A Derivation System for Security Protocols and its Logical Formalization

Anupam Datta Ante Derek John C. Mitchell Dusko Pavlovic

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

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

Abstract

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 introduce a basic framework for deriving security protocols from such simple components. 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 using a small set of composition, refinement and protocol transformations. As a step toward associating logical derivations with protocol derivations, we extend a previous security protocol logic with preconditions and temporal assertions. The logic is proved to be sound. The composition operation of the derivation framework is formalized in the logic and a general methodology for proving protocol composition results is presented. Using the logic, the security properties of the standard signature based Challenge-Response protocol and the Diffie-Hellman key exchange protocol are formally proved. The ISO-9798-3 protocol is then proved correct by composing the correctness proofs of these two simple protocols.

1 Introduction

While many historical authentication and secrecy protocols, such as those cataloged by Clark and Jacob [11], may be analyzed independently, modern protocols often have a number of different subprotocols and interrelated modes. The Internet Key Exchange (IKE) protocol [22], 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.

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. However, there is no comprehensive theory about how each of these protocol parts work, and how properties of a compound protocol can be derived from properties of its parts. 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 [15] 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.

The protocol derivation system is intended to formalize the well established practice of presenting protocols incrementally, starting from simple components, and extending them by features and functions. Examples of this intuitive and appealing idea appear in [14]. Efforts to formalize the practice, and produce modular, reusable derivations of families of protocols, go back to [7]. More recently, Bellare, Canetti and Krawczyk [4] have studied two interesting protocol transformations, which they call *authenticators*, which generically add authentication to a given protocol scheme. We notice that, in some cases, the composition of two protocols has the security properties of both. For example, Diffie-Hellman key exchange provides a shared secret, but no authentication. Conversely, signed challenge-response provides authentication, but no shared secret. Our composition of the two provides the advantages of both: an authenticated shared secret. In exploring this approach, we aim to capture the intuition that protocol designers use when they develop new protocols by modifying existing protocols for the purpose of adding new desired properties. Moreover, when new protocols are presented as derivations, their analysis could be easier.

Our eventual goal is to develop a logic in which protocol correctness proofs will follow the derivation steps. Each protocol component will be assigned a logical formula, expressing its meaning, and formally tied to it by a semantical relation in the style of [17]. Corresponding to each composition, refinement and transformation operation there will be a set of inference rules in the logic. Having verified soundness and correctness of such assignments once and for all, one could then routinely infer standard properties of protocols from the way they are derived. As a step in this direction, we extend a previous security protocol logic with preconditions and temporal assertions and prove properties of protocols that lie in the early branches of the family tree. The logic is proved to be sound. Thus, although the assertions we prove only mention steps of the protocol, each provable assertion about an action or sequence of actions holds in any run of the protocol that contains the given actions and arbitrary additional actions by any number of additional principals and malicious attacker(s).

The protocol composition operation of the derivation system is formalized within the proof system using three inference rules. Conceptually, this involves addressing two basic problems in modular security. The first may be called *additive combination* – we wish to combine protocol components in a way that accumulates security properties. 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 significants of nondestructive combination is the construction in [25] 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 ϕ and ψ are formulas asserting secrecy of some data, past actions of other principals, or other facts about a run of a protocol. The triple $\phi[P]_A\psi$ means that if ϕ is true before principal A does actions P, then ψ will be true afterwards. For example, the precondition might assert that A knows B's public key, the actions P allow A to receive a signed message and verify B's signature, and the postcondition may say that B sent the signed message that A received. The importance of before-after assertions is that we can combine assertions about individual protocol steps to derive properties of a sequence of steps: if $\phi[P]_A\psi$ and $\psi[P']_A\theta$, then $\phi[PP']_A\theta$. For example, an assertion assuming that keys have been successfully distributed can be combined with steps that do key distribution to prove properties of a protocol that distributes keys and uses them.

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

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

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 compositional logical framework using which security properties of protocols can be proved and also protocols can be formally derived using a general composition operation. To illustrate the use of this method, 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. In Section 5, we survey previous work on protocol derivation and composition. Finally, in Section 6, we present 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 are of three types: *composition*, *transformation* and *refinement*. Intuitively, the distinctions between these parts are as follows:

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 takes two protocols and puts them together in some way. Sequential composition and 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. It is a general rewrite operation that can 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 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 [14], standard Challenge-Response protocol [39], JFKi, JFKr, ISO-9798-3 protocol [2], and the core of the IKE protocol [22].

3 Derivation of the STS Family

In this section, we present a derivation of the STS protocol family. The STS family 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. 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 transformation and refinement operations used in deriving the STS family of key exchange protocols. The components and transformations are presented tersely, with additional intuition and explanation given where they are used.

In informally describing the derivation system, we use a standard informal notation for protocol steps. However, the reader should bear in mind that protocols actually involve initial conditions, communication steps, and internal actions used to compute data used in messages and data produced as output of the protocol for use in other operations. When we derive a protocol, the derivation steps may act on one or more of these aspects of a protocol.

3.1 Components

Diffie-Hellman component, C_1

The basic Diffie-Hellman protocol [15] 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. The security of the key depends on the computational hardness of the discrete logarithm problem. Our component C_1 contains only the internal computation steps of the Diffie-Hellman protocol.

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 [39])

$$\begin{aligned} &I \to R : m \\ &R \to I : SIG_R(m) \end{aligned}$$

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 Transformations

Message component move, T_1

This transformation moves a 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 Figure 1.

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

Figure 1. An example of a binding transformation

We can understand the value of this transformation by considering the signature-based authenticator, C_2, described above. Protocol C_2 provides one-sided authentication: after executing the protocol, I is assured that the second message was generated by R in response to the first message. However, R does not know the identity of I. Since the goal of a mutual authentication protocol is to provide the authentication guarantee to both parties, it seems likely that we can construct a mutual authentication protocol from two instances (executed in opposite directions) of C_2. However, the sequential composition of two runs of C_2 does not quite do the job, since neither party can be sure that the other participated in one of the runs. If we apply a transformation to obtain the protocol on the left side of Figure 1, and then apply the binding transformation to obtain the one on the right, we obtain a protocol with both nonces inside the signature, thus assuring that m and n belong to the same session.

We note, however, that the protocol on the right side of Figure 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 3.4.

Cookie, T_3

The purpose of the cookie transformation is to make a protocol resistant to blind Denial-of-Service (DoS) attacks. Under certain assumptions, it guarantees that the responder does not have to create state or perform expensive computation before a round-trip communication is established with the initiator. The cookie transformation is described in detail in [13]. Here, we only touch on the main idea.

An example of a cookie transformation is shown in Figure 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 unforge-

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

Figure 2. An example of a cookie transformation

able 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.

