms contain variables that receive values over channels. There is also a special variable $\mathbf{n}$ called the *security parameter*. Each expression defines a set of *processes*, one for each choice of value for the security parameter. Each channel name has a bandwidth polynomial in the security parameter associated with it by a function called σ. The bandwidth ensures that no message gets too large and, thus, ensures that the expression can be evaluated in time polynomial in the security parameter.

The class of terms used must satisfy the following two properties:

1. If θ is a term with k variables, then there exists a probabilistic Turing machine M_θ with k inputs and a polynomial $q_\theta(x_1, \ldots, x_k)$ such that:

   (a) The term θ, with $a_1, \ldots, a_k$ substituted for its k variables, reduces to a with probability p if and only if $M_\theta(a_1, \ldots, a_k)$ returns a with probability p; and,

   (b) For any choice of $a_1, \ldots, a_k$ we have that $M_\theta(a_1, \ldots, a_k)$ halts in time at most $q_\theta(|a_1|, \ldots, |a_k|)$.

2. For each probabilistic polynomial-time function $f\colon \mathbb{N}^m \to \mathbb{N}$, there exists a term θ such that M_θ computes f.

Essentially, the term language completely captures the class of probabilistic polynomial-time Turing machines. One example of such a set of terms is based on a term calculus called OSLR studied in [101] (based in turn on [17, 65]).

Although any probabilistic polynomial-time function can be computed by a term, communication requires additional syntactic forms. *Expressions* of PPC are given by the grammar in Section 6.1. The contexts, Con, of PPC are obtained from the grammar by adding a placeholder symbol for a "hole" to be filled in, as usual.

Operational Semantics

The evaluation of a variable-closed process proceeds in three steps: reduction, selection, and communication. In the *reduction step*, all terms and matches that are not in the scope of an input expression are evaluated. Since the expression is variable-closed and only inputs can bind variables, we know that every term outside the scope of an input has no free variables. This step simulates computation.

In the *selection step*, we use a probabilistic scheduler to select an action to perform. Actions include the silent action, τ; the input action $\mathtt{in}\langle c, a\rangle$ that reads the value a from the channel c into the variable x; the output action $\mathtt{out}\langle c, a\rangle$ that places the value a on the channel c; and the simultaneous action $\alpha \cdot \beta$ where one of α and β is an input action from the channel c of the value a and the other action is an output of the value a on the channel c obtained by using the action product $\cdot$ on α and β. We will say that two actions are of the same type if they are both inputs, outputs, or simultaneous actions with the same channel and value. The *scheduler* picks a particular type of simultaneous action from the set of available simultaneous action types according to the distribution defining the scheduler. However, silent actions must be performed if they are available since silent actions have higher priority. Then, one action of that type is picked uniformly at random from the set of available actions of that type. Further discussion may be found in [115].

In the *communication step*, we perform the indicated substitution taking care to truncate the value according to the bandwidth associated with the channel name. This is important for preserving the polynomial-time property of the process calculus.

We call this three-stage procedure an *evaluation step*; and evaluation proceeds in evaluation steps until the set of schedulable actions becomes empty. We refer the reader to [115] for more details.

Theorem 6.5.1. *Let P be a process. Then the evaluation of P can be performed in time polynomial in the security parameter.*

The proof proceeds by constructing a machine that evaluates P. The time-bound follows from the representation of terms and schedulers as probabilistic polynomial-time Turing machines.

A form of *weak probabilistic bisimulation* over asymptotically polynomial-time processes, or more simply *probabilistic bisimulation*, is developed in [115, 116] (see also [124]). Two processes P and Q are probabilistically bisimilar just when

1. If P can take an action α and with probability p become P', then Q must be able to take α to become processes $Q_1, \ldots, Q_k$ with total probability p; and,

2. If Q can take an action α and with probability p become Q', then P must be able to take α to become processes $P_1, \ldots, P_k$ with total probability p.

Using $\simeq$ to denote the bisimulation equivalence relation, [115, 116] show that $\simeq$ is a congruence.

Theorem 6.5.2. $\forall P, Q \in Proc. \forall C[\ ] \in Con\colon P \simeq Q \implies C[P] \simeq C[Q]$

Definition 6.5.3. *Let $\mathcal{P}$ and $\mathcal{Q}$ be two PPC expressions. Then $\mathcal{P} \cong \mathcal{Q}$ if, for sufficiently large n, $P^{\mathbf{n} \leftarrow n}$ is observationally indistinguishable from $Q^{\mathbf{n} \leftarrow n}$.*

A more precise definition can be found in [115, 116]. We also have the following theorem, proved in [115], which states that if two processes are probabilistically bisimilar, then they are observationally equivalent (in the sense of [115]). Hence, to prove observational equivalence, it is sufficient to demonstrate a probabilistic bisimulation.

Theorem 6.5.4. $\mathcal{P} \simeq \mathcal{Q} \implies \mathcal{P} \cong \mathcal{Q}$.

In [115], it is proved that all the equational principles of Table 6.1 hold in PPC. It remains to show that ($\mathbf{DUMMY}$), ($\mathbf{DBLBUF}$), and ($\mathbf{DUMBUF}$) hold in PPC. For simplicity we will construct uni-directional buffers, assuming that each public channel is *directional i.e.*, a channel name is used in a process only for inputs or only for outputs. We will say that a channel is an input channel (resp. output channel) just when it is to be used only for inputs (resp. outputs). Bi-directional buffers may be constructed by composing a pair of uni-directional channels.

Definition 6.5.5. *Let $A = \{a_1, \ldots, a_k\}$ and $B = \{b_1, \ldots, b_k\}$ be two equinumerous sets of channel names such that $a_i \in A$ is an input channel iff $b_i \in B$ is an output channel. We define $\mathcal{B}_{a_i}^{b_i}$ as*

$$!_{q(\cdot)}.\big(\mathtt{in}\,[a_i, y]\,.\mathtt{out}\,[b_i, y]\big)$$

in the case that a_i is an output channel, and

$$!_{q(\cdot)}.\big(\mathtt{in}\,[b_i, y]\,.\mathtt{out}\,[a_i, y]\big)$$

in the case that a_i is an input channel. Then we define the asynchronous buffer between A and B, *$\mathcal{B}_A^B$, as the expression $\mathcal{B}_{a_1}^{b_1} \mid \cdots \mid \mathcal{B}_{a_k}^{b_k}$.*

Essentially, an asynchronous buffer forwards messages between channels in A and channels in B without preserving any message-ordering since, for example, it is possible that an input on a_i is read, then a second input on a_i is read and forwarded onto b_i before the first input on a_i is forwarded onto b_i.

Definition 6.5.6. *Let $A = \{a_1, \ldots, a_k\}$ and $B = \{b_1, \ldots, b_k\}$ be two equinumerous sets of channel names such that $a_i \in A$ is an input channel iff $b_i \in B$ is an output channel. We define $\mathcal{D}_{a_i}^{b_i}$ as*

$$\mathtt{in}\,[a_i, y]\,.\mathtt{out}\,[b_i, y]\,.\mathtt{out}\,[syn_i, 1] \mid !_{q(\cdot)}.\big(\mathtt{in}\,[syn_i, x]\,.\mathtt{in}\,[a_i, y]\,.\mathtt{out}\,[b_i, y]\,.\mathtt{out}\,[syn_i, 1]\big)$$

in the case that a_i is an output channel, and

$$\mathtt{in}\,[b_i, y]\,.\mathtt{out}\,[a_i, y]\,.\mathtt{out}\,[syn_i, 1] \mid !_{q(\cdot)}.\big(\mathtt{in}\,[syn_i, x]\,.\mathtt{in}\,[b_i, y]\,.\mathtt{out}\,[a_i, y]\,.\mathtt{out}\,[syn_i, 1]\big)$$

in the case that a_i is an input channel. Then we define the dummy adversary between A and B, *$\mathcal{D}_A^B$, as the expression*

$$\nu_{syn}(\mathcal{D}_{a_1}^{b_i} \mid \cdots \mid \mathcal{D}_{a_k}^{b_k})$$

The expression $\mathcal{D}_A^B$ simply forwards communications between each channel $a_i \in A$ and $b_i \in B$. The channel syn_i is used to synchronize between the various inputs and outputs on the channel a_i in $\mathcal{D}_A^B$ to avoid situations where, for example, a value has been read on the channel b_i and, before it is forwarded, a new value is read on the channel b_i and then forwarded. Essentially, the use of syn_i allows us to preserve the ordering on communications on a_i by guaranteeing that if $\mathcal{D}_A^B$ receives the message o before o', it will transmit o before o'. Thus a dummy adversary is just a message-order-preserving buffer.

Theorem 6.5.7. *The equivalence principles* $(\mathbf{DUMMY})$*,* $(\mathbf{DBLBUF})$*, and* $(\mathbf{DUMBUF})$ *hold in PPC.*

We prove these equivalences by constructing a probabilistic bisimulation and then applying Theorem 6.5.4.

