# A derivation system and compositional logic for security protocols

**Article** *in* Journal of Computer Security · August 2005

DOI: 10.3233/JCS-2005-13304 · Source: DBLP



**4 authors**, including:

Ante Derek
University of Zagreb
**21** PUBLICATIONS **1,006** CITATIONS

Journal of Computer Security 13 (2005) 423–482
IOS Press


# A derivation system and compositional logic for security protocols

Anupam Datta [a,*], Ante Derek [a], John C. Mitchell [a] and Dusko Pavlovic [b]
[a] *Computer Science Department, Stanford University, Stanford, CA 94305-9045, USA*
*E-mail: {danupam,aderek,jcm}@cs.stanford.edu*
[b] *Kestrel Institute, Palo Alto, CA 94304, USA*
*E-mail: dusko@kestrel.edu*

Many authentication and key exchange protocols are built using an accepted set of standard concepts such as Diffie–Hellman key exchange, nonces to avoid replay, certificates from an accepted authority, and encrypted or signed messages. We propose a general framework for deriving security protocols from simple components, using composition, refinements, and transformations. As a case study, we examine the structure of a family of key exchange protocols that includes Station-To-Station (STS), ISO-9798-3, Just Fast Keying (JFK), IKE and related protocols, deriving all members of the family from two basic protocols. In order to associate formal proofs with protocol derivations, we extend our previous security protocol logic with preconditions, temporal assertions, composition rules, and several other improvements. Using the logic, which we prove is sound with respect to the standard symbolic model of protocol execution and attack (the "Dolev–Yao model"), the security properties of the standard signature based Challenge-Response protocol and the Diffie–Hellman key exchange protocol are established. The ISO-9798-3 protocol is then proved correct by composing the correctness proofs of these two simple protocols. Although our current formal logic is not sufficient to modularly prove security for all of our current protocol derivations, the derivation system provides a framework for further improvements.

## 1. Introduction

While many historical authentication and secrecy protocols, such as those cataloged by Clark and Jacob [13], may be analyzed independently, modern protocols often have a number of different subprotocols and interrelated modes. The Internet Key Exchange (IKE) protocol [27], for example, offers digital signature authentication, public-key encryption-based authentication, and pre-shared key authentication, each potentially used in one of several modes (e.g., Main Mode, Aggressive Mode, Quick Mode, New Group Mode). In both protocol design and protocol analysis, it is essential to understand a complex protocol in a systematic way, characterizing properties that are independent of specific modes or options and clearly understanding the security differences between different options.

* Corresponding author. Tel.: +1 650 723 1658; Fax: +1 650 725 4671.

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 [21], with efforts to formalize the practice appearing in [8]. More recently, Bellare, Canetti and Krawczyk [5], 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 paper, we describe a methodology for deriving protocols from components, and an associated logic for reasoning about components and their composition. As a step toward a general theory, we examine the structure of a family of key exchange protocols that includes Station-To-Station (STS), ISO-9798-3, Just Fast Keying (JFK) and related protocols, showing how all the protocols in this family may be derived systematically. The protocol derivation system for this class of protocols consists of two base protocol components, a composition operation, three transformations, and seven refinements. The two protocol components are Diffie–Hellman key exchange [20] and a two-message signature-based challenge and response authentication protocol. The refinements (which add data to message fields) include extending messages by certificates in order to discharge the assumption that each participant knows the other's public key. The transformations include moving data from a later message to an earlier one, and reordering messages using a denial-of-service prevention "cookie" technique. While additional refinements and transformations may be needed to derive other kinds of protocols, we have found in subsequent work [17] that the relatively small set described in this paper covers some interesting ground.

Our long-term goal is to develop a logic in which protocol correctness proofs follow the derivation steps. With each protocol component, we hope to prove properties that will be sufficient to reason about useful protocols built from that component. When components are composed, composition proof rules presented in this paper allow us to conclude properties of the composition. Although we do not present a systematic way of reasoning about a general class of refinements and transformations, we hope the logic originally presented in [23,24] and further developed in this paper can be extended to modular proofs that follow the structure of many protocol derivations. With this goal in mind, we extend our previous security protocol logic in several ways, prove the logic sound with respect to the standard symbolic model of protocol execution and attack (the "Dolev–Yao model"), and use the formal logic to establish security properties of protocols involving the standard signature based

Challenge-Response protocol and the Diffie–Hellman key exchange protocol are established.

The logic presented in [23,24] uses assertions of the form $[\text{actions}]_A\phi$, which states that after principal $A$ performs the indicated actions, the formula $\phi$ will be true, regardless of actions of any malicious attacker. By proving such assertions about each role of a protocol, security properties of the protocol may be established. We extend the logic of [23,24] with preconditions, temporal operators, the ability to reason more accurately about multiple roles executed by the same principal, asynchronous communication through buffers, signatures, and Diffie–Hellman exponentiation. The main reason for preconditions is to allow reasoning about protocol composition, as described in the next paragraph. Temporal operators allow us to express authentication properties, in the form of matching conversations. For example, a postcondition of the form $\mathsf{ActionsInOrder}(\mathsf{Send}(X,\ldots),\mathsf{Receive}(Y,\ldots),\mathsf{Send}(Y,\ldots),\mathsf{Receive}(X,\ldots))$, expressible using temporal operator $\diamondminus$ referring to the past and $\ominus$ referring to the previous state, says that actions of principals $X$ and $Y$ occurred in a certain order. If principal $X$ can be sure of this order, then $X$ is assured that $Y$ saw certain actions occur in a certain order. The addition of signatures and Diffie–Hellman exponentiation extend the range of applicability of the logic.

Conceptually, reasoning about protocol composition involves two basic problems. The first may be called *additive combination* – we wish to combine protocol components in a way that accumulates security properties. 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. An interesting illustration of the significance of nondestructive combination is the construction in [30], which shows that for every security protocol there is another protocol that interacts with it insecurely.

Intuitively, additive combination is captured in the logic by a before-after formalism for reasoning about steps in protocol execution. Suppose $P$ is a sequence of protocol steps, and $\phi$ and $\psi$ 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 $\phi$ is true before principal $A$ does actions $P$, then $\psi$ 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.

Nondestructive combination 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 [27] which contain a large number of sub-protocols.

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

The rest of the paper is organized as follows. Section 2 describes the main ideas underlying the protocol derivation system. In Section 3, we present the derivations of the STS family of key exchange protocols. Section 4 discusses the logic, using which we prove security properties of protocols, and methods for formalizing protocol composition. To illustrate the use of these methods, the *ISO-9798-3* protocol is formally derived from two component subprotocols based on the Diffie–Hellman key exchange protocol and the signature-based Challenge-Response protocol. Section 5 surveys previous work on protocol derivation and composition. Finally, Section 6 presents our conclusions and propose some interesting themes for future work.

## 2. Derivation framework

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. Since the present paper uses key exchange protocols as a worked example, we take Diffie–Hellman key exchange as a basic component. A *composition* operation combines two protocols. Parallel composition and sequential composition with term substitution 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 paper is to show how protocol composition, refinement, and transformation may be used by working out some examples and expanding our formal proof system. 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 [21], the standard signature-based challenge-response protocol [46], JFKi, JFKr, ISO-9798-3 [2], and the core of the IKE protocol [27].

## 3. 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.

### *3.1. 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 [20] 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 [46])

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

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.

### *3.2. Composition*

The composition operation used is sequential composition of two protocol components with term substitution. The precise definition of this operation is in Section 4.4. 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.

### *3.3. 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 $R_1$.* $SIG_X(m) \Rightarrow E_K(SIG_X(m))$, 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 in this paper, everything signed is public. So, an attacker can verify guesses at identities of a principal if the signature is not encrypted.

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

*Refinement* $R_4$. $SIG_X(m) \Rightarrow SIG_X(m, ID_Y)$, 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 [53]), 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 [33].

*Refinement* $R_5$. $g^x \Rightarrow g^x, n_x$, 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* $R_6$. $SIG_X(m) \Rightarrow SIG_X(m), ID_X$, 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* $R_7$. $E_K(m) \Rightarrow E_K(m), HMAC_{K'}(role, E_K(m))$, 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.

### 3.4. 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 in this paper adds a nonce from an earlier message into the signed portion of a later message, as illustrated in Fig. 1.