3.3 Refinements

While defining refinements, we use the notation $a \Rightarrow b$ to indicate that every instance of message component a in the protocol should be replaced by b. For ease of exposition, different refinements that serve the same purpose are grouped together below.

Refinement R_1 $\quad SIG_X(m) \Rightarrow E_K(SIG_X(m))$, where K is a shared key 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 was not encrypted.

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

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

Thus, two general methods of protocol derivation using refinements emerge: (i) construct a protocol with some properties and apply a refinement to add another property to the protocol; (ii) construct a protocol assuming some facts and discharge those assumptions by applying a refinement.

The next four refinements are all geared towards adding the property of mutual authentication to a protocol. We describe one of the refinements in detail here, deferring details of the others to the point where they are used in the derivation of the STS family in the next section.

Refinement R_4 $\quad SIG_X(m) \Rightarrow SIG_X(m, ID_Y)$, where Y is the peer. An assumption here is 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 [43]), 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 [27].

Refinement R_2 $\quad SIG_X(m) \Rightarrow SIG_X(HMAC_K(m, ID_X))$, where K is a shared key with the peer. The difference between this refinement and R_4 is that instead of signing the peer's identity, the principal signs a keyed hash of the message and her own identity. This refinement is used in the derivation of IKE.

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

Refinement R_7 $\quad E_K(m) \Rightarrow E_K(m), HMAC_{K'}(role, E_K(m))$, where K and K' are keys shared with the peer. In this approach, each party includes a keyed hash of the encrypted signature and it's own role (i.e., initiator or responder). This refinement is used in the derivation of JFKr.

3.4 The Derivation

We now use the components and operations of the derivation system defined above to systematically derive the protocols in the STS family. The complete derivation graph is shown in Figure 3. In what follows, we trace the derivations of the various protocols in the graph. At each derivation step, we attempt to intuitively explain what property that step helps achieve.

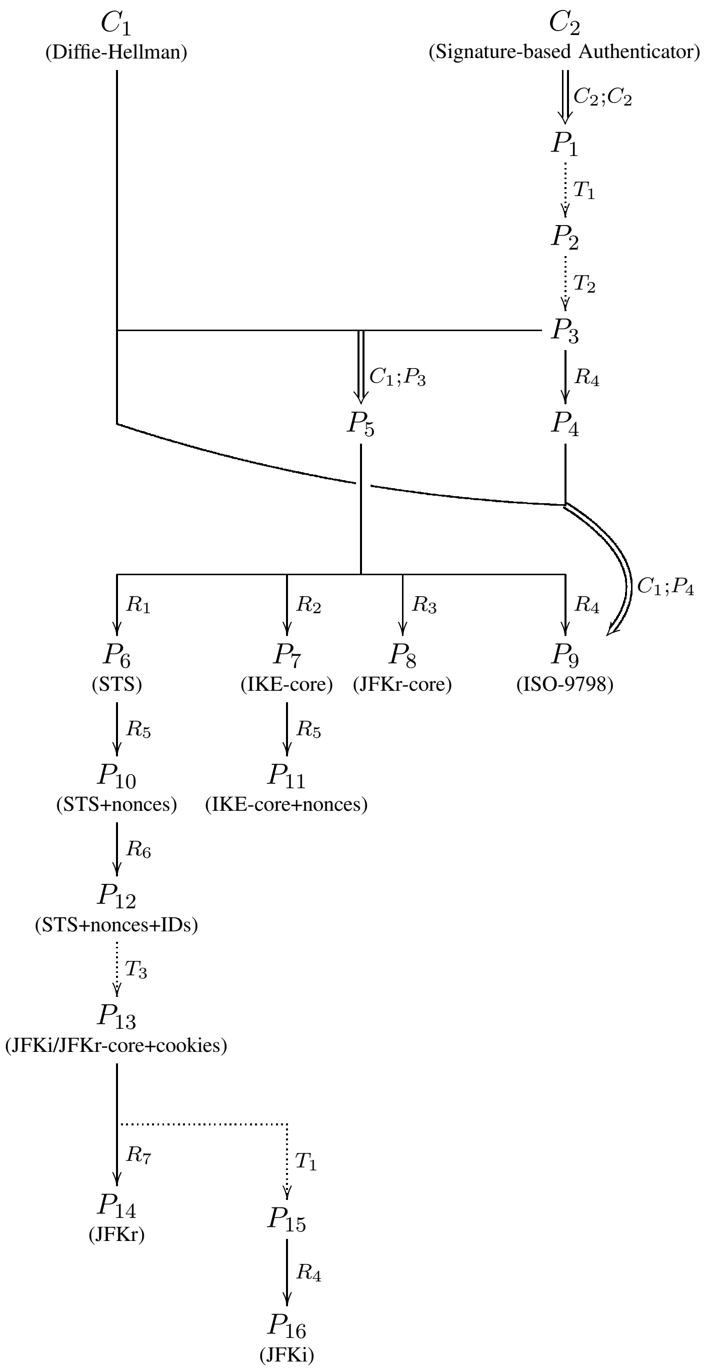

Figure 3. Derivation graph of the STS protocol family

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

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

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

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

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

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.

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

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

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

Protocol P_5 Obtained by composing component C_1 with protocol P_3.

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

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

After protocol P_5, four different derivation paths can be seen in Figure 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 [26] 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.

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

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

Protocol P_{10} Obtained by applying refinement R_5 to protocol P_6.

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

P_{10} 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_{12} Obtained by applying refinement R_6 to protocol P_{10}.

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

By applying refinement R_6 to P_{10}, no new properties are introduced. Instead, the assumption that the protocol principals possessed each other's public key certificates apriori is discharged by explicitly exchanging certificates alongside the signatures.

Protocol P_{13} Obtained by applying the cookie transformation, T_3, to protocol P_{12}.

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

The cookie transformation ensures that in addition to the properties of protocol P_{12}, 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_{13} and only differ in the form of identity protection that they offer.

Path 1.1: JFKr

Protocol P_{14} Obtained by applying refinement R_7 to P_{13}. This is essentially JFKr. We ignore some of the message fields (e.g., the security association and the group identifying information) which can be added using two more refinements.

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

P_{14} retains all the properties of P_{13}. 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. Also, Lowe's attack [28] doesn't work against this protocol.

Path 1.2: JFKi

Protocol P_{15} Obtained by applying transformation T_1 to protocol P_{13}.

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

The message component ID_R is moved from message 4 in P_{13} 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_{13} except for the fact that the form of identity protection is different. Unlike P_{13}, here the responder's identity is not protected. The initiator's identity is still protected against active attackers.

Protocol P_{16} Obtained by applying refinement R_4 to protocol P_{15}. This is JFKi (except for one additional signature in the second message which can be added using one more transformation). As with JFKr, some of the message fields which do not contribute to the core security property are ignored.