6.5.2 Spi-Calculus and Applied π-Calculus

Spi-calculus [6] and applied π-calculus [3] are two other process calculi that have been used to reason about security protocols. All the standard structural equivalence rules: associativity of parallel composition $(\mathbf{ASC})$, renaming of private channels $(\mathbf{RENAME})$, scope extrusion $(\mathbf{SCOPE})$, congruence $(\mathbf{CONG})$, which were collected in Table 6.1, hold in these calculi. The network-specific equivalences are also satisfied with appropriate definitions of dummy adversary and buffer processes. Hence the results proved in Section 6.3 and Section 6.4 also hold for these calculi. A representative proof for spi-calculus is given below.

Theorem 6.5.8. *Theorem 6.3.1 and Theorem 6.4.1 hold for spi-calculus.*

Proof. The standard equivalence rules used in proving the two theorems: associativity of parallel-or $(\mathbf{ASC})$, renaming of private channels $(\mathbf{RENAME})$, congruence $(\mathbf{CONG})$, and scope extrusion $(\mathbf{SCOPE})$ hold in spi-calculus. The only non-standard step corresponds to $\mathbf{DUMMY}$. The proof relies on the observation that the situation in which processes $\mathcal{P}$ and $\mathcal{A}$ communicate over a private channel is observationally equivalent to the one in which all such communication is routed through a dummy process that just forwards messages in both directions. For simplicity, we consider only the case when there are two channels c_0 and c_1 between P and A. Since channels are directional, without loss of generality, we assume that channel c_0 is from A to P (*i.e.*, only A outputs messages on c_0, and only P receives messages on c_0), and channel c_1 is from P to A. The proof extends directly to the multiple-channel case.

Rewriting the statement using spi-calculus formalism and letting A^d stand for $A[d/c]$, we wish to demonstrate that

$$(\nu c_0, c_1)(P \mid A) \quad \simeq \quad (\nu c_0, c_1, d_0, d_1)(P \mid ((\nu s_0, s_1)(D_0 \mid D_1) \mid A^d))$$

where

$$\begin{array}{rcl} D_0 & = & d_0(y).\overline{c_0}\langle y\rangle.\overline{s_0}\langle 1\rangle \mid !\, s_0(x).d_0(y).\overline{c_0}\langle y\rangle.\overline{s_0}\langle 1\rangle \\ D_1 & = & d_1(y).\overline{c_1}\langle y\rangle.\overline{s_1}\langle 1\rangle \mid !\, s_1(x).d_1(y).\overline{c_1}\langle y\rangle.\overline{s_1}\langle 1\rangle \end{array}$$

We outline the proof in the following direction:

$$(\nu c_0, c_1)(P \mid A) \sqsubseteq (\nu c_0, c_1, d_0, d_1)(P \mid ((\nu s_0, s_1)(D_0 \mid D_1) \mid A^d)).$$

The proof in the other direction is similar. For our purposes, it is sufficient to recall that, informally, P *passes a test* (R, β) if P produces an observable on a channel named β when run in parallel with R. By definition, $P_1 \sqsubseteq P_2$ if, for any test (R, β) passed by P_1, P_2 also passes the test.

Let (R, β) be some test passed by $(\nu c)(P \mid A)$. By Proposition 4 [6], this implies that there exist an agent A and a process Q such that $(\nu c_0, c_1)(P \mid A)|R \xrightarrow{\tau}^* Q$ and $Q \xrightarrow{\beta} A$. Since we assume that P and A communicate only via channels c_0 and c_1, every reaction of $P \mid A$ is a reaction of P, a reaction of A, or an interaction between P and A. In the latter case, because we assumed that channels are directional, $P = c_0(x).P', A = \overline{c_0}\langle m\rangle.A', P|A \xrightarrow{\tau} P'[m/x]|A'$, or $P = \overline{c_1}\langle m\rangle.P', A = c_1(x).A', P|A \xrightarrow{\tau} P'|A'[m/x]$. To prove the lemma by induction over all reactions of $(\nu c_0, c_1)(P|A)|R$, it is sufficient to demonstrate that, if $P = c_0(x).P', A = \overline{c_0}\langle m\rangle.A', c_0(x).P'|\overline{c_0}\langle m\rangle.A' \xrightarrow{\tau} P'[m/x]|A'$, then $c_0(x).P'|((\nu s_0, s_1)(D_0|D_1)|\overline{d_0}\langle m\rangle.A'^d) \xrightarrow{\tau} P'[m/x]|((\nu s_0, s_1)(D_0|D_1)|A'^d)$. The proof for the case $P = \overline{c_1}\langle m\rangle.P', A = c_1(x).A', \overline{c_1}\langle m\rangle.P'|c_1(x).A' \xrightarrow{\tau} P'|A'[m/x]$ is symmetric.

$$\begin{array}{l} c_0(x).P'|((\nu s_0, s_1)(D_0|D_1)|\overline{d_0}\langle m\rangle.A'^d) \;= \\ c_0(x).P'|((\nu s_0, s_1)((d_0(y).\overline{c_0}\langle y\rangle.\overline{s_0}\langle 1\rangle|!\, s_0(x).d_0(y).\overline{c_0}\langle y\rangle.\overline{s_0}\langle 1\rangle)|D_1)|\overline{d_0}\langle m\rangle.A'^d) \;\xrightarrow{\tau} \\ c_0(x).P'|((\nu s_0, s_1)((\overline{c_0}\langle m\rangle.\overline{s_0}\langle 1\rangle|!\, s_0(x).d_0(y).\overline{c_0}\langle y\rangle.\overline{s_0}\langle 1\rangle)|D_1)|A'^d) \;\xrightarrow{\tau} \\ P'[m/x]|((\nu s_0, s_1)((\overline{s_0}\langle 1\rangle|!\, s_0(x).d_0(y).\overline{c_0}\langle y\rangle.\overline{s_0}\langle 1\rangle)|D_1)|A'^d) \;\xrightarrow{\tau} \\ P'[m/x]|((\nu s_0, s_1)((d_0(y).\overline{c_0}\langle y\rangle.\overline{s_0}\langle 1\rangle|!\, s_0(x).d_0(y).\overline{c_0}\langle y\rangle.\overline{s_0}\langle 1\rangle)|D_1)|A'^d) \;= \\ P'[m/x]|((\nu s_0, s_1)(D_0|D_1)|A'^d) \end{array}$$

□

Chapter 7

Related Work

In this chapter, we compare our work with related research on similar topics. The main styles of relevant related work can be grouped into four categories. A comparison is presented between PCL and two previous approaches for proving security properties of network protocols—a specialized logic called BAN [24], and Paulson's Inductive Method [110]. We survey research results on systematic methods for protocol design that preceded our work on PDS. The composition paradigm of PCL is compared and contrasted with other approaches for reasoning compositionally in security, cryptography and distributed computing. Finally, we summarize other results on symbolic reasoning about the complexity-theoretic model of cryptographic protocols.

7.1 Proving security properties of network protocols

PCL shares several features with BAN [24], 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”. For example, in PCL the fact that key k is a ”shared secret” between agents X and Y is represented using the logical formula $\mathsf{Has}(X, k) \wedge \mathsf{Has}(Y, K) \wedge \forall Z.\ (\mathsf{Has}(Z, k) \supset (Z = X \vee Z = Y))$. 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 [110]. 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.

7.2 Systematic design of secure network protocols

There has been some work on systematizing the practice of constructing security protocols, starting from simple components and extending them by features and functions. In [23], Bird and co-authors describe the systematic design of a family of authentication protocols. A similar approach is taken by Diffie, Van Oorschot and Wiener in their presentation of the STS protocol in [46]. More recently, Bellare, Canetti and Krawczyk [19] have studied two interesting protocol transformations, which they call *authenticators*, which generically add authentication to a given protocol scheme. In [113], Perrig and Song present a method for automatic generation of protocols. Their approach involves searching the entire space of protocols for one that satisfies the security requirements and is minimal with respect to some metric (e.g., number of public key operations). A protocol is defined as a sequence of messages sent between two parties and the message space is specified by a grammar. Whether a particular protocol satisfies the security requirements is decided by running it through the automatic protocol analysis tool, Athena [119]. Independently, Clark and Jacob developed a similar approach for protocol synthesis [32]. They use genetic algorithms to search the space of protocols expressible in BAN logic [24]. Other works on using formal logic for protocol design include [9, 25].

7.3 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 [56] or something very similar to it [61]. 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 [90], 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 [123]. 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 [122]. A somewhat different approach is used by Lynch [80] to prove that the composition of a simple shared key communication protocol and the Diffie-Hellman key distribution protocol is secure. Her model uses I/O automata and the protocols are shown to compose if adversaries are only passive eavesdroppers.