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 Fig. 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 Fig. 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 3.5. 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

$$\begin{array}{ll} I \longrightarrow R : m & I \longrightarrow R : m \\ R \longrightarrow I : n, SIG_R(m) \quad \Longrightarrow & R \longrightarrow I : n, SIG_R(n, m) \\ I \longrightarrow R : SIG_I(n) & I \longrightarrow R : SIG_I(m, n) \end{array}$$

Fig. 1. An example of a binding transformation.

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

Fig. 2. An example of a cookie transformation.

round-trip communication is established with the initiator. The cookie transformation is described in detail in [19], where it is derived using more primitive operations. Here, we only touch on the main idea.

An example of a cookie transformation is shown in Fig. 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.

### 3.5. 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 Fig. 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 $C_2$.

$$
\begin{array}{l}
I \longrightarrow R : m \\
R \longrightarrow I : SIG_R(m) \\
R \longrightarrow I : n \\
I \longrightarrow R : SIG_I(n)
\end{array}
$$

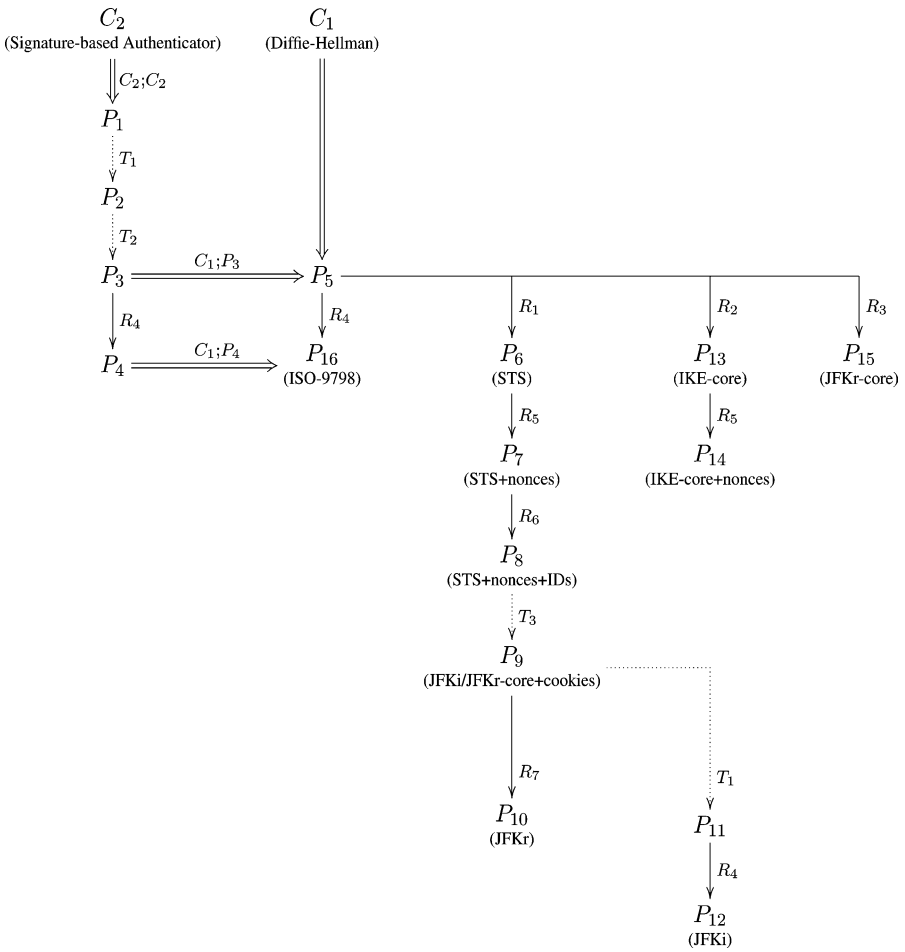


Fig. 3. Derivation graph of the STS protocol family.

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.

$$I \longrightarrow R : m$$
$$R \longrightarrow I : n, SIG_R(m)$$
$$I \longrightarrow R : SIG_I(n)$$

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

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

$$I \longrightarrow R : m$$
$$R \longrightarrow I : n, SIG_R(n, m)$$
$$I \longrightarrow R : SIG_I(m, n)$$

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 $P_3$.

$$I \longrightarrow R : g^i$$
$$R \longrightarrow I : g^r, SIG_R\big(g^r, g^i\big)$$
$$I \longrightarrow R : SIG_I\big(g^i, g^r\big)$$

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 Fig. 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 [32] 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.

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

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 [34]. It is not quite clear whether that attack breaks mutual authentication.

*Protocol* $P_7$. Obtained by applying refinement $R_5$ to protocol $P_6$.

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

$P_7$ 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 $P_7$.

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

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 certifi-

cates apriori is discharged by explicitly exchanging certificates alongside the signatures.

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

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

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.

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

$P_{10}$ 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 $P_9$.

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

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.

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

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 [34]. $P_{12}$ 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 [27].

*Protocol* $P_{13}$. Obtained by applying refinement $R_2$ to protocol $P_5$. This protocol has been described as the core for IKE in [2].

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

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 $P_{13}$.

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

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 [2].

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

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

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.

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

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.

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

$P_3$ 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.

### *3.6. Other issues*

#### *3.6.1. 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.

#### *3.6.2. 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.

## 4. Logical formalization

In this section, we present a formalization of a part of the derivation system. We describe a formal language called *cord calculus* for representing protocols and a logic for reasoning about protocol properties. With protocols expressed as cords, the composition operations of the derivation system become syntactic operations on sets of cords. To illustrate protocol derivation as a more formal operation, the *ISO-9798-3* protocol is derived by composing a Diffie–Hellman component and a challenge-response component. This corresponds to the step in the derivation tree for the STS family where $C_1$ and $P_4$ are composed to yield $P_{16}$. In continuing work, we have extended cord calculus and the protocol logic to reason about a class of refinements and transformations (see [17] for a preliminary report).

Cord calculus and a form of protocol logic were initially developed in [23,24]. While we use cord calculus in essentially its original form, the protocol logic has been significantly extended and some original concepts have been revised. The most important extension is a rigorous treatment of protocol composition. Other extensions include preconditions, temporal assertions, and a compositional variant of the honesty rule. Furthermore, session-ids are incorporated into predicates in order to distinguish actions executed by a principal in different sessions. An axiomatic treatment of digital signatures and Diffie–Hellman key exchange is also presented.

The rest of this section is organized as follows. Section 4.1 describes cord calculus. The syntax and semantics of the protocol logic are presented in Section 4.2. The core proof system is detailed in Section 4.3. The extension of the proof system with composition rules and the general methodology for proving protocol composition results is discussed in Section 4.4. Finally, in Section 4.5, we present the formal modular proof of the *ISO-9798-3* protocol.

### *4.1. 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 [23], we used a form of process calculus that we call cords. *Cords* form an action structure [47,48,55], based on $\pi$-calculus [50], and related to spi-calculus [1]. The cords formalism is also similar to the approach of the Chemical Abstract Machine formalism [7], in that the communication actions can be viewed as

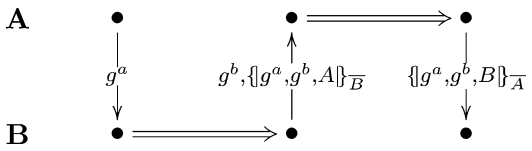


Fig. 4. *ISO-9798-3* as arrows-and-messages.

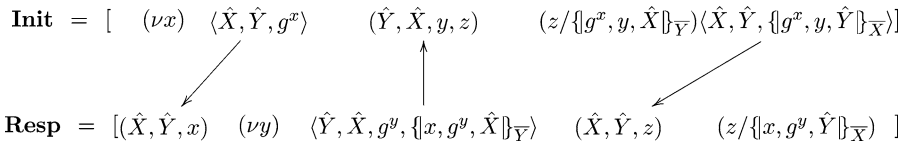


Fig. 5. Cords for *ISO-9798-3* .

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. Although cord calculus is presented in [23,24], a brief summary is included in Appendix A to make this paper more self-contained.