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

The refinement added the peer's identities inside the signatures. ID_R and ID_I are added inside I's and R's signatures in message 3 and message 4 respectively. Including the identities inside the signatures obviates the attack described while presenting protocol P_3 and Lowe's attack on STS [28]. P_{16} retains all the properties of P_{15}.

Path 2: IKE

We now consider the second path starting from protocol P_5. This path includes two protocols closely related to IKE [22].

Protocol P_7 Obtained by applying refinement R_2 to protocol P_5. This protocol has been described as the core for IKE in [2].

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

Instead of just signing the Diffie-Hellman exponentials, each principal now signs a keyed hash of the exponentials and their own identities. Since the key used is derived from the

Diffie-Hellman secret, g^{ir}, which is known only to I and R, an adversary cannot launch the person-in-the-middle attack described while presenting P_3 and to which P_5 is also susceptible. This protocol therefore provides both mutual authentication and a shared secret between I and R.

Protocol P_{11} Obtained by applying refinement R_5 to protocol P_7.

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

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

Path 3: JFKr/SIGMA-core

The third path starting from protocol P_5 consists of a protocol that has been described as the core for JFKr and IKE-SIGMA in [2].

Protocol P_8 Obtained by applying refinement R_3 to protocol P_5.

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

This protocol is very similar to protocol P_7 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_9 Obtained by applying refinement R_4 to protocol P_5. This is the ISO-9798-3 protocol.

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

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

Alternative Derivation of the ISO-9798-3 Protocol

Now we present a derivation of protocol P_9, *ISO-9798-3* .

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

$$
\begin{array}{l}
I \rightarrow R : m \\
R \rightarrow I : n, SIG_R(n, m, ID_I) \\
I \rightarrow R : SIG_I(m, n, ID_R)
\end{array}
$$

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_9 is now derived by composing component C_1 with protocol P_4 in exactly the same way that P_5 was derived.

3.5 Other Issues

3.5.1 Commutativity of Rules

As suggested by protocol P_9 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.

3.5.2 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

The protocol derivations presented in the previous section show how security properties accumulate as complex protocols are constructed from simple components through refinement, composition and transformation operations.

In this section, we present a formal system for describing security protocols and for reasoning about their properties. The *ISO-9798-3* protocol is then formally derived in this system from the signature-based challenge-response protocol and the standard Diffie-Hellman key exchange protocol. The derivation involves: (a) construction of the *ISO-9798-3* protocol by composition of the Diffie-Hellman and challenge-response protocols; and (b) proof of correctness of the composed protocol from proofs of correctness of its components. Note that this corresponds to the step in the derivation tree for the STS family where C_1 and P_4 are composed to yield P_9. It therefore provides some evidence that a complete formalization of the derivation system presented in the previous section may be achievable.

The formal logic we use is based on earlier work [17]. It consists of two parts: a language called cord calculus for representing protocols, and a logic for reasoning about properties of protocols. In order to derive protocols, we need a formal language to describe them. Cord calculus is our language of choice for this purpose. While we use cord calculus in its original form, the protocol logic has been extended in order to allow reasoning about a broader range of security protocols. The most significant extension is a rigorous treatment of *protocol composition*. The composition rules play a key role in the derivation of the *ISO-9798-3* protocol. Other extensions include a set of rules which allow reasoning about the *temporal ordering of actions* carried out by the different principals during the execution of a protocol. This form of reasoning is required in order to formalize the notion of authentication.

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. Note that the soundness proofs of the axioms and rules of the logic are sketched in Appendix C.

4.1 Cord Calculus

Cords [17] are the formalism we use to represent protocols and their parts. They form an action calculus [40, 41, 44], based on π-calculus [42], and related to spi-calculus [1]. The cords formalism is also similar to the approach of the Chemical Abstract Machine formalism [6], in that the communication actions can be viewed as reactions between “molecules”. Cord calculus serves as a simple “protocol programming language” which supports our Floyd-Hoare style logical annotations, and verifications in an axiomatic semantics. Cord calculus is presented in [17]. In order to make this paper self-contained, a brief summary is included in Appendix A.

In this section, we illustrate with an example how protocols are represented in cord calculus. Figure 4 shows the *ISO-9798-3* protocol [24] in the informal arrows-and-messages notation which is commonly used to describe security protocols. The same protocol is written out in the language of cord calculus in Figure 5. The common point between the two is that a protocol is described by listing out the sequence of actions that honest parties would execute in a run. In this example, the protocol consists of two roles, the initiator role and the responder role. The sequence of actions in the initiator role are given by the cord $\mathbf{A}$ in Figure 5. In words, the actions are: generate a fresh random number; send a message with

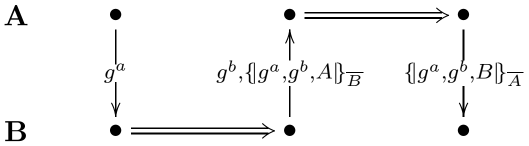

Figure 4. *ISO-9798-3* **as arrows-and-messages**

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

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

Figure 5. *ISO-9798-3* **as a cord space**

the Diffie-Hellman exponential of that number to the peer, $\hat{B}$; receive a message with source address $\hat{B}$; verify that the message contains $\hat{B}$'s signature over data in the expected format; and finally, send another message to $\hat{B}$ with the initiator's signature over the Diffie-Hellman exponential that she sent in the first message, the data she received from $\hat{B}$ (which should be a Diffie-Hellman exponential generated by $\hat{B}$) and $\hat{B}$'s identity. The notations (νx), $\langle t \rangle$, (x) refer respectively to the actions of nonce generation, sending a term and receiving a message. Here, a message is assumed to be of the form: (source, destination, content). Detailed syntax of cord calculus is presented in Appendix A.

4.2 A Protocol Logic

4.2.1 Syntax

The formulas of the logic are given by the grammar in Table 1, where ρ may be any role, written using the notation of cord calculus. Here, t and P denote a term and a process respectively. We use the word *process* to refer to a principal executing an instance of a role, e.g., Alice taking part in a session of a protocol in the initiator role. As a notational convention, we use X to refer to a process belonging to principal $\hat{X}$. We use ϕ and ψ to indicate predicate formulas, and m to indicate a generic term we call a "message". m is of the form (source, destination, protocol-identifier, content), i.e., each message has source and destination fields and a unique protocol identifier besides the contents. Note that the source field of a message may not be the same as the actual sender of the message since the intruder can spoof the source address. Also, the principal identified by the destination field may not receive the message since the intruder can intercept messages. The source and destination fields in the message are useful while proving authentication properties of protocols. When an honest principal sends out a message, the source field identifies her and the destination field identifies the intended recipient. Our formalization of authentication is based on the notion of matching records of runs [14] which requires that whenever $\hat{A}$ and

$\hat{B}$ accept each other's identities at the end of a run, their records of the run should match, i.e., each message that $\hat{A}$ sent was received by $\hat{B}$ and vice versa, each send event happened before the corresponding receive event, and moreover the messages sent by each principal ($\hat{A}$ or $\hat{B}$) appear in the same order in both the records. Including the source and destination fields in the message allows us to match up send-receive actions. Since in this paper, we reason about correctness of a protocol in an environment in which other protocols may be executing concurrently, it is important that when $\hat{A}$ and $\hat{B}$ accept each other's identities, they also agree on which protocol they have successfully completed with the other. One way to extend the matching histories characterization to capture this requirement is by adding protocol identifiers to messages. Now if $\hat{A}$ and $\hat{B}$ have matching histories at the end of a run, not only do they agree on the source, destination and content of each message, but also on which protocol this run is an instance of.