In a recent paper [31], Canetti, Meadows and Syverson, revisit the protocol composition problem. They show how the interaction between a protocol and its environment can have a major effect on the security properties of the protocol. In particular, they demonstrate a number of attacks on published and widely used protocols that are not feasible against the protocol running in isolation but become feasible when they are run in parallel with certain other protocols. This study further reinforces the importance of methods for reasoning about the composability of protocols. We believe that the results presented in this dissertation represent significant progress in this direction. The methods presented in Section 4.1.1 provide a way to implicitly characterize, using invariants, a class of protocols with which a specific protocol can be safely composed. In particular, our formalism justifies some of the design principles discussed by the authors. One recommendation is that the environment should not use keys or other secrets in unaltered form. Specifically, the protocol under consideration should not encrypt messages with a key used to encrypt messages by any protocol in its environment. The reason this makes sense is that if two protocols use a particular form of encrypted message as a test to authenticate a peer, then the attacker might be able to make a principal running the first protocol accept a message which actually originated in a run of the second protocol. If this is indeed the case, then in our formalism, the invariant for the protocol under consideration would fail to hold in such an environment, and the composition proof would therefore not go through. However, this seems like an overly conservative design approach since not every two protocols which use the same encryption keys interfere with each other's security. The invariant-preservation method can help identify protocols which can run safely in parallel even if they share keys. We note that the above principle has been followed in the design of real-world protocols like IKE [59]. Also, Guttman and Fábrega have proved a theoretical result to the same effect in their strand space model [58]. Another rule of thumb (also recommended by Kelsey, Schneier and Wagner in [72]), is the use of unique protocol identifiers to prevent a message intended for use in one protocol to be mistaken for use in another protocol. This idea is also founded on similar intuition. To give an example, in our logic, an invariant in proving an authentication property could be: "if Bob generated a signature of a particular form, he sent it in response to a particular message of a protocol"; adding the unique protocol identifier inside the signature will ensure that this invariant is trivially satisfied for all other protocols, thereby allowing composability. However, many existing protocols do not follow this principle.

It is well known that many natural security properties (e.g., noninterference) are not preserved either under composition or under refinement. This has been extensively explored using trace-based modelling techniques [82, 84, 85, 86, 87], 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" [26]. 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 [100], 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.

7.4 Computationally sound symbolic protocol analysis

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 [7] 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 [67]. Some semantic connections between symbolic and computational models have been developed by a team at IBM Zurich, *e.g.,* [13], with other connections explored in a series of related papers by Micciancio, Warinschi, and collaborators [94, 125, 33]. Herzog [62, 63] shows that if a protocol attack exists in a Dolev-Yao model, there is an attack in a computational model. More recent related work also appears in [69, 33]. Currently, there is a lot of activity in this area and we expect a number of other publications to appear soon.

Chapter 8

Conclusions and Future Work

In this dissertation, we have presented several results in the area of security analysis of network protocols. Our main contribution is PCL—a logic for proving security properties of network protocols. Security proofs in PCL are relatively short and intuitive and scale to protocols of practical interest. Two central results for this logic are a composition theorem and a computational soundness theorem. The composition theorem allows proofs of complex protocols to be built up from proofs of their constituent sub-protocols. It is formulated and proved by adapting ideas from the assume-guarantee paradigm for reasoning about distributed systems. The computational soundness theorem guarantees that, for a class of security properties and protocols, axiomatic proofs in a fragment of PCL carry the same meaning as hand-proofs done by cryptographers. The soundness proof uses standard proof techniques from cryptography, in particular, complexity-theoretic reductions. In addition, we have developed an abstraction-refinement method for proving protocol properties in an abstract template form using a higher-order extension of PCL. PCL has been applied to the IEEE 802.11i protocol suite (which includes TLS as a component) [60] and to the IETF GDOI protocol for secure group communication [92]. The second case study identified a previously undiscovered flaw in the protocol. In ongoing work, PCL is being used to analyze IKEv2 [71], IEEE 802.16e [2], Kerberos [106], and Mobile IPv6 [70] protocols. A second major contribution is PDS—a framework for incremental protocol construction, starting from simple components and extending them by applying a sequence of protocol transformation operations. PDS seeks to provide a rigorous foundation for common protocol design practice. Finally, we demonstrate using process calculus techniques that several related compositonal security definitions—universal composability, blackbox simulatability, and process equivalence—can be unified under reasonable assumptions about the communication model.

Although these results represent significant advances in the state-of-the-art, it will take several people a number of years to fully accomplish the goals of this program. One current effort seeks to extend and further refine PCL [10]. Specific goals include extending the programming language to model a larger class of protocols, simplifying the syntax and proof system, and expanding the reasoning methods to cover a larger set of protocol properties. While current applications focus on authentication and certain specific forms of secrecy properties, in future work we hope to generalize the reasoning method for proving secrecy properties, and develop methods for modelling and reasoning about certain forms of liveness properties like abuse-freeness, fairness, and denial-of-service protection. Knowledge-based specifications [51] seem useful to capture certain properties like key confirmation. We also hope to investigate questions about the decidability of the full PCL as well as fragments which are expressive enough to be useful in practice. The eventual goal is to develop a practical tool for industrial use. One current tool effort encodes the syntax and proof system of a fragment of PCL into Isabelle, a generic theorem-prover.

A second direction is to extend PCL to reason about security in different threat models. Threat models can differ on several respects, in particular, the computational capabilities of protocol principals and adversaries, and the degree of control the adversary has over the network. While the core PCL allows the adversary complete control over the network, in certain applications like Mobile IPv6, it is more realistic to assume that the attacker only has access to certain parts of the network. The computational abilities of the adversary spans the spectrum from a fixed set of actions in the symbolic model, to any polynomial time computation in the complexity-theoretic model, and any computable function in the information-theoretic model. The eventual goal is to develop a unified theory for reasoning about security in a broad range of models. The work on Computational PCL presented in this dissertation is a first step in this direction. In subsequent work, we hope to extend the scope of this logic to allow its application to industrial protocols. Specific extensions include expanding the set of cryptographic primitives modelled to cover digital signatures, symmetric encryption, and message authentication codes, and to formalize properties of protocols for key exchange following standard definitions in the cryptography literature. Some progress has been made towards this goal [57, 41]. Other directions for further work include developing a version of the logic that supports reasoning about concrete security reductions and investigation of minimal requirements on cryptographic primitives to guarantee security properties of the protocols in which they are used. Besides serving as a useful tool for reasoning about cryptography, we envision that this work will also help us understand better certain fundamental questions about the logical nature of complexity-theoretic reductions. To give a concrete example, we draw the attention of the reader to the semantics of implication in CPCL. Implication uses conditional probabilty. We note that the reason for this is the fact that most security definitions in cryptography are stated in terms of conditional probability. Consequently, the propositional fragment is not classical. While there appear to be some connections with probabilistic logics [107], the precise nature of the logic bears further investigation.

While most of the results in this dissertation focus on analysis of secure network protocols, PDS represents a step towards a systematic theory of protocol design. The current work can be extended in several directions. One medium-term goal is to use PDS to syntactically derive a library of practical protocols, starting from basic components like challenge-response and Diffie-Hellman key exchange. Another goal is to develop proof methods in PCL for each derivation operation in PDS. Specifically, a formal theory of protocol transformations is yet to emerge. The acid test for PDS is a case study where an open standards protocol gets developed using some tool that implements this framework. One such tool effort is underway.

Finally, the results in this dissertation provide a good starting point for a deeper investigation of the composition problem in computer security and cryptography. The importance of compositional methods in the design and analysis of secure systems is now widely recognized (cf. [126]). However, there is no comprehensive foundational theory for secure composition of secure systems and software. We believe that the assume-guarantee paradigm developed for PCL might be applicable to these other kinds of security mechanisms. In the field of cryptography also, the composition problem has received significant attention. One current approach to this problem is the framework of *universal composability* [26, 114]. The universal composability condition provides strong composition guarantees: a primitive or protocol that satisfies this condition retains its security guarantees in any environment in which it is used. In contrast, the assume-guarantee paradigm of PCL only allows conditional composability: a protocol is secure only in an environment which satisfies a certain set of invariant assumptions associated with the protocol. The ability to reason about protocols under assumptions about the way they will be used offers greater flexibility and appears essential for developing modular proofs about certain classes of protocols. In addition, a number of impossibility results about the realizability of UC-secure primitives and protocols [26, 29, 40] indicates that the UC condition may be too stringent to apply to certain protocols of interest.

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] IEEE P802.16e/D10.0. IEEE Standard for local and metropolitan area networks. part 16: Air interface for fixed and mobile broadband wireless access systems. amendment for physical and medium access control layers for combined fixed and mobile operation in licensed bands., August 2005.

[3] 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.

[4] 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).

[5] M. Abadi and A. D. Gordon. A bisimulation method for cryptographic protocol. In *Proc. ESOP 98*, Lecture notes in Computer Science. Springer, 1998.

[6] M. Abadi and A. D. Gordon. A calculus for cryptographic protocols: the spi calculus. *Information and Computation*, 143:1–70, 1999. Expanded version available as SRC Research Report 149 (January 1998).

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

[8] W. Aiello, S.M. Bellovin, M. Blaze, R. Canetti, J. Ioannidis, A.D. Keromytis, and O. Reingold. Just fast keying (JFK), 2002. Internet draft.

[9] J. Alves-Foss and T. Soule. A weakest precondition calculus for analysis of cryptographic protocols. In *DIMACS Workshop on Design and Formal Verification of Crypto Protocols*, 1997.