In this section, we show how protocols are represented in cord calculus with an example. Figure 4 shows the *ISO-9798-3* protocol [29] in the informal arrows-and-messages notation. The roles of the same protocol are written out as cords in Fig. 5, writing $\hat{X}$ and $\hat{Y}$ for the agents executing cords **Init** and **Resp**, respectively. The arrows between the 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 Fig. 5. 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 $(\nu x)$, $\langle t \rangle$, $(x)$ 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 4.5 for a complete example).

### *4.2. Protocol logic*

#### *4.2.1. Syntax*

The formulas of the logic are given by the grammar in Table 1, where $\rho$ may be any role, written using the notation of cord calculus. Here, $t$ and $P$ denote a

Table 1
Syntax of the logic

| |
|---|
| Action formulas |
| $\mathsf{a} ::= \mathsf{Send}(P, m) \mid \mathsf{Receive}(P, m) \mid \mathsf{New}(P, t) \mid \mathsf{Decrypt}(P, t) \mid \mathsf{Verify}(P, t)$ |
| Formulas |
| $\phi ::= \mathsf{a} \mid \mathsf{Has}(P, t) \mid \mathsf{Fresh}(P, t) \mid \mathsf{Honest}(N) \mid \mathsf{Contains}(t_1, t_2) \mid \phi \wedge \phi \mid \neg\phi \mid \exists x.\phi \mid \diamondminus\phi \mid \ominus\ \phi \mid$ $\mathsf{Start}(P)$ |
| Modal formulas |
| $\Psi ::= \phi\ \rho\ \phi$ |

term and a *thread*, respectively. A thread is the 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 $\phi$ and $\psi$ 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 $\theta$ is true, formula $\phi$ 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 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 [36]. Since we view a run as a linear sequence of states, $\diamondminus\,\phi$ means that in some state in the past $\phi$ holds, whereas $\ominus\ \phi$ means that in the previous state $\phi$ 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 [61], the temporal requirements language for the NRL Protocol Analyzer [43,44]. 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 [21] which requires that whenever $\hat{A}$ and $\hat{B}$ accept each other's identities at the end of a run, their records of the run should match, i.e., each message that $\hat{A}$ sent was received by $\hat{B}$ and vice versa, each send event happened before the corresponding receive event, and moreover the messages sent by each principal ($\hat{A}$ or $\hat{B}$) appear in the same order in both the records. Including the source and destination fields in the message allows us to match up send-receive actions. Since in this paper, we reason about correctness of a protocol in an environment in which other protocols may be executing concurrently, it is important that when $\hat{A}$ and $\hat{B}$ accept each other's identities, they also agree on which protocol they have successfully completed with the other. One way to extend the matching histories characterization to capture this requirement is by adding protocol identifiers to messages. Now if $\hat{A}$ and $\hat{B}$ have matching histories at the end of a run, not only do they agree on the source, destination and content of each message, but also on which protocol this run is an instance of.

*4.2.2. 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 $\phi$ 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 $\phi$ has free variables, then $Q, R \models \phi$ if we have $Q, R \models \sigma\phi$ for all substitutions $\sigma$ that eliminate all the free variables in $\phi$. 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 $\mathsf{a}$ 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 $\mathsf{a}$ was true in some state, i.e., if the action had occurred in the past.

### *4.3. 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.

#### *4.3.1. Axioms for protocol actions*

The axioms about protocol actions are listed in Table 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 **AA1** and **AA2** is any one of the 5 actions and $\mathsf{a}$ is the corresponding predicate in the logic. **AA1** states that if a principal has executed an action in some role, then the corresponding predicate asserting that the action had occurred in the past is true. **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 **AN2** says that no one else knows $n$, and **AN3** says that $n$ is fresh.

#### *4.3.2. Axioms relating atomic predicates*

Table 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. **ORIG** and **REC** state respectively that a principal possesses a term if she freshly generated it (a nonce) or if she received it in some message. **TUP** and **ENC** enable construction of tuples and encrypted terms if the parts are known. **PROJ** and **DEC** allow decomposition of a tuple into its components and decryption of an encrypted term if the key is known. The next two axioms are aimed at capturing the *black-box model of*

Table 2
Axioms for protocol actions

| | |
|---|---|
| **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
Basic axioms

Possession Axioms:

**ORIG** $\diamondsuit \mathsf{New}(X, n) \supset \mathsf{Has}(X, n)$

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

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

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

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

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

Encryption and Signature:

**SEC** $\mathsf{Honest}(\hat{X}) \wedge \diamondsuit \mathsf{Decrypt}(Y, \{|n|\}_X) \supset (\hat{Y} = \hat{X})$

**VER** $\mathsf{Honest}(\hat{X}) \wedge \diamondsuit \mathsf{Verify}(Y, \{|n|\}_{\overline{X}}) \wedge \hat{X} \neq \hat{Y} \supset$

$\exists X. \exists m. (\diamondsuit \mathsf{Send}(X, m) \wedge \mathsf{Contains}(m, \{|n|\}_{\overline{X}}))$

Uniqueness of Nonces:

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

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

**F1** $\diamondsuit \mathsf{Fresh}(X, t) \wedge \diamondsuit \mathsf{Fresh}(Y, t) \supset (X = Y)$

Subterm Relation:

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

*encryption and signature*. **VER** refers to the unforgeability of signatures while **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 **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. **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 $\diamondsuit(\mathsf{b} \wedge \ominus\diamondsuit\mathsf{a})$). **F1** states that fresh values generated in different threads are distinct. **N1**, **N2**, and **F1** together capture the intuition that fresh nonces and Diffie–Hellman exponentials are unique. Finally, **CON** states that a term contains its subterms.

### *4.3.3. Modal axioms and rules*

Table 4 collects the inference rules and some additional axioms. The generic inference rules follow naturally from the semantics. **G2** is exactly of the same form as the

Table 4
Modal axioms and rules

Generic Rules:

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

Sequencing rule:

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

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 $t \not\sqsubseteq a$ or $a \neq \langle m\rangle$

**P3** $\mathsf{HasAlone}(X,n)[a]_X\mathsf{HasAlone}(X,n)$, where $n \not\sqsubseteq_v a$ 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 $(t \sqsubseteq m)$

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 $\mathsf{Fresh}$ predicate is preserved only if the fresh term $t$ is not sent out in a message (see **P2**). 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 $P'$.

*4.3.4. 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 5). See [56] for a complete axiomatization of PLTL. The axioms and rules specific to the temporal ordering of actions are presented in Table 5. We use $\mathsf{a}_1, \ldots, \mathsf{a}_n$, to denote action formulas corresponding to actions $a_1, \ldots, a_n$. Similarly, $\mathsf{b}_1$ and $\mathsf{b}_2$ stand for any action predicates. The rules are fairly straightforward. **AF0** simply states that before a thread $X$ executes any action, it is true that $X$ did not execute any actions in the past. **AF1** orders the actions within a role. This is consistent with the way we view a role as an ordered sequence of actions. **AF2** uses the freshness of terms to reason about the ordering of actions carried out by different threads. Intuitively, **AF2** states that if a thread $X$ has a fresh value $t$ at some point in the run and than

Table 5
Axioms and rules for temporal ordering

PLTL Axioms:

**T1** $\diamondminus(\phi \wedge \psi) \supset (\diamondminus \phi \wedge \diamondminus \psi)$

**T2** $\diamondminus(\phi \vee \psi) \supset (\diamondminus \phi \vee \diamondminus \psi)$

**T3** $\ominus\neg\phi \leftrightarrow \neg \ominus \phi$

Temporal Generalization Rule:

$$\frac{\phi}{\neg\diamondminus\neg\phi}\textbf{TGEN}$$

Temporal Ordering of actions:

$$\mathsf{After}(a, b) \equiv \diamondminus(b \wedge \ominus\diamondminus a)$$

$$\mathsf{ActionsInOrder}(a_1, \ldots, a_n) \equiv \mathsf{After}(a_1, a_2) \wedge \cdots \wedge \mathsf{After}(a_{n-1}, a_n)$$

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

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

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

executes action $b_1(X, t_1)$, then any action $b_2(Y, t_2)$ 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 $b_1$.

*4.3.5. 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 [36] 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 [23,24], we reformulate the rule here is a more convenient form using preconditions and postconditions.

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

Recall that a protocol $\mathcal{Q} = \{\rho_1, \rho_2, \ldots, \rho_k\}$ 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 $\rho$, written $P \in BS(\rho)$, if $P$ is a contiguous subsequence of $\rho$ such that either (i) $P$ starts at the beginning of $\rho$ 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 $\rho$ 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.