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

The formula $\mathsf{Has}(X, x)$ means that principal $\hat{X}$ possesses information x in the process X. This is "possesses" in the limited sense of having either generated the data or received it in the clear or received it under encryption where the decryption key is known. The formula $\mathsf{Send}(X, m)$ means that the last action in a run of the protocol corresponds to principal $\hat{X}$ sending message m in in the process X. $\mathsf{Receive}(X, m)$, $\mathsf{New}(X, t)$, $\mathsf{Decrypt}(X, t)$, and $\mathsf{Verify}(X, t)$ are similarly associated with the receive, new, decrypt and signature verification actions of a protocol. $\mathsf{Fresh}(X, t)$ means that the term t generated in X is "fresh" in the sense that no one else has seen any term containing t as a subterm. Typically, a fresh term will be a nonce and freshness will be used to reason about the temporal ordering of actions in runs of a protocol. This form of reasoning is useful in proving authentication properties of protocols. The formula $\mathsf{Honest}(\hat{X})$ means that the actions of principal $\hat{X}$ in the current run are precisely an interleaving of initial segments of traces of a set of roles of the protocol. In other words, $\hat{X}$ assumes some set of roles and does exactly the actions prescribed by them. $\mathsf{Contains}(t_1, t_2)$ means that t_2 is a subterm of t_1. This predicate helps us identify the components of a message. The two temporal operators $\diamondminus$ and $\ominus$ have the same meaning as in Linear Temporal Logic [30]. Since we view a run as a linear sequence of states, $\diamondminus\ \phi$ means that in some state in the past ϕ holds, whereas $\ominus\ \phi$ means that in the previous state ϕ holds. The predicate $\mathsf{After}(\mathsf{a}_1, \mathsf{a}_2)$ means that the action a_2 happened after the action a_1 in a run. In this paper, we restrict attention to protocol roles in which all actions are unique. In particular, a principal executing a role of a protocol does not send the same message twice. This seems like a reasonable assumption since even if she did send two messages with the same content, she would probably distinguish the two by using message sequence numbers or a similar mechanism. The technical benefit of this assumption is that After becomes a transitive relation for actions executed by honest principals.

We note here that the temporal operator $\diamondminus$ and some of the predicates (Send, Receive) bear semblance to those used in NPATRL [48], the temporal requirements language for the NRL Protocol Analyzer [36, 37]. However, while NPATRL is used for specifying protocol requirements, our logic is also used to infer properties of protocols. This leads to essential

Action formulas

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

Formulas

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

Modal forms

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

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

Table 1. Syntax of the logic

semantical differences.

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 ϕ holds for run R of protocol $\mathcal{Q}$." In this relation, R may be a complete run, with all sessions that are started in the run completed, or an incomplete run with some principals waiting for additional messages to complete one or more sessions. If $\mathcal{Q}$ is a protocol, then let $\bar{\mathcal{Q}}$ be the set of all initial configurations of protocol $\mathcal{Q}$, each including a possible intruder cord. Let $\mathsf{Runs}(\bar{\mathcal{Q}})$ be the set of all runs of protocol $\mathcal{Q}$ with intruder, each a sequence of reaction steps within a cord space. If ϕ has free variables, then $Q, R \models \phi$ if we have $Q, R \models \sigma\phi$ for all substitutions σ that eliminate all the free variables in ϕ. We write $\mathcal{Q} \models \phi$ if $\mathcal{Q}, R \models \phi$ for all $R \in \mathsf{Runs}(\bar{\mathcal{Q}})$.

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

4.3 Proof System

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 in a state in which formula ϕ

$$
\begin{array}{ll}
\mathbf{AA1} & \phi[a]_X \diamondminus (\mathsf{a} \wedge \ominus\phi) \\
\mathbf{AN2} & \phi[(\nu n)]_X \ \mathsf{Has}(Y, n) \supset (Y = X) \\
\mathbf{AN3} & \phi[(\nu n)]_X \ \mathsf{Fresh}(X, n)
\end{array}
$$

Table 2. Axioms for protocol actions

holds. Note that the a in axiom $\mathbf{AA1}$ is any one of the 5 actions and a is the corresponding predicate in the logic. $\mathbf{AA1}$ states that if a principal has executed an action in some role, then the corresponding predicate asserting that the action had occurred in the past is true. Also, in the previous state, ϕ is true. If process X generates a new value n and does no further actions, then $\mathbf{AN2}$ says that no one else knows n, and $\mathbf{AN3}$ says that n is fresh.

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. $\mathbf{ORIG}$ and $\mathbf{REC}$ state respectively that a principal possesses a term if she had freshly generated it (a nonce) or if she received it in some message. $\mathbf{TUP}$ and $\mathbf{ENC}$ enable construction of tuples and encrypted terms if the parts are known. $\mathbf{PROJ}$ and $\mathbf{DEC}$ allow decomposition of a tuple into its components and decryption of an encrypted term if the key is known. The next two axioms are aimed at capturing the *blackbox model of encryption and signature*. $\mathbf{VER}$ refers to the unforgeability of signatures while $\mathbf{SEC}$ stipulates the need to possess the private key in order to decrypt a message encrypted with the corresponding public key. The additional condition requiring principal $\hat{X}$ to be honest guarantees that the intruder is not in possession of the private keys. An important axiom is $\mathbf{N1}$ which states that if a process X has generated a value n, then that value is distinct from all other values generated in all other roles. $\mathbf{N2}$ states that freshly generated values within the same process are distinct from each other. $\mathbf{F1}$ states that fresh values generated in different processes are distinct. $\mathbf{N1}$, $\mathbf{N2}$, and $\mathbf{F1}$ together capture the intuition that fresh nonces and Diffie-Hellman exponentials are unique. Finally, $\mathbf{CON}$ states that the tuple x, y contains x and y as 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. $\mathbf{G2}$ is exactly of the same form as the rule of consequence in Hoare Logic. It is clear that most predicates are preserved by additional actions. For example, if in some state $\mathsf{Has}(X, n)$ holds, then it continues to hold, when X executes additional actions. Intuitively, if a process possesses some information at a point in

Possession Axioms:

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

Encryption and Signature:

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

Uniqueness of Nonces:

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

Subterm Relationship:

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

Table 3. Relationship between atomic predicates

Generic Rules:

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

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

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

Freshness Loss Axiom:

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

Table 4. Modal Axioms and Rules

a run, then she remembers it for the rest of the run. Note, however, that the Fresh predicate is not preserved if the freshly generated value n is sent out in a message (see $\mathbf{F}$).