[10] 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*, pages 94–110. IEEE, 2005.

[11] 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. to appear.

[12] M. Backes, B. Pfitzmann, and M. Waidner. Reactively secure signature schemes. In *Proceedings of 6th Information Security Conference*, volume 2851 of *Lecture Notes in Computer Science*, pages 84–95. Springer, 2003.

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

[14] M. Backes, B. Pfitzmann, and M. Waidner. A general composition theorem for secure reactive systems. In *Proceedings of 1st Theory of Cryptography Conference*, volume 2951 of *Lecture Notes in Computer Science*. Springer, 2004.

[15] M. Barr and C. Wells. *Category Theory for Computing Science*. Prentice Hall, New York, 1999. Third Edition.

[16] M. Baugher, B. Weis, T. Hardjono, and H. Harney. The Group Domain of Interpretation, 2003. RFC 3547.

[17] S. Bellantoni. *Predicative Recursion and Computational Complexity*. PhD thesis, University of Toronto, 1992.

[18] 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.

[19] M. Bellare, R. Canetti, and H. Krawczyk. A modular approach to the design and analysis of authentication and key exchange protocols. In *Proceedings of 30th Annual Symposium on the Theory of Computing*. ACM, 1998.

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

[21] 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.

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

[23] R. Bird, I. Gopal, A. Herzberg, P. Janson, S. Kutten, R. Molva, and M. Yung. Systematic design of a family of attack resistant authentication protocols. *IEEE Journal on Selected Areas in Communications*, 1(5), June 1993.

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

[25] L. Buttyan, S. Staamann, and U. Wilhelm. A simple logic for authentication protocol design. In *Proceedings of 11th IEEE Computer Security Foundations Workshop*, pages 153–162. IEEE, 1999.

[26] 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/`.

[27] R. Canetti and M. Fischlin. Universally composable commitments. In *Proc. CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 19–40, Santa Barbara, California, 2001. Springer.

[28] R. Canetti and H. Krawczyk. Universally composable notions of key exchange and secure channels. In *Advances in Cryptology—EUROCRYPT 2002*, volume 2332 of *Lecture Notes in Computer Science*, pages 337–351. Springer, 2002.

[29] R. Canetti, E. Kushilevitz, and Y. Lindell. On the limitations of universally composable two-party computation without set-up assumptions. In *Advances in Cryptology—EUROCRYPT 2003*, volume 2656 of *Lecture Notes in Computer Science*, pages 68–86. Springer, 2003.

[30] R. Canetti, Y. Lindell, R. Ostrovsky, and A. Sahai. Universally composable two-party and multi-party secure computation. In *Proc. ACM Symp. on the Theory of Computing*, pages 494–503, 2002.

[31] R. Canetti, C. Meadows, and P. Syverson. Environmental requirements for authentication protocols. In *Proceedings of Software Security - Theories and Systems, Mext-NSF-JSPS International Symposium, ISSS, LNCS 2609*, pages 339–355. Springer-Verlag, 2003.

[32] J. A. Clark and J. L. Jacob. Searching for a solution: Engineering tradeoffs and the evolution of provably secure protocols. In *Proceedings IEEE Symposium on Research in Security and Privacy*, pages 82–95. IEEE, 2000.

[33] 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. Springer-Verlag, 2005.

[34] I. Damgard and J. B. Nielsen. Universally composable efficient multiparty computation from threshold homomorphic encryption. In *Proc. CRYPTO 2003*, volume 2729 of *Lecture Notes in Computer Science*, pages 247–264, Santa Barbara, California, 2003. Springer.

[35] 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.

[36] 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.

[37] 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.

[38] A. Datta, A. Derek, J. C. Mitchell, and D. Pavlovic. A derivation system and compositional logic for security protocols. *Journal of Computer Security*, 2004. to appear.

[39] 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*, volume 83 of *Electronic Notes in Theoretical Computer Science*, 2004.

[40] A. Datta, A. Derek, J. C. Mitchell, A. Ramanathan, and A. Scedrov. The impossibility of realizable ideal functionality. Cryptology ePrint Archive, Report 2005/211, 2005.

[41] A. Datta, A. Derek, J. C. Mitchell, and B. Warinschi. Key exchange protocols: Security definition, proof method and applications, 2005. In preparation.

[42] A. Datta, R. Küsters, J.C. Mitchell, and A. Ramanathan. On the Relationships Between Notions of Simulation-Based Security. In J. Kilian, editor, *Proceedings of the 2nd Theory of Cryptography Conference (TCC 2005)*, volume 3378 of *Lecture Notes in Computer Science*, pages 476–494. Springer-Verlag, 2005.

[43] A. Datta, J. C. Mitchell, and D. Pavlovic. Derivation of the JFK protocol. Technical Report KES.U.02.03, Kestrel Institute, 2002.

[44] T. Dierks and C. Allen. The Tls Protocol Version 1.0, 1999. RFC 2246.

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

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

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

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

[49] 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.

[50] 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.

[51] R. Fagin, J. Y. Halpern, M. Y. Vardi, and Y. Moses. *Reasoning about knowledge*. MIT Press, 1995.

[52] R. W. Floyd. Assigning meaning to programs. In J. T. Schwartz, editor, *Mathematical Aspects of Computer Science: Proceedings of American Mathematics Society Symposia*, volume 19, pages 19–31, Providence RI, 1967. American Mathematical Society.

[53] A. Freier, P. Karlton, and P. Kocher. The SSL protocol version 3.0. `draft-ietf-tls-ssl-version3-00.txt`, November 18 1996.

[54] W. D. Goldfarb. The undecidability of the second-order unification problem. *Theoretical Computer Science*, 13:225–230, 1981.

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

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

[57] P. Gupta and V. Shmatikov. Towards computationally sound symbolic analysis of key exchange protocols. Cryptology ePrint Archive, Report 2005/171, 2005.

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

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

[60] C. He, M. Sundararajan, A. Datta, A. Derek, and J. C. Mitchell. A modular correctness proof of tls and ieee 802.11i. In *12th ACM Conference on Computer and Communications Security (CCS)*, 2005. To appear.

[61] 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.

[62] J. Herzog. The Diffie-Hellman key-agreement scheme in the strand-space model. In *Proceedings of 16th IEEE Computer Security Foundations Workshop*, pages 234–247, 2003.

[63] J. Herzog. *Computational Soundness for Standard Assumptions of Formal Cryptography*. PhD thesis, MIT, 2004.

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

[65] M. Hofmann. *Type Systems for Polynomial-Time Computation*. Habilitation Thesis, Darmstadt; see `www.dcs.ed.ac.uk/home/mxh/papers.html`, 1999.

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

[67] 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.

[68] M. Jakobsson J. A. Garay and P. MacKenzie. Abuse-free optimistic contract signing. In *Proceedings of the 19th Annual International Cryptology Conference on Advances in Cryptology*, pages 449–466. Springer-Verlag, 1999.

[69] 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. Springer-Verlag, 2005.

[70] D. Johnson, C. Perkins, and J. Arkko. Mobility Support in IPv6, 2004. RFC 3775.

[71] C. Kauffman. Internet Key Exchange (IKEv2) protocol, 2004. Internet Draft.

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

[73] S. Kent and R. Atkinson. Security architecture for the internet protocol, 1998. RFC 2401.

[74] H. Krawczyk. Sigma: The sign-and-mac approach to authenticated diffie-hellman and its use in the IKE protocols. In *Advances in Cryptology - CRYPTO 2003*, volume 2729, pages 400–425. Springer-Verlag Heidelberg, 2003.

[75] Hugo Krawczyk. The IKE-SIGMA protocol, 2002. Internet draft.

[76] P. D. Lincoln, J. C. Mitchell, M. Mitchell, and A. Scedrov. Probabilistic polynomial-time equivalence and security protocols. In Jeannette M. Wing, Jim Woodcock, and Jim Davies, editors, *Formal Methods World Congress, vol. I*, number 1708 in Lecture Notes in Computer Science, pages 776–793, Toulouse, France, 1999. Springer.

[77] P.D. Lincoln, J.C. Mitchell, M. Mitchell, and A. Scedrov. A probabilistic poly-time framework for protocol analysis. In *ACM Conf. Computer and Communication Security*, 1998.

[78] G. Lowe. An attack on the Needham-Schroeder public-key protocol. *Info. Proc. Letters*, 56:131–133, 1995.

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

[80] 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.

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

[82] 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.

[83] P. Martin-Lof. *Intuitionistic Type Theory*. Bibliopolis, 1984.

[84] 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.

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

[86] 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.

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

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

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

[90] 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.

[91] C. Meadows. Open issues in formal methods for cryptographic protocol analysis. In *Proceedings of DISCEX 2000*, pages 237–250. IEEE, 2000.

[92] C. Meadows and D. Pavlovic. Deriving, attacking and defending the gdoi protocol. In *ESORICS*, pages 53–72, 2004.

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

[94] 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.

[95] R. Milner. *Communication and Concurrency*. International Series in Computer Science. Prentice Hall, 1989.