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

no free variable in $\phi$ except $X$ bound in $[P\,]_X$

In words, if $\phi$ 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 $\phi$. The side condition prevents free variables in the conclusion $\mathsf{Honest}(\hat{X}) \supset \phi$ from becoming bound in any hypothesis. As explained in [23,24], 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.

*4.3.6. 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 $\gamma$ is provable from the formulas in $\Gamma$ and any axiom or inference rule of the proof system except the honesty rule ($\mathbf{HON_Q}$ for any protocol $\mathcal{Q}$). We write $\Gamma \vdash_{\mathcal{Q}} \gamma$ if $\gamma$ is provable from the formulas in $\Gamma$, 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 $\gamma$ is either a modal formula or a basic formula (i.e., of the syntactic form $\Psi$ or $\phi$ in Table 1).

**Theorem 4.1.** *If* $\Gamma \vdash_{\mathcal{Q}} \gamma$, *then* $\Gamma \models_{\mathcal{Q}} \gamma$. *Furthermore, if* $\Gamma \vdash \gamma$, *then* $\Gamma \models \gamma$.

*4.4. Formalizing protocol composition*

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 4.1, the STS protocol is defined by two cords, one each for the initiator and responder role of the protocol.

**Definition 4.2** (Parallel Composition). The parallel composition $\mathcal{Q}_1 \otimes \mathcal{Q}_2$ of protocols $\mathcal{Q}_1$ and $\mathcal{Q}_2$ is the union of the sets of cords $\mathcal{Q}_1$ and $\mathcal{Q}_2$.

For example, consider the protocol obtained by parallel composition of SSL 2.0 and SSL 3.0. The definition above allows an honest principal to simultaneously engage in sessions of the two protocols. Clearly, a property proved about either protocol individually might no longer hold when the two are run in parallel, since an adversary might use information acquired by executing one protocol to attack the other. Formally, some step in the logical proof of the protocol property is no longer correct. Since all the axioms and inference rules in Section 4.3 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. Prove separately the security properties of protocols $\mathcal{Q}_1$ and $\mathcal{Q}_2$.

   $\vdash_{\mathcal{Q}_1} \Psi_1$ *and* $\vdash_{\mathcal{Q}_2} \Psi_2$

[1] A preliminary version of this result was presented in [16,18].

2. Identify the set of invariants used in the two proofs, $\Gamma_1$ and $\Gamma_2$. 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,

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

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

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

4. Prove that the invariants, $\Gamma_1 \cup \Gamma_2$, hold for both the protocols. This step uses the transitivity of entailment in the logic: if $\vdash_Q \Gamma$ and $\Gamma \vdash \gamma$, then $\vdash_Q \gamma$. Since $\vdash_{Q_1} \Gamma_1$ was already proved in Step 1, in this step, it is sufficient to show that $\vdash_{Q_1} \Gamma_2$ and similarly that $\vdash_{Q_2} \Gamma_1$. By Lemma 4.3 below, we therefore have $\vdash_{Q_1 \otimes Q_2} \Gamma_1 \cup \Gamma_2$. 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.

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

**Lemma 4.3.** *If $\vdash_{Q_1} \phi$ and $\vdash_{Q_2} \phi$, then $\vdash_{Q_1 \otimes Q_2} \phi$, where the last step in the proof of $\phi$ in both $\mathcal{Q}_1$ and $\mathcal{Q}_2$ uses the honesty rule and no previous step uses the honesty rule.*

**Theorem 4.4.** *If $\vdash_{Q_1} \Gamma$ and $\Gamma \vdash \Psi$ and $\vdash_{Q_2} \Gamma$, then $\vdash_{Q_1 \otimes Q_2} \Psi$.*

**Definition 4.5** (Sequential Composition). A protocol $\mathcal{Q}$ is the sequential composition of two protocols $\mathcal{Q}_1$ and $\mathcal{Q}_2$, if each role of $\mathcal{Q}$ is obtained by the sequential composition of a cord of $\mathcal{Q}_1$ with a cord of $\mathcal{Q}_2$.

**Definition 4.6** (Sequential Composition of Cords). Given closed cords $r = (x_0 \dots x_{\ell-1})[R]_X \langle u_0 \dots u_{m-1} \rangle$, $s = (y_0 \dots y_{m-1})[S]_Y \langle t_0 \dots t_{n-1} \rangle$, their sequential composition is defined by

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

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 $\alpha$-rename bound variables to avoid variable capture.

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

The sequencing rule, **S1** (see Table 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 $\mathcal{Q}_2$.

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

2. Identify the set of invariants used in the two proofs, $\Gamma_1$ and $\Gamma_2$. The formulas included in these sets will typically be the formulas in the two proofs, which

were proved using the honesty rule. The proofs from the previous step can be decomposed into two parts – the first part proves the protocol invariants using the honesty rule for the protocol, while the second proves the protocol property using the invariants as hypotheses, but without using the honesty rule. Formally,

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

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

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

4. If the post-condition of the modal formula $\Psi_1$ matches the pre-condition of $\Psi_2'$, then the two can be sequentially composed by applying the sequencing rule **S1**. Here $\Psi_2'$ is obtained from $\Psi_2$ 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 $\Psi_1$ and $\Psi_2'$ are respectively $\theta[P_1]\phi$ and $\phi[P_2]\psi$, we have:

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

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 4.7, 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.

$$\vdash_{Q_3} \theta[P_1 P_2]\psi$$

**Lemma 4.7.** *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* $\phi$ *in both* $\mathcal{Q}_1$ *and* $\mathcal{Q}_2$ *uses the honesty rule and no previous step uses the honesty rule.*

**Theorem 4.8.** *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* $\vdash_{Q_3} \theta[P_1 P_2]\psi$*, where* $\mathcal{Q}_3$ *is a sequential composition of* $\mathcal{Q}_1$ *and* $\mathcal{Q}_2$*.*

### 4.5. An example of protocol composition

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 Fig. 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 4.5.1 and present the complete formal proofs in Section 4.5.2.

#### 4.5.1. Compositional proof sketch

As illustrated in Section 4.4, 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* [6] and is similar to the idea of proving authentication using *correspondence assertions* [64]. The same basic idea is also presented in [21] 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 7 in Section 4.5.2. 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:

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

The *actions* in the formula are the actions of the initiator cord of $CR$, given in Section 4.4. 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 $\phi_{auth}$ 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, $\Gamma_2$, contains only one formula (line (7) of Table 7).

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

Intuitively, this invariant states that whenever honest $\hat{Y}$ signs a term which is a triple with 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 6 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. **DH1** states that if $X$ can compute the Diffie–Hellman secret, then she also possesses it. **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

Table 6
Diffie–Hellman axioms

| | |
|---|---|
| **DH1** | $\mathsf{Computes}(X, g^{ab}) \supset \mathsf{Has}(X, g^{ab})$ |
| **DH2** | $\mathsf{Has}(X, g^{ab}) \supset$ |
| | $(\mathsf{Computes}(X, g^{ab}) \vee \exists m.(\diamondsuit\, \mathsf{Receive}(X, m) \wedge \mathsf{Contains}(m, g^{ab})))$ |
| **DH3** | $(\diamondsuit\, \mathsf{Receive}(X, m) \wedge \mathsf{Contains}(m, g^{ab})) \supset$ |
| | $\exists Y, m'.(\mathsf{Computes}(Y, g^{ab}) \wedge \diamondsuit\, \mathsf{Send}(Y, m') \wedge \mathsf{Contains}(m', g^{ab}))$ |
| **DH4** | $\mathsf{Fresh}(X, a) \supset \mathsf{Fresh}(X, g^a)$ |
| $\mathsf{Computes}(X, g^{ab}) \equiv$ | $(\ (\mathsf{Has}(X, a) \wedge \mathsf{Has}(X, g^b)) \vee (\mathsf{Has}(X, b) \wedge \mathsf{Has}(X, g^a))\ )$ |

containing it. **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, $\Gamma_1$, 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 4.4. 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:

$$\Gamma_1 \vdash \mathsf{Start}(X)\big[\mathbf{Init_{DH_0}}\big]_X \mathsf{Fresh}(X, g^x) \tag{1}$$

$$\Gamma_2 \vdash \mathsf{Fresh}(X, x)\big[\mathbf{Init_{CR}}\big]_X \phi_{auth} \tag{2}$$