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 processes. For this purpose, we use a fragment of the proof system for Propositional Linear Temporal Logic, PLTL (Table 5). See [45] for a complete axiomatization of PLTL. The axioms and rules specific to the temporal ordering of actions are presented in Table 5. The first two rules are fairly straightforward. $\mathbf{AF1}$ orders the actions within a role. This is consistent with the way we view a role as an ordered sequence of actions. $\mathbf{AF2}$ states that the After relation is transitive on actions executed by honest participants. It makes sense since we assume that in a role of a protocol, an honest principal does not send the same message twice. $\mathbf{AF3}$ uses the freshness of nonces to reason about the ordering of actions carried out by different processes. Intuitively, $\mathbf{AF3}$ states that if a process X creates a fresh value n and then sends out a message containing it as a subterm, then any action carried out by any other process which involves n (e.g. if Y receives a message containing n inside a signature), happens after the send action.

4.3.5 The Honesty Rule

The honesty rule is essentially an invariance rule for proving properties of all roles of a protocol. It is similar to the basic invariance rule of LTL [30]. The honesty rule is used to combine facts about one role with inferred actions of other roles. For example, suppose Alice receives a response from a message sent to Bob. Alice may wish to use properties of Bob's role to reason about how Bob generated his reply. In order to do so, Alice may

PLTL Axioms:

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

Temporal Generalization Rule:

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

Temporal Ordering of actions:

$$\begin{array}{ll}
\mathbf{AF1} & \theta[a_1 \ldots a_n]_X\ \mathsf{After}(\mathsf{a_1}, \mathsf{a_2}) \wedge \ldots \wedge \mathsf{After}(\mathsf{a_{n-1}}, \mathsf{a_n}) \\
\mathbf{AF2} & \mathsf{Honest}(\hat{X}) \wedge \mathsf{Honest}(\hat{Y}) \wedge \mathsf{Honest}(\hat{Z}) \supset \\
 & \quad (\mathsf{After}(\mathsf{a_1}(X), \mathsf{a_2}(Y)) \wedge \mathsf{After}(\mathsf{a_2}(\mathsf{Y}), \mathsf{a_3}(Z)) \supset \mathsf{After}(\mathsf{a_1}(X), \mathsf{a_3}(Z))) \\
\mathbf{AF3} & (\diamondminus\ (\mathsf{Send}(X, m) \wedge \ominus\mathsf{Fresh}(X, n)) \wedge \diamondminus\ \mathsf{a}(Y) \wedge \mathsf{Contains}(m, n) \wedge \mathsf{Contains}(a, n) \wedge (Y \neq X)) \\
 & \qquad\qquad \supset \mathsf{After}(\mathsf{Send}(X, m), \mathsf{a}(Y))
\end{array}$$

Table 5. Axioms and rules for temporal ordering

assume that Bob is honest and derive consequences from this assumption. Since honesty, by definition in our framework, means "following one or more roles of the protocol," honest principals must satisfy every property that is a provable invariant of the protocol roles.

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

Since the honesty rule depends on the protocol, we write $\mathcal{Q} \vdash \theta[P]\phi$ if $\theta[P]\phi$ is provable using the honesty rule for $\mathcal{Q}$ and the other axioms and proof rules. Using the notation just introduced, the honesty rule may be written as follows.

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

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

4.4 Formalizing Protocol Composition

The machinery developed till this point only allow protocols to be analyzed in isolation. In this section, we add inference rules to the deductive system to reason about protocol composition. These rules formalize within the logic the abstract notion of a composition operation introduced in the derivation system. We address the two basic problems that arise in modular design and analysis of security protocols: additive combination and non-destructive combination. Intuitively, if a complex protocol is built up from two simpler components, additive combination guarantees that it retains the security properties of both, assuming they do not degrade each other's security; whereas non-destructive combination ensures that the two components actually do not interact insecurely.

The protocol composition rules are collected in Table 6. Γ denotes a set of formulas which we refer to as *environment invariants*. The idea is to capture, using these formulas, the constraints that the environment must satisfy in order to enable a specific protocol to retain its security property. Typically, these constraints will impose restrictions on the actions of the honest principals, i.e., the principals who are faithfully executing one or more of the protocols running in the environment. We write $\Gamma \vdash \phi$ if ϕ is provable using the formulas in Γ and the axioms and proof rules of the deductive system. The semantic entailment, $\Gamma \models \phi$,

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

Table 6. Composition Rules

is defined in Appendix B. Essentially, it says, that in any run in which the invariants in Γ hold, the formula ϕ is true.

The weakening rule, $\mathbf{W}$, states that a formula ϕ which is provable from a set of hypotheses, Γ, remains provable if additional formulas are added to the set of hypotheses. The protocol composition rule $\mathbf{C1}$ gives us a way of sequentially composing two roles P and P' when the logical formula guaranteed by the execution of P, i.e., the post-condition of P, matches the pre-condition required in order to ensure that P' achieves some property. As before, Γ denotes a set of hypotheses which are used in proving the properties of the protocols. This form of reasoning allows a proof of correctness of a protocol to be built up incrementally from a proof of its component sub-protocols. The other composition rule $\mathbf{C2}$ states that if the environment invariants hold for two protocols, $\mathcal{Q}$ and $\mathcal{Q}'$, then the invariants also hold for their composition. This rule is sound if the formulas in Γ capture trace-based invariants, which are proved using the honesty rule in our deductive system. Soundness proofs of the rules in Table 6 are presented in Appendix C.

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

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

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

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

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

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

required only in the case of sequential composition.

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

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

4.5 An Example of Protocol Composition

In this section, we use our logical framework to formally derive properties of the *ISO-9798-3* protocol from properties of its parts, the signature-based challenge-response protocol (CR) and a protocol which forms the essence of the standard Diffie-Hellman key exchange protocol (DH_0). The formal logical derivation is in keeping with the main idea of the protocol derivation system. It corresponds to the step in Figure 3, where P_9 is derived from C_1 and P_4 by applying a composition operation (note that P_9 is actually *ISO-9798-3* and C_1 and P_4 are DH_0 and CR respectively.) We sketch below the outline of the formal derivation, relegating detailed proofs to Appendix D.