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

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

[98] R. Milner. Action calculi, or syntactic action structures. In Andrzej M. Borzyszkowski and Stefan Sokolowski, editors, *Proceedings of MFCS'93*, volume 711 of *Lecture Notes in Computer Science*, pages 105–121. Springer, 1993.

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

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

[101] J. C. Mitchell, M. Mitchell, and A. Scedrov. A linguistic characterization of bounded oracle computation and probabilistic polynomial time. In *Proc. 39th Annual IEEE Symposium on the Foundations of Computer Science*, pages 725–733, Palo Alto, California, 1998. IEEE.

[102] 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.

[103] J. C. Mitchell, A. Ramanathan, A. Scedrov, and V. Teague. A probabilistic polynomial-time calculus for the analysis of cryptographic protocols (preliminary report). In Stephen Brookes and Michael Mislove, editors, *17th Annual Conference on the Mathematical Foundations of Programming Semantics, Arhus, Denmark, May, 2001*, volume 45. Electronic notes in Theoretical Computer Science, 2001.

[104] V. Shoup N. Asokan and M. Waidner. Asynchronous protocols for optimistic fair exchange. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 86–99. IEEE, 1998.

[105] 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.

[106] C. Neuman, T. Yu, S. Hartman, and K. Raeburn. The Kerberos Network Authentication Service (v5), 2005. RFC 4120.

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

[108] L. C. Paulson. Inductive analysis of the internet protocol TLS. *ACM Transactions on Information and System Security*, 2(3):332–351, 1999.

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

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

[111] D. Pavlovic. Categorical logic of names and abstraction in action calculi. *Mathematical Structures in Computer Science*, 7(6):619–637, 1997.

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

[113] A. Perrig and D. Song. A first step towards the automatic generation of security protocols. In *Proceedings of ISOC Network and Distributed Systems Security Symposium*, 2000.

[114] 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.

[115] A. Ramanathan, J. C. Mitchell, A. Scedrov, and V. Teague. Probabilistic bisimulation and equivalence for security analysis of network protocols. Unpublished, see `http://www-cs-students.stanford.edu/~ajith/`, 2003.

[116] A. Ramanathan, J. C. Mitchell, A. Scedrov, and V. Teague. Probabilistic bisimulation and equivalence for security analysis of network protocols. In *FOSSACS 2004 - Foundations of Software Science and Computation Structures*, March 2004.

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

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

[119] 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.

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

[121] P. Syverson and P. van Oorschot. On unifying some cryptographic protocol logics. In *Proc. IEEE Symposium on Research in Security and Privacy*, pages 14–28, 1994.

[122] F. J. Thayer-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.

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

[124] R. J. van Glabbeek, S. A. Smolka, and B. Steffen. Reactive, generative, and stratified models of probabilistic processes. *International Journal on Information and Computation*, 121(1), August 1995.

[125] 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.

[126] J. M. Wing. Beyond the horizon: A call to arms. IEEE Security and Privacy, 2003.

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

Appendix A

Cord Calculus

Cord calculus is the basic action structure [97, 98, 111] that we use to represent protocols. It was introduced in [49, 50]. Here we provide a brief summary. Cord calculus was inspired by the strand space formalism [122], 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 [22]. To represent the stores where the messages are received, we added variables, and a substitution mechanism expressed by simple reaction rules, corresponding to the basic communication and computation operations. The result is a simple process calculus, combining strand spaces and chemical abstract machine. This is our protocol execution model.

Its formal components are as follows.

A.1 Terms, Actions, Strands and Cords

A basic algebra of *terms* t is assumed to be given. As usually, 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 $\{|t|\}_K$, and the signature $\{|t|\}_{\overline{K}}$. The decryption and the

(names)	N	$::=$	$\hat{X}$	variable name
			$\hat{A}$	constant name
(agents)	P	$::=$	X	variable agent
			A	constant agent
(basic keys)	K_0	$::=$	k	constant key
			y	variable key
			N	name
(keys)	K	$::=$	K_0	basic key
			$\overline{K_0}$	inverse key
(terms)	t	$::=$	x	variable term
			c	constant term
			N	name
			P	agent
			K	key
			t, t	tuple of terms
			$\{\!\|t\|\!\}_K$	term encrypted with key K
			$\{\!\|t\|\!\}_{\overline{K}}$	term signed with key $\overline{K}$
(actions)	a	$::=$	ϵ	the null action
			$\langle t \rangle$	send a term t
			(x)	receive term into variable x
			(νx)	generate new term x
			(t/t)	match a term to a pattern
(strands)	S	$::=$	$aS \mid a$	

Table A.1: Syntax of terms, actions and strands

signature verification subsume under pattern matching. The dedicated operations could be readily added. We assume enough typing to distinguish the keys K from the agents A, the nonces n and so on. Each type is given with enough variables. As usually, the computation is modelled as term evaluation. The closed terms, that can be completely evaluated, can be sent as messages. The terms containing free variables cannot be sent until the variables are bound to some values, received or generated.

The language of *actions*, built upon the language of terms, describes communication and computation. The actions here include sending a term $\langle t \rangle$, receiving into a variable (x), matching a term against a pattern $(t/q(x))$, and creating a new value (νx). As appropriate, further actions can be added to the calculus. For instance, to model Kerberos we add the action "read time".

A list of actions is called *strand*[1]. The idea is that a strand represents a sequence of actions of an agent. For example, the strand $(\nu x)\langle x\rangle_A$ tells that the agent A generates a fresh value into the variable x and then sends it out as a message. In a strand, each of the actions (x), $(t/p(x))$ and (νx) all *binds* the free occurrences of x that appear on the right.[2] As usually, the bound variables are taken up to renaming, i.e. α-conversion. Operational semantics of the binding operations is given below, in terms of the substitution: e.g., a value is received in (x) is propagated through the occurrences of x bound to that operator.

A *cord* is an equivalence class of semantically indistinguishable strands, annotated by the name of the agent executing it. For some processes, the order of actions may be irrelevant, even unobservable: e.g., some constant streams c and d may be sent in arbitrary order, or in parallel, while the same process is independently receiving into a variable y. So the strands like $\langle c\rangle\langle d\rangle(y)$, $\langle d\rangle(y)\langle c\rangle$ etc. may be viewed as equivalent. When needed, the order of actions can be imposed using the tupling of variables. By identifying equivalent strands, we get cords. The equivalence class containing the strand S of agent A is written as $[S]_A$. We often elide the agent name, when it is irrelevant, or obvious. The inaction is denoted by the empty cord $[]$. We assume that $[] = []_X$ holds for all agents X. (There is just one silence.)

Since the semantic equivalence of strands does not play a role in the present paper, cords are here just lists of actions with variables, annotated by the agent names whenever nonempty. Table A.1 summarizes the formal definition of cords.

A.2 Cord Spaces, Agents and Processes

The idea of a *cord space* is that it represents a system of agents ready to engage in communication and distributed computation. Formally, a cord space is a nonempty multiset (bag) of cords, usually in the form $C = \{[S_1]_{A_1}, [S_2]_{A_2}, \ldots [S_k]_{A_k}\}$, where each S_i is a nonempty

[1]This is slightly more general than the original strands from [122], because the term calculus, underlying actions, contains variables and substitution.

[2]The tradition of denoting the operators binding x by the round brackets around it goes back to Milner [98, 97].

$$[S(x)S'] \otimes [T\langle t\rangle T'] \otimes C \quad \rhd\!\rhd \quad [SS'(t/x)] \otimes [TT'] \otimes C \tag{A.1}$$
$$[S\,(p(t)/p(x))\,S'] \otimes C \quad \rhd\!\rhd \quad [SS'(t/x)] \otimes C \tag{A.2}$$
$$[S(\nu x)S'] \otimes C \quad \rhd\!\rhd \quad [SS'(m/x)] \otimes C \tag{A.3}$$

Where the following conditions must be satisfied:
(A.1) $FV(t) = \emptyset$
(A.2) $FV(t) = \emptyset$
(A.3) $m \notin FV(S) \cup FV(S') \cup FV(C)$

Table A.2: Basic reaction steps

strand. The only cord space which is not in this form is $\{[]\}$. We abuse notation, and write it as $[]$. Cord spaces can thus be viewed as the elements of the free commutative monoid $(\mathcal{C}, \otimes, [])$ generated by cords. The monoid operation $\otimes$ is the union of multisets, except that $C \otimes [] = [] \otimes C = C$ holds by definition.

Table A.2 gives an operational semantics of cord spaces. In all rules, we assume that the name clashes are avoided by renaming the bound variables. The respective side conditions, required for each of the reactions, are shown in the same table. The substitution (t/x) acts on the strand to the left. Reaction (A.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 (A.2) represents basic pattern matching, where the cord matches the term $p(t)$ with the expected pattern $p(x)$, and substitutes t for x. Reaction (A.3) generates a fresh value m, and substitutes it for x in the cord to the right. The condition $FV(t) = \emptyset$, imposed on the first two reactions, means that a term cannot be sent, or tested, until all of its free variables have been instantiated, so that it can be evaluated. The condition $m \notin FV(S) \cup FV(S') \cup FV(C)$ on the last rule means that the value m, created by (νx), must be globally fresh. This is modeled by treating it as a variable which does not occur anywhere in the process.