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

$$\Gamma_2 \vdash \mathsf{Start}(X)\big[\mathbf{Init_{DH_0}}\big]_X \mathsf{Fresh}(X, g^x) \tag{3}$$

$$\Gamma_2 \vdash \mathsf{Fresh}(X, x)\big[\mathbf{Init_{CR}}\big]_X \phi_{auth} \tag{4}$$

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.

$$\Gamma_2\big[g^x/x\big] \vdash \mathsf{Fresh}\big(X, g^x\big)\big[\mathbf{Init_{CR'}}\big]_X \phi_{auth}\big[g^x/x\big] \tag{5}$$

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 **S1**. The resulting formula is:

$$\Gamma_2\big[g^x/x\big] \vdash \mathsf{Start}(X)\big[\mathbf{Init_{DH_0}}; \mathbf{Init_{CR'}}\big]_X \phi_{auth}\big[g^x/x\big] \tag{6}$$

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 $\Gamma_2$ are still satisfied. Finally, using the honesty rule, it is easily proved that $DH_0$ respects the environmental invariants in $\Gamma_2$ (Step 5). Therefore, from Lemma 4.7, we conclude that the sequential composition of $DH_0$ and $CR$, which is *ISO-9798-3* , respects the invariants in $\Gamma_2$. 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.

$$\vdash \mathsf{Start}(X)\big[\mathbf{Init_{DH_0}}\big]_X \mathsf{HasAlone}(X, x) \tag{7}$$

$$\vdash \mathsf{HasAlone}(X, x)\big[\mathbf{Init_{CR'}}\big]_X \mathsf{HasAlone}(X, x) \tag{8}$$

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

$$\vdash \mathsf{Start}(X)\big[\mathbf{Init_{DH_0}}; \mathbf{Init_{CR'}}\big]_X \mathsf{HasAlone}(X, x) \tag{9}$$

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:

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

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 8 in Section 4.5.2.

*4.5.2. Formal correctness proofs of protocols*

The proof of the shared secret property of *ISO-9798-3* is given in Table 8. 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 7. 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 7 naturally breaks down into three parts:

Table 7
Deductions of $\hat{X}$ executing **Init** role of Challenge-Response protocol

| | |
|---|---|
| **AA2, P1** | $\mathsf{Fresh}(X,x)[\langle \hat{X},\hat{Y},x\rangle(\hat{Y},\hat{X},y,z)(z/\{\|x,y,\hat{X}\|\}_{\overline{Y}})\langle \hat{X},\hat{Y},\{\|x,y,\hat{Y}\|\}_{\overline{X}}\rangle]_X$ |
| | $\diamondminus(\mathsf{Send}(X,\{\hat{X},\hat{Y},x\}) \wedge \ominus\mathsf{Fresh}(X,x))$ (1) |
| **AA1, P1** | $\mathsf{Fresh}(X,x)[\langle \hat{X},\hat{Y},x\rangle(\hat{Y},\hat{X},y,z)(z/\{\|x,y,\hat{X}\|\}_{\overline{Y}})\langle \hat{X},\hat{Y},\{\|x,y,\hat{Y}\|\}_{\overline{X}}\rangle]_X$ |
| | $\diamondminus\,\mathsf{Verify}(X,\{\|x,y,\hat{X}\|\}_{\overline{Y}})$ (2) |
| **AF1, ARP** | $\mathsf{Fresh}(X,x)[\langle \hat{X},\hat{Y},x\rangle(\hat{Y},\hat{X},y,z)(z/\{\|x,y,\hat{X}\|\}_{\overline{Y}})\langle \hat{X},\hat{Y},\{\|x,y,\hat{Y}\|\}_{\overline{X}}\rangle]_X$ |
| | $\mathsf{ActionsInOrder}($ |
| | $\mathsf{Send}(X,\{\hat{X},\hat{Y},x\}),$ |
| | $\mathsf{Receive}(X,\{\hat{Y},\hat{X},y,\{\|x,y,\hat{X}\|\}_{\overline{Y}}\}),$ |
| | $\mathsf{Send}(X,\{\hat{X},\hat{Y},\{\|x,y,\hat{Y}\|\}_{\overline{X}}\}))$ (3) |
| (3), **F1, P1, G2** | $\mathsf{Fresh}(X,x)[\mathbf{Init_{CR}}]_X \neg\diamondminus\,\mathsf{Fresh}(Y,x)$ (4) |
| **VER** | $\mathsf{Honest}(\hat{Y}) \wedge \diamondminus\,\mathsf{Verify}(X,\{\|x,y,\hat{X}\|\}) \supset$ |
| | $\exists Y.\exists x'.(\diamondminus\,\mathsf{Send}(Y,x') \wedge \mathsf{Contains}(x',\{\|x,y,\hat{X}\|\}_{\overline{Y}}))$ (5) |
| (2), (5), **P1, G1 − 3** | $\mathsf{Fresh}(X,x)[\mathbf{Init_{CR}}]_X\,\mathsf{Honest}(\hat{Y}) \supset$ |
| | $\exists Y.\exists x'.(\diamondminus\,\mathsf{Send}(Y,x') \wedge \mathsf{Contains}(x',\{\|x,y,\hat{X}\|\}_{\overline{Y}}))$ (6) |
| **HON** | $\mathsf{Honest}(\hat{Y}) \supset (((\diamondminus\,\mathsf{Send}(Y,x_0) \wedge$ |
| | $\mathsf{Contains}(x_0,\{\|x,y,\hat{X}\|\}_{\overline{Y}}) \wedge \neg\diamondminus\,\mathsf{Fresh}(Y,x)) \supset$ |
| | $(\ x_0 = \{\hat{Y},\hat{X},\{y,\{\|x,y,\hat{X}\|\}_{\overline{Y}}\}\} \wedge$ |
| | $\diamondminus(\mathsf{Send}(Y,\{\hat{Y},\hat{X},\{y,\{\|x,y,\hat{X}\|\}_{\overline{Y}}\}\}) \wedge \ominus\mathsf{Fresh}(Y,y)) \wedge$ |

Table 7
(Continued)

$$\mathsf{ActionsInOrder}(\mathsf{Receive}(Y, \{\hat{X}, \hat{Y}, x\}), \mathsf{Send}(Y, \{\hat{Y}, \hat{X}, \{y, \{\|x, y, \hat{X}\|_{\overline{Y}}\}\}))))) \quad (7)$$

(4), (6), (7), **G1 − 3** $\mathsf{Fresh}(X, x)[\mathbf{Init_{CR}}]_X \mathsf{Honest}(\hat{Y}) \supset$

$$\exists Y.\ \diamondminus\,(\mathsf{Send}(Y, \{\hat{Y}, \hat{X}, \{y, \{\|x, y, \hat{X}\|_{\overline{Y}}\}\}) \wedge \ominus\mathsf{Fresh}(Y, y)) \wedge \mathsf{After}(\mathsf{Receive}(Y, \{\hat{X}, \hat{Y}, x\}), \mathsf{Send}(Y, \{\hat{Y}, \hat{X}, \{y, \{\|x, y, \hat{X}\|_{\overline{Y}}\}\})) \quad (8)$$

(1), **AF2** $\mathsf{Fresh}(X, x)[\mathbf{Init_{CR}}]_X \diamondminus\,\mathsf{Receive}(Y, \{\hat{X}, \hat{Y}, x\})) \supset$

$$\mathsf{After}(\mathsf{Send}(X, \{\hat{X}, \hat{Y}, x\}), \mathsf{Receive}(Y, \{\hat{X}, \hat{Y}, x\})) \quad (9)$$

(3), **AF2** $\mathsf{Fresh}(X, x)[\mathbf{Init_{CR}}]_X \mathsf{Send}(Y, \{\hat{Y}, \hat{X}, \{y, \{\|x, y, \hat{X}\|_{\overline{Y}}\}\}) \wedge \ominus\mathsf{Fresh}(Y, y) \supset$

$$\mathsf{After}(\mathsf{Send}(Y, \{\hat{Y}, \hat{X}, \{y, \{\|x, y, \hat{X}\|_{\overline{Y}}\}\}), \mathsf{Receive}(X, \{\hat{Y}, \hat{X}, y, \{\|x, y, \hat{X}\|\}\}) \quad (10)$$