Example 4.1 *The ISO-9798-3 Protocol*

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

Challenge Response Protocol, CR: Our formulation of authentication is based on the concept of *matching conversations* [5] and is similar to the idea of proving authentication using *correspondence assertions* [50]. The same basic idea is also presented in [14] 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 10 in Appendix D. We also discuss the structure of the proof and identify a general method for proving authentication results in the logic. The final property proved about the initiator role (referred as ϕ_{auth} henceforth) is of the form: *precondition [actions] postcondition*, where:

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

Here, the predicate $\mathsf{ActionsInOrder}(\mathsf{a_1}, \mathsf{a_2}, \ldots, \mathsf{a_n})$ means that the actions $a_1, a_2, \ldots, a_n$ were executed in that order. It can be defined in a natural way using the transitivity axiom about the After predicate, $\mathbf{AF2}$. Intuitively, this formula means that after executing the actions in the initiator role purportedly with $\hat{B}$, $\hat{A}$ is guaranteed that her record of the run matches that of $\hat{B}$, provided that $\hat{B}$ is honest. The set of environmental invariants used in this proof, Γ, contains only one formula (line (9) of Table 10), i.e.,

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

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

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

Table 7 presents the rules specific to the way that Diffie-Hellman secrets are computed. The predicate Computes() is used as a shorthand to denote the fact that the only way to compute a Diffie-Hellman secret is to possess one exponent and the other exponential. $\mathbf{DH1}$ states that if X can compute the Diffie-Hellman secret, then she also possesses it.

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

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

Table 7. Diffie-Hellman Axioms

$\mathbf{DH2}$ captures the intuition that the only way to possess a Diffie-Hellman secret is to either compute it directly or obtain it from a received message containing it. $\mathbf{DH3}$ states that if a principal receives a message containing a Diffie-Hellman secret, someone who has computed the secret must have previously sent a (possibly different) message containing it. $\mathbf{DH4}$ captures the intuition that if a is fresh at some point of a run, then g^a is also fresh at that point.

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

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

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

Composing the Protocols: We now prove the security properties of the *ISO-9798-3* protocol by composing the correctness proofs of DH_0 and CR. In doing so, we follow the general methodology for proving composition results outlined in Section 4.4. Let us go back and look at the form of the logical formulas characterizing the initiator roles of DH_0 and CR:

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

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

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

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

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

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

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

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

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

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

Intuitively, the property proved is that if $\hat{B}$ is honest, then $\hat{A}$ and $\hat{B}$ are the only people who know the Diffie-Hellman secret g^{ab}. In other words, the *ISO-9798-3* protocol can be used to compute an authenticated shared secret. The complete proof is presented in Table 11 in Appendix D. It requires two additional invariants (lines (3), (10)) capturing the intuition that the honest agents sign Diffie-Hellman exponentials only and that they do not send out the computed secret in any message.

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 [7], Bird, Gopal 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 [14]. More recently, Bellare, Canetti and Krawczyk [4] have studied two interesting protocol transformations, which they call *authenticators*, which generically add authentication to a given protocol scheme. Within the formal methods community, there has been some research on this problem too. In [46], 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 [47]. Independently, Clark and Jacob developed a similar approach for protocol synthesis [12]. They use genetic algorithms to search the space of protocols expressible in BAN logic [8]. Other works on using formal logic for protocol design include [3, 9].

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 [20] or something very similar to it [23]. 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 [38], 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 [49]. 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 [19]. A somewhat different approach is used by Lynch [29] 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 [10], Canetti, Meadows and Syverson, revisit the protocol composition problem. They show how the interaction between a protocol and its environment can have a major effect on the security properties of the protocol. In particular, they demonstrate a number of attacks on published and widely used protocols that are not feasible against the protocol running in isolation but become feasible in some environments. The main question that this study leaves open is: how should the environment be constrained so that it does not subvert the security goals of a protocol? The authors put forward some rules of thumb that could be useful in answering this question. Of these, at least two can be justified using our formalization. The first of these states that the environment should not use keys or other secrets in unaltered form. Specifically, the protocol under consideration should not encrypt messages with a key used to encrypt messages by any protocol in its environment. The reason this makes sense is that if two protocols use a particular form of encrypted message as a test to authenticate a peer, then the attacker might be able to make a principal running the first protocol accept a message which actually originated in a run of the second protocol. In our formalism, the environmental invariant for the protocol under consideration would fail to hold in such an environment, and the composition proof would therefore not go through. We note that this principle has been followed in the design of real-world protocols like IKE [22]. Also, Guttman and Fábrega have proved a theoretical result to the same effect in their strand space model [21]. The other rule of thumb (also recommended by Kelsey, Schneier and Wagner in [25]), is the use of unique protocol identifiers to prevent a message intended for use in one protocol to be mistaken for use in another protocol. This rule can also be similarly justified.

It is well known that many natural security properties (e.g., noninterference) are not preserved either under composition or under refinement. This has been extensively explored using trace-based modeling techniques [31, 32, 33, 34, 35], using properties that are not first-order predicates over traces, but second-order predicates over sets of traces that may not have closure properties corresponding to composition and refinement. In contrast, our security properties are safety properties over sets of traces that satisfy safety invariants, thus avoiding negative results about composability.

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 logical derivations with protocol derivations, we have extended a previous protocol logic with preconditions and temporal assertions. The logic is proved to be sound. 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 Figure 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.

Acknowledgements This work was partially supported by NSF CCR-0121403, Computational Logic Tools for Research and Education, the DoD University Research Initiative (URI) program administered by the Office of Naval Research (ONR) under Grant N00014-01-1-0795, and ONR Grants N00014-00-C-0945 and N00014-01-C-0454.

References

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

[2] W. Aiello, S.M. Bellovin, M. Blaze, R. Canetti, J. Ioannidis, A.D. 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. 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.

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

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

[7] Ray Bird, Inder Gopal, Amir Herzberg, Phil Janson, Shay Kutten, Refik Molva, and Moti Yung. Systematic design of a family of attack resistant authentication protocols. *IEEE Journal on Selected Areas in Communications*, 1(5), June 1993.

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

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

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

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

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

[13] Anupam Datta, John C. Mitchell, and Dusko Pavlovic. Derivation of the JFK protocol. Technical report, Kestrel Institute, 2002.

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

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

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

[17] Nancy Durgin, John C. Mitchell, and Dusko Pavlovic. A compositional logic for protocol correctness. In *Proceedings of 14th IEEE Computer Security Foundations Workshop*, pages 241–255. IEEE, 2001.

[18] Nancy Durgin, John C. Mitchell, and Dusko Pavlovic. A compositional logic for proving security properties of protocols. *Journal of Computer Security*, 11:677–721, 2003.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

[38] 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.

[39] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1996.

[40] Robin 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.