Cord category

Category theory (see, e.g., [15]) is a general mathematical framework that is used in various ways in the study of algebra, semantics of computation, and logical foundations. Without going into all of the steps in detail, we describe a category of cords that highlights the main constructions that are used in this paper. This cord category involves both sequential and parallel composition of cords and cord spaces.

By definition, a *process* is a cord space, together with an explicit *input interface* consisting of a sequence of distinct variables and an explicit *output interface* consisting of a sequence of terms. A process s may be written in the form

$$s \quad = \quad (y_0 \dots y_{m-1}) S \langle t_0 \dots t_{n-1} \rangle$$

where $(y_0 \dots y_{m-1})$ is an input interface, S is a cord space, and $\langle t_0 \dots t_{n-1} \rangle$ is an output interface. If S is a single cord and all free variables of S are bound by the input interface, then we call this process a *closed cord*. Intuitively, the input variables $y_0, \dots y_{m-1}$ represent the *entry ports* and the output terms $t_0, \dots t_{n-1}$ are offered at the *exit ports*. While the input interface variables must be distinct, the output values need not be mutually different. The input interface binds the variables $y_0, \dots y_{m-1}$ and α-equivalents define the same process. In other words, renaming $y_0, \dots y_{m-1}$ throughout S and $t_0 \dots t_{n-1}$ yields another, equivalent representative of the same process.

The morphisms of the cord category $\mathcal{C}$ are processes given by α-equivalence classes of process expressions such as s above. The objects of the cord category $\mathcal{C}$ are the *arities* of such processes. An arity is a list of variables, such as $(y_0 \dots y_{m-1})$, modulo renaming. Ignoring the types of variables as before, an arity thus boils down to a number, in this case $m = \{0, 1, \dots m-1\}$. If the objects of the category $\mathcal{C}$ are identified with the natural numbers, then the process s written above becomes a morphism $s : m \longrightarrow n$. More intuitively, and in the tradition of functorial semantics, one might prefer to write the arity of m variables as the exponent A^m of some abstract ground type A (which itself corresponds to the generator 1 of the arities, since $A = A^1$). The process s thus becomes $s : A^m \longrightarrow A^n$. The formal justification for this will become clearer after we spell out the categorical structure of $\mathcal{C}$.

Given the morphisms

$$\begin{aligned} r &= (x_0 \ldots x_{\ell-1}) R \langle u_0 \ldots u_{m-1} \rangle \ : \ A^\ell \longrightarrow A^m \\ s &= (y_0 \ldots y_{m-1}) S \langle t_0 \ldots t_{n-1} \rangle \ : \ A^m \longrightarrow A^n \end{aligned}$$

their sequential composition is defined

$$(r; s) \quad = \quad (x_0 \ldots x_{\ell-1}) R S' \langle t'_0 \ldots t'_{n-1} \rangle \ : \ A^\ell \longrightarrow A^n$$

where S' and t'_i are the substitution instances of S and t_i, respectively, with each variable y_k replaced by the term u_k. In performing these substitutions, the variables must be chosen (or renamed) so that the free variables of S, t_j and u_k do not become bound in $r; s$. Intuitively, the cord space RS' corresponds to running in sequence, for each agent X, the actions of X in R followed by the actions of X in S'. This leads to the definition

$$RS' \quad = \quad \{[UV]_X \mid [U]_X \in R, [V]_X \in S'\}$$

A less syntactic and possibly more elegant view is that if the cord spaces R and S' are regarded as partially-ordered multisets (pomsets) of actions, then RS' is their concatenation in the usual sense for partial orders, putting R before S'. If R and S' have no common agents, then RS' is inactive. On the other hand, since $[] = []_X$ for all X, the process

$$\mathsf{id}_m \quad = \quad (y_0 \ldots y_m) [] \langle y_0 \ldots y_m \rangle \ : \ A^m \longrightarrow A^m$$

is the identity.

We may also define parallel composition on processes. Given, furthermore, a morphism $p : A^k \longrightarrow A^\ell$, in the form

$$p \quad = \quad (z_0 \ldots z_{k-1}) P \langle v_0 \ldots v_{\ell-1} \rangle$$

the parallel composition $p \otimes s \ : \ A^{k+m} \longrightarrow A^{\ell+n}$ may be defined

$$p \otimes s \quad = \quad (\vec{z}\vec{y}) P \otimes S \langle \vec{v}\vec{t} \rangle$$

with bound variables of p and s renamed so that the concatenation $\vec{z}\vec{y}$ of variables in their input interfaces produces a sequence of distinct variables. The parallel composition operator $\otimes$ forms a tensor on objects and morphisms, with the tensor unit I the arity A^0.

With this structure, $\mathcal{C}$ turns out to be the free monoidal category generated by the object $A = A^1$, and the morphisms $()[(\nu x)]_Y\langle x\rangle : I \longrightarrow I$, $()[(x)]_Y\langle x\rangle : I \longrightarrow I$ and $()[\langle x\rangle]_Y\langle x\rangle : I \longrightarrow A$, corresponding to the basic actions, together with the variable morphisms $()[]\langle x\rangle : I \longrightarrow A$, and the generic abstraction operators, binding the variables to the entry ports. Formally, this follows from the results of [111]. Intuitively, the universal property of $\mathcal{C}$ can perhaps be understood by noticing, first of all, that a process in the form $(y_0, \ldots y_{m-1})[]\langle v_0, \ldots v_{n-1}\rangle : m \longrightarrow n$, where $\{v_0, \ldots v_{n-1}\} \subseteq \{y_0, \ldots y_{m-1}\}$ represents a function from n to m (backwards!). This is a trivial process, assigning to each exit port a unique entry port, from which it simply copies the values. The subcategory of $\mathcal{C}$ spanned by such processes is thus isomorphic with the opposite of the category of finite sets and functions. This subcategory of $\mathcal{C}$ is thus the free cartesian category over one generating object $A = A^1$ representing the arity 1. In particular, the morphisms $(xy)[]\langle x\rangle$ and $(xy)[]\langle y\rangle$ are the projections, and $(x)[]\langle xx\rangle$ is the diagonal for the cartesian products, whereas $(x)[]\langle\rangle$ is the unique map to the unit type $I = A^0$. However, when we add the morphisms where agents send and receive messages, and generate fresh values, and close all that under the categorical operations, variables and abstraction, we get the cord category, which is still monoidal, but not cartesian (because the projections and the diagonals are not natural, polymorphic operations with respect to the morphisms with nontrivial actions).

For simplicity, we have so far systematically ignored the typing of the terms and variables in cord calculus. In fact, any type structure of the term language of cords may be directly reflected on the generated cord category. If we distinguish a type K of keys, for example, the cord category will be generated not by one, but by two generators, say A and K; if we also distinguish nonces, there will be three generators. Less trivially, if the type of keys is taken to be *indexed* over the type of agents, as the family $K(X)$, where $X : \mathsf{Agents}$, then the arities will not be just lists of independent variables, as above, but will be the contexts of *dependent types*. Since a key variable $x : K(X)$ can be assigned only after the variable $X : \mathsf{Agent}$ on which it depends has been assigned, the interfaces will need to display typing and dependencies. The precise syntax for such arities can be found on the early pages of [83], for example. The objects of the resulting cord category are then the closed type expressions, not just in the form $A^k = \prod_k A$, but also e.g. $\prod_{X:A} K(X)$.

The combination of reaction rules and the categorical structure of cord spaces constitutes our process model. While the dynamic binding of variables, defined by the reaction rules, captures the communication and the computation in cord spaces, the static binding, defined by the categorical operations, allows composition of agents from operations, or from various component processes; and it also allows sharing ports and resources between different agents statically, i.e., without sending any messages. Composition of processes allows composition of protocols. Sharing resources allows modeling principals and attackers, which can play several roles in one or more protocol sessions, and be subdivided into agents in various ways.

A.3 Protocols

A *protocol* $\mathcal{Q}$ is defined by a finite set of roles, such as initiator, responder and server, each specified by a closed cord describing actions to be executed in a single instance of a role. A *principal* is a set of agents sharing all static data, such as keys, but directly, and not by messages. This is formalized using static binding, described above. We will denote principals by $\hat{A}, \hat{B}$, etc. An agent executing a single instance of a particular role will be called *thread*. 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.

A.3.1 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. 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 [105] and a model presented by Dolev and Yao [47].

A.3.2 Buffer cord

Cords reactions, as we defined them, can only 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* $[(x)\langle x\rangle]$, it models a mesage being received and than eventually send. We will require that all send and receive actions by principals and the intruder are performed via buffer cords and assume that in every protocol there are enough instances of the buffer cord to guarantee delivery of every message. Buffer cords are a part of the infrastructure rather than a part of the protocol, we assume that they are executed by special nameless agents. Unless otherwise specified, when we refer to a thread, we mean a non-buffer thread, similarly, when we refer to an action, we mean an action performed by a non-buffer thread.