(8), (9), (10), **AF2** $\mathsf{Fresh}(X, x)[\mathbf{Init_{CR}}]_X \mathsf{Honest}(\hat{Y}) \supset$

$$\exists Y.\ \mathsf{ActionsInOrder}(\mathsf{Send}(X, \{\hat{X}, \hat{Y}, x\}), \mathsf{Receive}(Y, \{\hat{X}, \hat{Y}, x\}), \mathsf{Send}(Y, \{\hat{Y}, \hat{X}, \{y, \{\|x, y, \hat{X}\|_{\overline{Y}}\}\}), \mathsf{Receive}(X, \{\hat{Y}, \hat{X}, y, \{\|x, y, \hat{X}\|\}\})) \quad (11)$$

- 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 **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,$msg_2$),

Table 8
Deductions of $\hat{X}$ executing **Init** role of $CR'$ protocol

| | | |
|---|---|---|
| **P3** | $\mathsf{HasAlone}(X,x) \wedge \mathsf{Fresh}(X,g^x)[\mathbf{Init}_{\mathbf{CR}'}]_X\ \mathsf{HasAlone}(X,x)$ | (1) |
| $CR$ | $\mathsf{HasAlone}(X,x) \wedge \mathsf{Fresh}(X,g^x)[\mathbf{Init}_{\mathbf{CR}'}]_X \mathsf{Honest}(\hat{Y}) \supset$ $\exists Y.\mathsf{ActionsInOrder}($ $\mathsf{Send}(X,\{\hat{X},\hat{Y},g^x\}),$ $\mathsf{Receive}(Y,\{\hat{X},\hat{Y},g^x\}),$ $\mathsf{Send}(Y,\{\hat{Y},\hat{X},\{n,\{\|g^x,n,\hat{X}\|_{\overline{Y}}\}\}),$ $\mathsf{Receive}(X,\{\hat{Y},\hat{X},n,\{\|g^x,n,\hat{X}\|_{\overline{Y}}\}))$ | (2) |
| **HON** | $\mathsf{Honest}(\hat{Y}) \wedge \diamondsuit\mathsf{Send}(Y,\{\hat{Y},\hat{X},\{n,\{\|g^x,n,\hat{X}\|_{\overline{Y}}\}\}) \supset$ $\exists y'.(n = g^{y'} \wedge \mathsf{HasAlone}(Y,y'))$ | (3) |
| (2), (3) | $\mathsf{HasAlone}(X,x) \wedge \mathsf{Fresh}(X,g^x)[\mathbf{Init}_{\mathbf{CR}'}]_X \mathsf{Honest}(\hat{Y}) \supset$ $\exists Y.\ \exists y.(n = g^y \wedge \mathsf{HasAlone}(Y,y))$ | (4) |
| **AA1**, **REC**, **PROJ**, **P1** | $\mathsf{HasAlone}(X,x) \wedge \mathsf{Fresh}(X,g^x)[\mathbf{Init}_{\mathbf{CR}'}]_X \mathsf{Has}(X,n)$ | (5) |
| (1), (4), (5), **Computes** | $\mathsf{HasAlone}(X,x) \wedge \mathsf{Fresh}(X,g^x)[\mathbf{Init}_{\mathbf{CR}'}]_X \mathsf{Honest}(\hat{Y}) \supset$ $\exists Y.\exists y.(n = g^y \wedge \mathsf{Computes}(X,g^{xy}))$ | (6) |
| (1), (4), **Computes** | $\mathsf{HasAlone}(X,x) \wedge \mathsf{Fresh}(X,g^x)[\mathbf{Init}_{\mathbf{CR}'}]_X \mathsf{Honest}(\hat{Y}) \supset$ $\exists Y.\exists y.(n = g^y \wedge (\mathsf{Computes}(Z,g^{xy}) \supset (Z = X \vee Z = Y))))$ | (7) |
| (6), (7) | $\mathsf{HasAlone}(X,x) \wedge \mathsf{Fresh}(X,g^x)[\mathbf{Init}_{\mathbf{CR}'}]_X \mathsf{Honest}(\hat{Y}) \supset$ $\exists Y.\exists y.(n = g^y \wedge \mathsf{Computes}(X,g^{xy}) \wedge (\mathsf{Computes}(Z,g^{xy}) \supset$ $(Z = X \vee Z = Y))))$ | (8) |
| **DH2**, **DH3** | $\mathsf{Has}(X,g^{xy}) \supset (\mathsf{Computes}(X,g^{xy}) \vee \exists Y,m'.$ $(\mathsf{Computes}(Y,g^{xy}) \wedge \diamondsuit\ \mathsf{Send}(Y,m') \wedge \mathsf{Contains}(m',g^{xy}))$ | (9) |
| **HON** | $\mathsf{Honest}(\hat{Y}) \supset (\mathsf{Computes}(Y,g^{xy}) \supset$ $\neg\exists m'.(\diamondsuit\ \mathsf{Send}(Y,m') \wedge \mathsf{Contains}(m',g^{xy})))$ | (10) |
| (8), (9), (10), **DH1** | $\mathsf{HasAlone}(X,x) \wedge \mathsf{Fresh}(X,g^x)[\mathbf{Init}_{\mathbf{CR}'}]_X \mathsf{Honest}(\hat{Y}) \supset$ $\exists Y.\exists y.(\mathsf{Has}(X,g^{xy}) \wedge (\mathsf{Has}(Z,g^{xy}) \supset (Z = X \vee Z = Y)))$ | (11) |

$\mathsf{Receive}(\mathrm{A},msg_2)$). 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.

## 5. Related work

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 [8], 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 [21]. More recently, Bellare, Canetti and Krawczyk [5] have studied two interesting protocol transformations, which they call *authenticators*, which generically add authentication to a given protocol scheme.

In [57], 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 [60]. Independently, Clark and Jacob developed a similar approach for protocol synthesis [14]. They use genetic algorithms to search the space of protocols expressible in BAN logic [9]. Other works on using formal logic for protocol design include [3,10].

An important sub-problem in modular construction and analysis of security protocols is 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 [25] or something very similar to it [28]. 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 [45], 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 [63]. 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 [62]. A somewhat different approach is used by Lynch [35] 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 [12], 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 paper represent significant progress in this direction. The methods presented in Section 4.4 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 [27]. Also, Guttman and Fábrega have

proved a theoretical result to the same effect in their strand space model [26]. Another rule of thumb (also recommended by Kelsey, Schneier and Wagner in [30]), 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 [37,39–42], 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 several differences between the work described in this paper and some other protocol analysis efforts. To begin with, our basic model of protocol execution and possible attacker actions is the traditional "Dolev–Yao model" [22,53] that has been used in many other efforts [31,43,51,54]. While it is an important research direction to relate our model to computational models such as [11,52,58,59], we currently believe that "computational soundness" of symbolic methods is a separable goal that will lead to greater use of the kind of logical methods considered in this paper. At a more detailed level, there are some important differences between the way that we reason about incremental protocol construction and alternative approaches such as "universal composability" [11]. 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.

## 6. Conclusions and future work

We have presented a method for systematically deriving security protocols from basic components using a set of protocol composition, refinement and transformation steps. The goal has been to formalize the well-established practice of presenting protocols incrementally, starting from simple components and refining them by features and functions. As a case study, we examined the structure of the STS family of key exchange protocols in this system. The complete derivation graph is shown

in Figure 3. It shows how the various security properties – secrecy, authentication, DoS protection etc. – accumulate as the derivation proceeds. The study of the security properties of the STS family seems to be relevant since it includes protocols like IKE which are actually deployed on the internet and JFKi and JFKr which are currently being considered by IETF as replacements for IKE.

As an initial step towards associating formal proofs with protocol derivations, we have extended a previous protocol logic with preconditions and temporal assertions and proved the logic sound over the symbolic "Dolev–Yao" model of protocol execution and attack. The protocol composition operation of the derivation system is formalized within the proof system using three inference rules. The *ISO-9798-3* protocol is then formally derived from the standard challenge response protocol and a protocol that forms the essence of the Diffie–Hellman key exchange protocol. The logical derivation corresponds to the step in Fig. 3, where $P_9$ is derived from $C_1$ and $P_4$. It shows how the secrecy and authentication properties of Diffie–Hellman and challenge-response are preserved by the composition process.