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

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

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

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

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

[46] 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.

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

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

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

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

A Cord Calculus

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

A.1 Terms, Actions, Strands and Cords

The *terms* t are built starting from the variables x and the constants c. Moreover, the set of basic terms also contains the names N, which can be variables $\hat{X}$, $\hat{Y}$, $\hat{Z}$, or constants $\hat{A}$, $\hat{B}$, $\hat{C}$, and keys K which can be variables y and constants k. Upon these basic sets, the term language is then generated by some given constructors p, which always include tupling, the public key encryption $\{|t|\}_K$ of the term t by the key K, and the signature $\{|t|\}_{\overline{K}}$ over the term t with the private key $\bar{K}$. The language of *actions* is then built upon the terms by further constructors. They include sending a term $\langle t \rangle$, receiving into a variable (x), matching a term against a pattern $(t/q(x))$, and creating a new value (νx). A *strand* is a list of actions. The idea is that they should be the subsequent actions of a single role in a protocol. For example, the strand $[(\nu x)\langle x \rangle]$ represents a role in which a principal generates x and then sends out a message containing the freshly generated value. Since some actions of a role may be mutually independent, they can in principle be executed in any order. Different strands can thus be semantically equivalent. A *cord* is an equivalence class of behaviorally indistinguishable strands. We use the word *process* to refer to a principal executing an instance of a role. Table 8 summarizes the formal definition of cords. In addition to the sequence of actions, a cord has an *input interface* and an *output interface*. As the name suggests, the output interface represents the output of that cord. The input interface is used to provide initial data to a cord. These input parameters (called static parameters) can represent data known apriori (e.g. signing key) or data that becomes known by executing another cord via its output interface.

A.2 Cord Spaces and Runs

A *cord space* is a multiset of cords that may interact via communication. We use $\otimes$ for multiset union and [ ] for the empty multiset. The *runs* of a protocol arise as reaction sequences of cord spaces. The basic reactions within a cord space are shown in Table 9, with the required side conditions for each reaction shown below them. The substitution (t/x) is assumed to act on the strand left of it, *viz* S'. As usual, it is assumed that no free variable becomes bound after substitution, which can always be achieved by renaming the bound variables. Reaction (1) is a send and receive interaction, showing the simultaneous sending of term t by the first cord, with the receiving of t into variable x by the second cord. We call this an *external action* because it involves an interaction between two cords. The other reactions all take place within a single cord. We call these *internal actions*. Reaction (2) is a basic pattern match action, where the cord matches the pattern $p(t)$ with the expected pattern $p(x)$, and substitutes t for x. Reaction (3) is a decryption pattern match action, where the cord matches the pattern $\{|p(t)|\}_y$ with the decryption pattern $\{|p(x)|\}_{\overline{y}}$ and substitutes t for x. Reaction (4) is a signature verification pattern match action. Finally, reaction (5) shows the binding action where the cord creates a new value that doesn't appear elsewhere in the cordspace, and substitutes that value for x in the cord to the right. The intuitive motive for the condition $FV(t) = \emptyset$ should be clear: a term cannot be sent, or tested, until all of its free variables are instantiated.

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

Table 8. Syntax of terms, actions and strands

$$
\begin{aligned}
[S(x)S'] \otimes [T\langle t\rangle T'] \otimes C \quad &\rhd\!\rhd \quad [SS'(t/x)] \otimes [TT'] \otimes C && (1)\\
[S\,(p(t)/p(x))\,S'] \otimes C \quad &\rhd\!\rhd \quad [SS'(t/x)] \otimes C && (2)\\
[S\,(\{\!|p(t)|\!\}_y/\{\!|p(x)|\!\}_{\overline{y}})\,S'] \otimes C \quad &\rhd\!\rhd \quad [SS'(t/x)] \otimes C && (3)\\
[S\,(\{\!|p(t)|\!\}_{\overline{y}}/\{\!|p(t)|\!\}_y)\,S'] \otimes C \quad &\rhd\!\rhd \quad [SS'] \otimes C && (4)\\
[S(\nu x)S'] \otimes C \quad &\rhd\!\rhd \quad [SS'(m/x)] \otimes C && (5)
\end{aligned}
$$

Where the following conditions must be satisfied:

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

Table 9. Basic reaction steps

A.3 Protocols

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

The protocol intruder is capable of taking any of several possible actions, including receiving a message, decomposing it into parts, decrypting the parts if the key is known, remembering parts of messages, and generating and sending new messages. This is the standard "Dolev-Yao model", which appears to have developed from positions taken by Needham and Schroeder [43] and a model presented by Dolev and Yao [16]. A *run of a protocol* is a sequence of reaction steps from an *initial configuration*. An *initial configuration* is determined by a set of principals, a subset of which are designated as honest, a cord space constructed by assigning one or more roles to each honest principal, and an intruder cord that may use only the secret keys of dishonest principals. A particular initial configuration may give rise to many possible runs. Intuitively, a protocol has a property if in all runs of the protocol arising from all possible initial configurations, that property is preserved.

B Semantics of Protocol Logic

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

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

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

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

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

Action Formulas:

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

Other Formulas:

- $\mathcal{Q}, R \models \mathsf{Has}(A, m)$ if there exists an i such that $\mathsf{Has}_i(A, m)$ where Has_j is defined by induction on j as follows:
  $(\mathsf{Has}_0(A, m)$ if $((m \in FV(R|_A))$
  $\vee\, EVENT(R, A, (\nu x), m, x)$
  $\vee\, EVENT(R, A, (x), m, x)$
  and $\mathsf{Has}_{i+1}(A, m)$ if $\mathsf{Has}_i(A, m) \vee$ $(\mathsf{Has}_i(A, m')$
  $\wedge\, ((m' = \{\!|p(t)|\!\}_K \wedge m = t$

$\wedge\, EVENT(R, A, (m'/\{\!|p(y)|\!\}_{\overline{K}}), t, y))$
$\vee\, (m' = p(t) \wedge m = t$
$\wedge\, EVENT(R, A, (m'/p(y)), t, y))))$
$\vee (\mathsf{Has}_i(A, m') \wedge \mathsf{Has}_i(A, m'')$
$\wedge\, ((m = m', m'') \vee (m = m'', m')))$
$\vee (\mathsf{Has}_i(A, m') \wedge \mathsf{Has}_i(A, K)$
$\wedge\, m = \{\!|m'|\!\}_K)$
$\vee (\mathsf{Has}_i(A, a) \wedge \mathsf{Has}_i(A, g^b)$
$\wedge\, m = g^{ab})$
$\vee (\mathsf{Has}_i(A, g^{ab}) \wedge m = g^{ba})$

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

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

- $\mathcal{Q}, R \models \mathsf{Honest}(A)$ if $A \in HONEST(\mathbf{C})$ in initial configuration $\mathbf{C}$ for R.