A.3.3 Configurations and runs

Initial configuration of a protocol $\mathcal{Q}$ is determinded 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) An intruder cord, 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.

A.3.4 Events and traces

Since the protocol logic we introduce 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 $\langle m \rangle$ is a send event of A. Alternatively, we can a look at a run as a linear sequence of events starting from an initial configuration.

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

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

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

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*.

A.3.5 Protocol properties

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

Lemma A.3.1. (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 role 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 process, where each variable is bound either statically, to some entry port, or dynamically, to some receive action, or fresh value generation, or to a pattern match. □

Lemma A.3.2. (Asynchronous communication) *In every run, any thread that wished to send a message can always eventually 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 A.3.3. *For every receive action there is a corresponding send action. More formally,*
$EVENT(R, X, (x), m, x) \supset \exists Y.EVENT(R, Y, \langle m \rangle, \emptyset, \emptyset)$.

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

Lemma A.3.4. *For any initial configuration* $\mathbf{C}$ *of protocol $\mathcal{Q}$, and any run R, if agent $\hat{X} \in HONEST(\mathbf{C})$, then for any thread X performed by principal $\hat{X}$, $R|_X$ is a trace of a single role of $\mathcal{Q}$ executed by X.*

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

Appendix B

Semantics of Protocol Logic

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

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

Action Formulas:

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

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

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

Other Formulas:

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

  $$
  \begin{aligned}
  &(\mathsf{Has}_0(A, m) \text{ if } ((m \in FV(R|_A)) \\
  &\quad \vee EVENT(R, A, (\nu x), m, x) \\
  &\quad \vee EVENT(R, A, (x), m, x) \\
  &\text{and } \mathsf{Has}_{i+1}(A, m) \text{ if } \mathsf{Has}_i(A, m) \vee (\mathsf{Has}_i(A, m') \\
  &\quad \vee (\mathsf{Has}_i(A, m') \wedge \mathsf{Has}_i(A, m'') \\
  &\qquad \wedge ((m = m', m'') \vee (m = m'', m'))) \\
  &\quad \vee (\mathsf{Has}_i(A, m') \wedge \mathsf{Has}_i(A, K) \\
  &\qquad \wedge m = \{\!|m'|\!\}_K) \\
  &\quad \vee (\mathsf{Has}_i(A, a) \wedge \mathsf{Has}_i(A, g^b) \\
  &\qquad \wedge m = g^{ab}) \\
  &\quad \vee (\mathsf{Has}_i(A, g^{ab}) \wedge m = g^{ba})
  \end{aligned}
  $$

  Intuitively, Has_0 holds for terms that are known directly, either as a free variable of the role, or as the direct result of receiving or generating the term. Has_{i+1} holds for terms that are known by applying i operations (decomposing via pattern matching, composing via encryption or tupling, or by computing a Diffie-Hellman secret) to terms known directly.

- $\mathcal{Q}, R \models \mathsf{Fresh}(A, m)$ if $\mathcal{Q}, R \models (\diamondminus \mathsf{New}(A, m) \vee (\diamondminus \mathsf{New}(A, n) \wedge m = g(n))) \wedge \neg(\diamondminus \mathsf{Send}(A, m') \wedge m \subseteq m')$.

- $\mathcal{Q}, R \models \mathsf{Honest}(\hat{A})$ 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 basing sequences of roles in Q.

- $\mathcal{Q}, R \models \mathsf{Contains}(t_1, t_2)$ if $t_2 \subseteq t_1$.
- $\mathcal{Q}, R \models (\phi_1 \wedge \phi_2)$ if $\mathcal{Q}, R \models \phi_1$ and $\mathcal{Q}, R \models \phi_2$
- $\mathcal{Q}, R \models \neg\phi$ if $\mathcal{Q}, R \not\models \phi$
- $\mathcal{Q}, R \models \exists x.\phi$ if $\mathcal{Q}, R \models (d/x)\phi$, for some d, where $(d/x)\phi$ denotes the formula obtained by substituting d for x in ϕ.
- $\mathcal{Q}, R \models \diamondminus\phi$ if $\mathcal{Q}, R' \models \phi$, where R' is a (not necessarily proper) prefix of R. Intuitively, this formula means that in some state in the past, formula ϕ is true.
- $\mathcal{Q}, R \models \ominus\phi$ if $\mathcal{Q}, R' \models \phi$, where $R = R'e$, for some event e. Intuitively, this formula means that $\ominus\phi$ is true in a state if ϕ is true in the previous state.
- $\mathcal{Q}, R \models \mathsf{Start}(X)$ if $R|_X$ is empty. Intuitively this formula means that X didn't execute any actions in the past.

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 $R_1|_A$.

Appendix C

Soundness of Axioms and Proof Rules

In this section we prove the soundness of the axioms and proof rules used in the proof system, hence proving Theorem 4.2.1. Since the logic and the proof system are an extension of the earlier work [49, 50], here we only give brief and informal proofs for those axioms that are same or similar to axioms in [50]. We omit proofs for standard axioms and rules of temporal logic. Also, we show that the composition methodology is sound by proving Lemmas 4.1.2 and 4.1.6.

C.1 Axioms for protocol actions

AA1 $\phi[a]_X \diamondminus \mathsf{a}$

Informally, this axiom says that if a is an action, and a a corresponding action formula, when thread X executes a, in the resulting state $\diamondminus \mathsf{a}$ 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 $\mathcal{Q}, R_0R_1 \models \diamondminus \sigma\mathsf{a}$. 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 temporal operator "$\diamondminus$", it has to be that $\mathcal{Q}, R_0R_1 \models \diamondminus \sigma\mathsf{a}$. Now, by the definition of modal formulas we have $Q \models \phi\,[a]_X \diamondminus \mathsf{a}$.

AA2 $\mathsf{Fresh}(X,t)[a]_X \diamondminus (\mathsf{a} \wedge \ominus\mathsf{Fresh}(X,t))$

Informally, this axiom says that if a term t is fresh in some state, then it remains fresh at least until the corresponding thread executes an action. 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 $\mathcal{Q}, R_0 \models \sigma\mathsf{Fresh}(X,t)$, we need to prove that $\mathcal{Q}, R_0R_1 \models \sigma\diamondminus(\mathsf{a} \wedge \ominus\mathsf{Fresh}(X,t))$. Since $R_1|_X$ matches a under substitution σ, R_1 has to contain action $\sigma\mathsf{a}$. Let R_1' be a prefix of R_1 containing all actions preceding action σa. It holds trivially $Q, R_0R_1'\sigma a \models \sigma a$. On the other hand, $Q, R_0 \models \sigma\mathsf{Fresh}(X,t)$. Clearly, R_1 does not contain any actions by thread X, and it follows from semantics of the predicate "Fresh" that the validity of the formula $\mathsf{Fresh}(X,t)$ depends only on actions done by X in the past. Therefore, $Q, R_0R_1 \models \sigma\mathsf{Fresh}(X,t)$, and hence $Q, R_0R_1\sigma a \models \ominus\sigma\mathsf{Fresh}(X,t)$, by the semantics of the temporal operator "$\ominus$". Finally, by the semantics of the temporal operator "$\diamondminus$", it has to be that $\mathcal{Q}, R_0R_1 \models \sigma\diamondminus(\mathsf{a} \wedge \ominus\mathsf{Fresh}(X,t))$.

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

Informally, this axiom says that fresh nonces are secret. 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 [50].

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

Informally, this axiom 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".

ARP $\diamondminus\mathsf{Receive}(X,p(x))[(q(x)/q(t))]_X\ \diamondminus\mathsf{Receive}(X,p(t))$

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 $\mathcal{Q}, R_0 \models \sigma\diamondminus\mathsf{Receive}(X,p(x))$, we need to prove that $\mathcal{Q}, R_0R_1 \models \sigma\diamondminus\mathsf{Receive}(X,p(t))$. 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 $\mathcal{Q}, R_0 \models$

$\diamondminus \mathsf{Receive}(X, p(t))$. Clearly, formulas of the form $\diamondminus$ a remain valid as new actions are executed, hence $\mathcal{Q}, R_0R_1 \models \sigma \diamondminus \mathsf{Receive}(X, p(t))$.

C.2 Possession axioms

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

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

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

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

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 $\mathbf{ENC}$, 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 $\mathsf{Has}_{i+1}(X, \{|x|\}_K)$.

$\mathbf{ORIG}$ $\quad \diamondminus \mathsf{New}(X, n) \supset \mathsf{Has}(X, n)$

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

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" and temporal operator "$\diamondminus$".

C.3 Encryption and signature

$\mathbf{SEC}$ $\quad \mathsf{Honest}(\hat{X}) \wedge \diamondminus\ \mathsf{Decrypt}(Y, \{\!|n|\!\}_X) \supset (\hat{Y} = \hat{X})$

Informally, $\mathbf{SEC}$ says that if an agent $\hat{X}$ is honest, and some thread Y executed by principal $\hat{Y}$ has decrypted a message $\{\!|n|\!\}_X$ (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 [50].