There are several different directions in which this work could be extended. One direction is to develop derivation graphs for other sets of related protocols. Another family of protocols that might be interesting to look at is the Needham–Schroeder protocol family. This family will include well-known protocols like Kerberos, Otway–Rees, etc. The connecting point between these protocols is that they all use encryption for achieving authenticated key exchange. Such derivation graphs can be used to develop a taxonomy of security protocols. Another direction worth exploring is protocol synthesis. Once a set of generic components and composition, refinement and transformation operations are identified, it might be possible to automatically synthesize protocols that satisfy complex security specifications using the basic ideas of the derivation system. Finally, our initial results suggest that formal analysis of security protocols may be easier when proofs of correctness of complex protocols can be built from the proofs of their constituent sub-protocols. It should be an interesting challenge to extend the logical system to allow formal reasoning about all the protocol refinement and transformation steps that have been presented in this paper.

## Acknowledgments

Partially supported by NSF CCR-0121403, Computational Logic Tools for Research and Education, the Don University Research Initiative (URL) program administered by the Office of Naval Research (ONR) under Grant N00014-01-1-0795, NSF CCR-0209004 and CCR-0345397, and by ONR N00014-03-C-0237. This journal paper is a revised and extended version of two earlier conference papers [15,18].

We thank Iliano Cervesato, Nancy Durgin, Ralf Küsters, Catherine Meadows, and the anonymous reviewers for many helpful comments.

## Appendix A. Cord calculus

Cord calculus is the basic action structure [48,49,55] that we use to represent protocols. It was introduced in [23,24]. Here we provide a brief summary. Cord calculus was inspired by the strand space formalism [62], 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 [7]. 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 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 $(\nu x)$. 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*.[2] 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 $(\nu x)$ all *binds* the free occurrences of $x$

[2] This is slightly more general than the original strands from [62], because the term calculus, underlying actions, contains variables and substitution.

that appear on the right.[3] As usually, the bound variables are taken up to renaming, i.e., $\alpha$-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 9 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 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 10 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 (12) 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 (13) represents basic pattern matching, where the cord matches the term $p(t)$ with the expected pattern $p(x)$, and substitutes $t$ for $x$. Reaction (14) 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 $m \notin FV(S) \cup FV(S') \cup FV(C)$

[3] The tradition of denoting the operators binding $x$ by the round brackets around it goes back to Milner [49,48].

Table 9
Syntax of terms, actions and strands

| | | | |
|---|---|---|---|
| (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) | $a$ | $::= \epsilon$ | the null action |
| | | $\langle t \rangle$ | send a term $t$ |
| | | $(x)$ | receive into variable $x$ |
| | | $(\nu x)$ | generate new term $x$ |
| | | $(t/t)$ | match a term to a pattern |
| (strands) | $S$ | $::= aS \mid a$ | |

on the last rule means that the value $m$, created by $(\nu x)$, 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., [4]) 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

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

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

Table 10
Basic reaction steps

$$[S(x)S'] \otimes [T\langle t\rangle T'] \otimes C \rhd\rhd [SS'(t/x)] \otimes [TT'] \otimes C \quad (12)$$

$$[S(p(t)/p(x))S'] \otimes C \rhd\rhd [SS'(t/x)] \otimes C \quad (13)$$

$$[S(\nu x)S'] \otimes C \rhd\rhd [SS'(m/x)] \otimes C \quad (14)$$

Where the following conditions must be satisfied:

(12) $FV(t) = \emptyset$

(13) $FV(t) = \emptyset$

(14) $m \notin FV(S) \cup FV(S') \cup FV(C)$

input interface, then we call this process a *closed cord*. Intuitively, the input variables $y_0, \ldots y_{m-1}$ represent the *entry ports* and the output terms $t_0, \ldots t_{n-1}$ 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 $y_0, \ldots y_{m-1}$ and $\alpha$-equivalents define the same process. In other words, renaming $y_0, \ldots y_{m-1}$ throughout $S$ and $t_0 \ldots t_{n-1}$ yields another, equivalent representative of the same process.

The morphisms of the cord category $\mathcal{C}$ are processes given by $\alpha$-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 $(y_0 \ldots y_{m-1})$, modulo renaming. Ignoring the types of variables as before, an arity thus boils down to a number, in this case $m = \{0, 1, \ldots m-1\}$. 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

$$r = (x_0 \ldots x_{\ell-1})R\langle u_0 \ldots u_{m-1}\rangle \ : \ A^\ell \longrightarrow A^m$$

$$s = (y_0 \ldots y_{m-1})S\langle t_0 \ldots t_{n-1}\rangle \ : \ A^m \longrightarrow A^n$$

their sequential composition is defined

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

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

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

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

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

is the identity.

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

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

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

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

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 [55]. Intuitively, the universal property of $\mathcal{C}$ can perhaps be understood by noticing, first of all, that a process in the form $(y_0, \dots y_{m-1})[]\langle v_0, \dots v_{n-1}\rangle : m \longrightarrow n$, where $\{v_0, \dots v_{n-1}\} \subseteq \{y_0, \dots y_{m-1}\}$ 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 [38], 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 [53] and a model presented by Dolev and Yao [22].

*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 message being received and than eventually send. We will require that all send and receive actions by principals and the intruder are performed via buffer cords and assume that in every protocol there are enough instances of the buffer cord to guarantee delivery of every message. Buffer cords are a part of the infrastructure rather than a part of the protocol, we assume that they are executed by special nameless agents. Unless otherwise specified, when we refer to a thread, we mean a non-buffer thread, similarly, when we refer to an action, we mean an action performed by a non-buffer thread.

*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:

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

In words, $\mathit{EVENT}(R, X, P, \vec{n}, \vec{x})$ 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 $\mathit{LAST}(R, X, P, \vec{n}, \vec{x})$ to denote that the last event of run $R$ is $\mathit{EVENT}(R, X, P, \vec{n}, \vec{x})$.

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 $\sigma P$, where $\sigma$ is a substitution of values for variables. If $P$ matches $R|_X$ using substitution $\sigma$, then $\sigma$ 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.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 $\rho_i$. Then every symbol in the term $m$ is either generated in $\rho_i$, received in $\rho_i$, or was in the static interface of $\rho_i$.*

**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.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.** *For every receive action there is a corresponding send action. More formally, $\mathit{EVENT}(R, X, (x), m, x) \supset \exists Y.\mathit{EVENT}(R, Y, \langle m \rangle, \emptyset, \emptyset)$.*

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

**Lemma A.4.** *For any initial configuration* **C** *of protocol* $\mathcal{Q}$*, and any run* $R$*, if agent* $\hat{X} \in$ *HONEST*(**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 $\phi$ 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 $\phi$ has free variables, then $Q, R \models \phi$ if we have $Q, R \models \sigma\phi$ for all substitutions $\sigma$ that eliminate all the free variables in $\phi$. For a set of formulas $\Gamma$, 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 $\mathsf{Has}_i$ is inductively as follows:
  ($\mathsf{Has}_0(A, m)$ if $((m \in FV(R|_A))$
  $\vee$ *EVENT*$(R, A, (\nu x), m, x)$
  $\vee$ *EVENT*$(R, A, (x), m, x)$
  and $\mathsf{Has}_{i+1}(A, m)$ if $\mathsf{Has}_i(A, m) \vee$ ($\mathsf{Has}_i(A, m')$

$\vee$ ($\mathsf{Has}_i(A, m') \wedge \mathsf{Has}_i(A, m'')$
$\wedge ((m = m', m'') \vee (m = m'', m')))$
$\vee(\mathsf{Has}_i(A, m') \wedge \mathsf{Has}_i(A, K)$
$\wedge\, m = \{|m'|\}_K)$
$\vee(\mathsf{Has}_i(A, a) \wedge \mathsf{Has}_i(A, g^b)$
$\wedge\, m = g^{ab})$
$\vee(\mathsf{Has}_i(A, g^{ab}) \wedge m = g^{ba})$

Intuitively, $\mathsf{Has}_0$ 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. $\mathsf{Has}_{i+1}$ 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 $\hat{A} \in \mathit{HONEST}(\mathbf{C})$ 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 $\phi$.
- $\mathcal{Q}, R \models \diamondminus\phi$ 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 $\phi$ 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 $\phi$ 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 $\sigma$ 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 4.1. Since the logic and the proof system are an extension of the earlier work [23,24], here we only give brief and informal proofs for those axioms that are same or similar to axioms in [24]. We omit proofs for standard axioms and rules of temporal logic. Also, we show that the composition methodology is sound by proving Lemmas 4.3 and 4.7.

*C.1. Axioms for protocol actions*

**AA1**

$\phi[a]_X \diamondsuit\mathsf{a}$

Informally, this axiom says that if $a$ is an action, and $\mathsf{a}$ a corresponding action formula, when thread $X$ executes $a$, in the resulting state $\diamondsuit\mathsf{a}$ 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 $\sigma$ and $\mathcal{Q}, R_0 \models \sigma\phi$, we need to prove that $\mathcal{Q}, R_0R_1 \models \diamondsuit\sigma\mathsf{a}$. Since $R_1|_X$ matches $a$ under substitution $\sigma$, $R_1$ has to contain action $\sigma\mathsf{a}$, and therefore, by the semantics of the temporal operator "$\diamondsuit$", it has to be that $\mathcal{Q}, R_0R_1 \models \diamondsuit\sigma\mathsf{a}$. Now, by the definition of modal formulas we have $Q \models \phi\,[a]_X\diamondsuit\mathsf{a}$.

**AA2**

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

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 $\sigma$ and $\mathcal{Q}, R_0 \models \sigma\mathsf{Fresh}(X,t)$, we need to prove that $\mathcal{Q}, R_0R_1 \models \sigma\diamondsuit(\mathsf{a} \wedge \ominus\mathsf{Fresh}(X,t))$. Since $R_1|_X$ matches $a$ under substitution $\sigma$, $R_1$ has to contain action $\sigma\mathsf{a}$. Let $R_1'$ be a prefix of $R_1$ containing all actions preceding action $\sigma a$. 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 "$\diamondsuit$", it has to be that $\mathcal{Q}, R_0R_1 \models \sigma\diamondsuit(\mathsf{a} \wedge \ominus\mathsf{Fresh}(X,t))$.

**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 [24].

**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**

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

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 $\sigma$ and $\mathcal{Q}, R_0 \models \sigma\diamondsuit\,\mathsf{Receive}(X,p(x))$, we need to

prove that $\mathcal{Q}, R_0 R_1 \models \sigma \diamondminus \mathsf{Receive}(X, p(t))$. Since $R_1|_X$ matches $(q(x)/q(t))$ under substitution $\sigma$, and events of $R_1$ only contain ground terms, it has to be that $\sigma x$ is same as $\sigma t$, and therefore $\mathcal{Q}, R_0 \models \diamondminus \mathsf{Receive}(X, p(t))$. Clearly, formulas of the form $\diamondminus \mathsf{a}$ remain valid as new actions are executed, hence $\mathcal{Q}, R_0 R_1 \models \sigma \diamondminus \mathsf{Receive}(X, p(t))$.

### *C.2. Possession axioms*

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

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

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

**DEC**
$\mathsf{Has}(X, \{|x|\}_K) \wedge \mathsf{Has}(X, K) \supset \mathsf{Has}(X, x)$
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 **ENC**, proofs for other axioms are similar.

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

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

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

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

Informally, **SEC** says that if an agent $\hat{X}$ is honest, and some thread $Y$ executed by principal $\hat{Y}$ has decrypted a message $\{|n|\}_X$ (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 [24].

**VER**

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

Informally, **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 $\hat{X} \in$ *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 $\hat{X} \in$ *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 *EVENT*($R, Y, (x), m', x$) and $\{|n|\}_{\overline{X}} \subseteq m'$. By Lemma A.3 there is a corresponding send action for every receive, hence there exists a thread $Z$ such that *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 *EVENT*($R', Z, \langle m \rangle, \emptyset, \emptyset$). By Lemma A.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.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**
$\diamondsuit\mathsf{Fresh}(X, n) \wedge \diamondsuit\mathsf{Fresh}(Y, n) \supset (X = Y)$
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** $\mathsf{Persist} \in \{\diamondsuit\phi, \mathsf{Has}\}$
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** $n \not\sqsubseteq_v a$ **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.1 and A.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**

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

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**

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

Informally, this axiom says that after an agent does some actions $\mathsf{a_1},\ldots,\mathsf{a_n}$ in that order, it is true that $\mathsf{After}(\mathsf{a_i},\mathsf{a_{i+1}})$ for all $i = 1,\ldots,n-1$. 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 $\mathsf{a}$ 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.1 and A.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**

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

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, 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 $\phi$ is true after any run, then $\phi$ 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 $\sigma$, 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 $\sigma$ and $R_1''|_A$ matches $P'$ under $\sigma$. 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

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

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.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 4.3*

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 $\mathcal{Q}_1 \otimes \mathcal{Q}_2$ 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, $\vdash_{\mathcal{Q}_1 \otimes \mathcal{Q}_2} \mathsf{Honest}(\hat{X}) \supset \phi$.

*Proof of Lemma 4.7*

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 **S1**. Therefore, by the application of the honesty rule, $\vdash_{\mathcal{Q}} \mathsf{Honest}(\hat{X}) \supset \phi$.

## References


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

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

[3] 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.

[4] M. Barr and C. Wells, *Category Theory for Computing Science*. New York.

[5] 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.

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

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

[8] 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) (1993).

[9] M. Burrows, M. Abadi and R. Needham, A logic of authentication, *ACM Transactions on Computer Systems*.

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

[11] 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/.

[12] 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*, Springer-Verlag, 2003, pp. 339–355.

[13] J.A. Clark and J.L. Jacob, A survey of authentication protocol literature, Web Draft Version 1.0 available from www.cs.york.ac.uk/~jac/, 1997.

[14] 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*, IEEE, 2000, pp. 82–95.

[15] 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*, IEEE, 2003, pp. 109–125.

[16] 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*, 2003, pp. 11–23.

[17] 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*, IEEE, 2004, pp. 30–45.

[18] 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.

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

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

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

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

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

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

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

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

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

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

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

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

[31] R. Kemmerer, C. Meadows and J. Millen, Three systems for cryptographic protocol analysis, *J. Cryptology* **7**(2) (1994), 79–130.

[32] H. Krawczyk, The IKE-SIGMA protocol, Internet draft, 2002.

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

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

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

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

[37] H. Mantel, On the composition of secure systems, in: *Proceedings of the IEEE Symposium on Security and Privacy*, Oakland, CA, USA, IEEE Computer Society, 2002, pp. 88–101.

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

[39] D. McCullough, Noninterference and the composability of security properties, in: *Proceedings of the IEEE Symposium on Security and Privacy*, Oakland, CA, USA, IEEE Computer Society, 1988, pp. 177–186.

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

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

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

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

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

[45] 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.

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

[47] 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.

[48] R. Milner, Action calculi and the pi-calculus, in: *NATO Summer School on Logic and Computation*, Marktoberdorf, 1993.

[49] R. Milner, Action calculi, or syntactic action structures, in: *Proceedings of MFCS'93*, A.M. Borzyszkowski and S. Sokolowski, eds, Volume 711 of *Lecture Notes in Computer Science*, Springer-Verlag, 1993, pp. 105–121.

[50] R. Milner, *Communicating and Mobile Systems: The $\pi$-Calculus*, Cambridge University Press, Cambridge, UK, 1999.

[51] J.C. Mitchell, M. Mitchell and U. Stern, Automated analysis of cryptographic protocols using Mur$\varphi$, in: *Proc. IEEE Symp. Security and Privacy*, 1997, pp. 141–151.

[52] J.C. Mitchell, A. Ramanathan, A. Scedrov and V. Teague, A probabilistic polynomial-time calculus for the analysis of cryptographic protocols (preliminary report), in: *17th Annual Conference on the Mathematical Foundations of Programming Semantics*, Arhus, Denmark, 2001, S. Brookes and M. Mislove, eds, Volume 45, Electronic notes in Theoretical Computer Science, 2001.

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

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

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

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

[57] 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.

[58] 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*, Washington, 2001, pp. 184–200.

[59] 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*, 2004.

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

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

[62] 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*, Oakland, CA, IEEE Computer Society Press, 1998, pp. 160–171.

[63] 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.

[64] 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.