$\mathbf{VER}$ $\quad \mathsf{Honest}(\hat{X}) \wedge \diamondminus\ \mathsf{Verify}(Y, \{\!|n|\!\}_{\overline{X}}) \wedge \hat{X} \neq \hat{Y} \supset$
$\exists X. \exists m. (\diamondminus\ \mathsf{Send}(X, m) \wedge \mathsf{Contains}(m, \{\!|n|\!\}_{\overline{X}}))$

Informally, $\mathbf{VER}$ says that if an agent $\hat{X}$ is honest, and some thread Y executed by principal $\hat{Y}$ has verified a signature $\{\!|n|\!\}_X$ (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 $\hat{X} \in \mathit{HONEST}(\mathbf{C})$. Suppose that R is a run of $\mathcal{Q}$ starting from $\mathbf{C}$, such that $\mathcal{Q}, R \models \diamondminus\ \mathsf{Verify}(Y, \{\!|n|\!\}_{\overline{X}})$. By the definition of the execution model, when $\hat{X} \in \mathit{HONEST}(\mathbf{C})$, only threads of $\hat{X}$ can construct signatures with $\hat{X}$'s private key. Since, $\hat{X} \neq \hat{Y}$, it has to be that the thread Y received term $\{\!|n|\!\}_{\overline{X}}$ as a part of some message m', i.e. there exists a term m' such that $\mathit{EVENT}(R, Y, (x), m', x)$ and $\{\!|n|\!\}_{\overline{X}} \subseteq m'$. By Lemma A.3.3 there is a corresponding send action for every receive, hence there exists a thread Z such that $\mathit{EVENT}(R, Z, \langle m \rangle, \emptyset, \emptyset)$ is true. Therefore, there exists at least one action in the run R where $\{\!|n|\!\}_{\overline{X}}$ 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 $\{\!|n|\!\}_{\overline{X}} \subseteq m$, it is true that $\mathit{EVENT}(R', Z, \langle m \rangle, \emptyset, \emptyset)$. By Lemma A.3.1 $\{\!|n|\!\}_{\overline{X}}$ has to be either received or generated by Z, since R' is the shortest run in which $\{\!|n|\!\}_{\overline{X}}$ is sent out as a part of some message it has to be that the thread Z generated $\{\!|n|\!\}_{\overline{X}}$. By the definition of the execution model, and honesty of $\hat{X}$ it follows that Z is a thread of $\hat{X}$. Now, $Q, R \models$

$\diamondminus \mathsf{Send}(Z, m) \wedge \mathsf{Contains}(m, \{|n|\}_{\overline{Z}}))$ holds by the semantics of temporal operators and Lemma A.3.2.

C.4 Uniqueness of Nonces

N1 $\diamondminus \mathsf{New}(X, n) \wedge \diamondminus \mathsf{New}(Y, n) \supset (X = Y)$

Informally, this axiom says that nonces are unique across different threads. If two threads X and Y have generated the same nonce n in the past, then it must be the case that $X = Y$. The soundness of this axiom follows directly from the definition of the execution model and the semantics of the predicate "New".

N2 $\mathsf{After}(\mathsf{New}(X, n_1), \mathsf{New}(X, n_2)) \supset (n_1 \neq n_2)$

Informally, this axiom says that nonces are unique within the same thread. If some thread X generated two nonces n_1 and n_2 by two distinct actions, then n_1 and n_2 must be different. The soundness of this axiom follows directly from the definition of the execution model and the semantics of the predicate "New".

F1 $\diamondminus \mathsf{Fresh}(X, n) \wedge \diamondminus \mathsf{Fresh}(Y, n) \supset (X = Y)$

Informally, this axiom says that the freshness is local. If some nonce n is fresh in two different threads X and Y then it must be that $X = Y$. The soundness of this axiom follows directly from the definition of the execution model and the semantics of the predicate "Fresh".

C.5 Subterm relationship

CON $\mathsf{Contains}((x, y), x) \wedge \mathsf{Contains}((x, y), y)$

Informally, this axiom states that a tuple contains its parts. Informally, this axiom states that a tuple contains its parts. The soundness of this axiom follows directly from the semantics of the predicate "Contains".

C.6 Modal axioms

P1 $\mathsf{Persist}(X,t)[a]_X\mathsf{Persist}(X,t)$ where $\mathsf{Persist} \in \{\diamondminus\, \phi, \mathsf{Has}\}$

Informally this axiom says that the for some formulas stay valid when an agent 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, the fact that some formula was true in the past remains valid when add additional actions to the run.

P3 $\mathsf{HasAlone}(X,n)[a]_X\mathsf{HasAlone}(X,n)$, where $n \not\subseteq_v a$ or $a \neq \langle m \rangle$

Informally this axiom says that a nonce n remains secret as long as it is not send out as a part of some message m. The soundness of this axiom follows from the semantics of the predicate "Has" and Lemmas A.3.1 and A.3.3.

F $\phi[\langle m \rangle]_X\neg\mathsf{Fresh}(X,t)$, where $(t \subseteq m)$

Informally, this axiom talks about loss of freshness. A value is not fresh anymore after it is send out as a part of some message. The soundness of this axiom follows directly from the semantics of the predicate "Fresh".

C.7 Temporal ordering of actions

AF0 $\mathsf{Start}(X)[]_X\ \neg\diamondminus\, \mathsf{a(X,t)}$

Informally, this axiom says that before a thread X executes any action, it is true that X did not execute any actions in the past. The soundness of this axiom follows directly from the semantics of the predicate "Start", semantics of modal formulas, and the temporal operator "$\diamondminus$".

AF1 $\theta[a_1 \dots a_n]_X$ $\mathsf{After}(\mathsf{a_1}, \mathsf{a_2}) \wedge \dots \wedge \mathsf{After}(\mathsf{a_{n-1}}, \mathsf{a_n})$

Informally, this axiom says that after an agent does some actions $\mathsf{a_1}, \dots, \mathsf{a_n}$ in that order, it is true that $\mathsf{After}(\mathsf{a_i}, \mathsf{a_{i+1}})$ for all $i = 1, \dots, n-1$. The soundness of this axiom follows directly from the definition of "After" and semantics of temporal operators.

AF2 $(\diamondminus \mathsf{b_1}(X, t_1) \wedge \ominus \mathsf{Fresh}(X, t)) \wedge \diamondminus \mathsf{b_2}(Y, t_2) \supset$
$\mathsf{After}(\mathsf{b_1}(X, t_1), \mathsf{b_2}(Y, t_2))$**,where** $t \subseteq t_2$ **and** $X \neq Y$

Informally, this axiom 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 "Fresh", semantics of temporal operators and Lemmas A.3.1 and A.3.3.

C.8 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 $\mathsf{Has}(X, g^{ab}) \supset (\mathsf{Computes}(X, g^{ab})$
$\vee\, \exists m.(\diamondminus\ \mathsf{Receive}(X, m) \wedge \mathsf{Contains}(m, g^{ab})))$

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

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 A.3.3, 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".

C.9 Generic rules

$$\frac{\theta[P]_X\phi \quad \theta[P]_X\psi}{\theta[P]_X\phi \wedge \psi}\,\mathbf{G1} \qquad \frac{\theta[P]_X\phi \quad \theta' \supset \theta \quad \phi \supset \phi'}{\theta'[P]_X\phi'}\,\mathbf{G2} \qquad \frac{\phi}{\theta[P]_X\phi}\,\mathbf{G3}$$

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 axiom **G2** can be verified similarly. Axiom **G3** is trivially valid because if ϕ is true after any run, then ϕ is true after a specific run that contains actions P.

C.10 Sequencing rule

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

Sequencing rule **S1** 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 $\mathcal{Q}$ is a protocol and R is a run of $\mathcal{Q}$ such that $\mathcal{Q}, R \models \phi_1[P]_A\phi_2$ and $\mathcal{Q}, R \models \phi_2[P']_A\phi_3$. We need to prove that $\mathcal{Q}, R \models \phi_1[PP']_A\phi_3$. Let $R = R_0R_1R_2$, assume that $R_1|_A$ matches PP' under substitution σ, and $Q, R_0 \models \sigma\phi_1$. Run R can be written as $R = R_0R_1'R_1''R_2$ 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 $\mathcal{Q}, R_0R_1'R_1'' \models \sigma\phi_3$.

C.11 The Honesty rule

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

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 A.3.4, 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**.

C.12 Composition theorems

Proof of Lemma 4.1.2

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 $\forall\rho \in \mathcal{Q}_1 \cup \mathcal{Q}_2.\forall P\epsilon BS(\rho).\ \vdash \phi\ [P]_X\ \phi$. 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, $\vdash_{\mathcal{Q}_1|\mathcal{Q}_2} \mathsf{Honest}(\hat{X}) \supset \phi$.

Proof of Lemma 4.1.6

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 $\mathsf{Start}(X) \vdash []_X \phi$ 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 $\mathbf{S1}$. Therefore, by the application of the honesty rule, $\vdash_{\mathcal{Q}} \mathsf{Honest}(\hat{X}) \supset \phi$.