- $\mathcal{Q}, R \models \mathsf{Contains}(t_1, t_2)$ if $t_2 \subseteq_v t_1$. t_2 is a visible subterm of t_1, $t_2 \subseteq_v t_1$, if $t_2 \subseteq t_1$ and it is not the case that all occurrences of t_2 in t_1 are as parameters of one-way functions. For example, $n \not\subseteq_v g(n)$. The only one-way function that we consider here is the Diffie-Hellman exponentiation function, $g(x)$.

- $\mathcal{Q}, R \models (\phi_1 \wedge \phi_2)$ if $\mathcal{Q}, R \models \phi_1$ and $\mathcal{Q}, R \models \phi_2$

- $\mathcal{Q}, R \models \neg\phi$ if $\mathcal{Q}, R \not\models \phi$

- $\mathcal{Q}, R \models \exists x.\phi$ if $\mathcal{Q}, R \models (d/x)\phi$, for some d, where $(d/x)\phi$ denotes the formula obtained by substituting d for x in ϕ.

- $\mathcal{Q}, R \models \diamondminus\phi$ if $\mathcal{Q}, R' \models \phi$, where R' is a (not necessarily proper) prefix of R. Intuitively, this formula means that in some state in the past, formula ϕ is true.

- $\mathcal{Q}, R \models \ominus\phi$ if $\mathcal{Q}, R' \models \phi$, where $R = R'e$, for some event e. Intuitively, this formula means that $\ominus\phi$ is true in a state if ϕ is true in the previous state.

Modal Formulas:

- $\mathcal{Q}, R \models \phi_1\,[P]_A\,\phi_2$ if $R = R_0R_1R_2$, for some R_0, R_1 and R_2, and either P does not match $R_1|_A$ or P matches $R_1|_A$ and $\mathcal{Q}, R_0 \models \sigma\phi_1$ implies $\mathcal{Q}, R_0R_1 \models \sigma\phi_2$, where σ is the substitution matching P to $R_1|_A$.

- $\mathcal{Q}, R \models [P]_A\,\phi$ if $R = R_1R_2$, for some R_1 and R_2, and either P does not match $R_1|_A$ or P matches $R_1|_A$ and $\mathcal{Q}, R_1 \models \sigma\phi$, where σ is the substitution matching P to $R_1|_A$.

Note: The semantics of $\mathcal{Q}, R \models [P]_A\,\phi$ can be expressed in terms of the semantics of $\mathcal{Q}, R \models \phi_1\,[P]_A\,\phi_2$ by setting ϕ_1 to $true$ and requiring that R_0 be empty.

Semantic Entailment:

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

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.

Since the logic and the proof system are an extension of the earlier work [18], here we only give brief and informal proofs for those axioms that are same or similar to axioms in [18].

Since the composition ideas are the main contribution of this paper, we give more detailed proofs for composition rules.

First, we state two general lemmas about an execution model. Both lemmas are proved in [18] and we omit the proofs here.

Lemma C.1 *For every send action there is a corresponding receive action and vice-versa. More formally,* $EVENT(R, X, \langle m\rangle, \emptyset, \emptyset) \supset \exists Y.EVENT(R, Y, (x), m, x)$ *and* $EVENT(R, X, (x), m, x) \supset \exists Y.EVENT(R, Y, \langle m\rangle, \emptyset, \emptyset)$

Lemma C.2 *For any configuration* $\mathbf{C}$ *of protocol* $\mathcal{Q}$*, and any run* R*, if principal* $X \in HONEST(\mathbf{C})$*, then* $R|_X$ *is an interleaving of traces of roles of* $\mathcal{Q}$ *executed by* X*.*

C.1 Axioms for protocol actions

This set of axioms describes formulas that are true as a result of an agent performing some action.

C.1.1 AA1 $\phi[a]_X \diamondminus (\mathsf{a} \wedge \ominus\phi)$

The soundness of this axiom follows directly from the semantics of temporal operators.

C.1.2 AN2 $\phi[(\nu n)]_X\ \mathsf{Has}(Y, n) \supset (Y = X)$

Informally, this axiom says that fresh nonces are secret. If a principal X generates a new value m and takes no further actions, then X is the only principal 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 [18].

C.1.3 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".

C.2 Possession axioms

This set of axioms describes ways in which agents can accumulate knowledge.

C.2.1 PROJ $\mathsf{Has}(X,(x,y)) \supset \mathsf{Has}(X,x) \wedge \mathsf{Has}(X,y)$

C.2.2 TUP $\mathsf{Has}(X,x) \wedge \mathsf{Has}(X,y) \supset \mathsf{Has}(X,(x,y))$

C.2.3 ENC $\mathsf{Has}(X,x) \wedge \mathsf{Has}(X,K) \supset \mathsf{Has}(X,\{|x|\}_K)$

Informally, these axioms say that if an agent has all the necessary parts to build some term then he has the term itself. Also, if an agent has a tuple, then he also has the components.

Soundness of these three axioms follows directly from the semantics of the predicate "Has", and logical conectives "$\wedge$" and "$\supset$".

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

Proofs for other two axioms are similar.

C.2.4 ORIG $\diamondminus \mathsf{New}(X,n) \supset \mathsf{Has}(X,n)$

C.2.5 REC $\diamondminus \mathsf{Receive}(X,x) \supset \mathsf{Has}(X,x)$

C.2.6 DEC $\diamondminus\ \mathsf{Decrypt}(X,\{|x|\}_K) \supset \mathsf{Has}(X,x)$

Informally, these axioms make connection between agents knowledge and the actions he did. An agent that has some term if he performed some action involving that term in the past.

Soundness of these three axioms follows directly from the semantics of the predicate "Has", temporal operator "$\diamondminus$" and "$\supset$".

Let us prove the axiom $\mathbf{DEC}$. When $Q,R \not\models \diamondminus\ \mathsf{Decrypt}(X,\{|x|\}_K)$ then $Q,R \models \mathbf{DEC}$ holds trivially. Otherwise, by the semantics of "$\diamondminus$", there is a prefix R' of R such that $Q,R' \models \mathsf{Decrypt}(X,\{|x|\}_K)$. By the semantics of "Decrypt", it has to be the case that $LAST(R',A,(\{|m|\}_K/\{|x|\}_{\overline{K}}),m,x)$ also holds. Since R' is a prefix of R, last event of R' is also an event of R and therefore, $EVENT(R',A,(\{|m|\}_K/\{|x|\}_{\overline{K}}),m,x)$ also holds. $Q,R \models \mathsf{Has}(X,x)$ now follows from the semantics of the predicate "Has".

Proofs for other two axioms are similar.

C.3 Encryption and signature

This set of axioms states properties of public-key encryption and signature mechanisms.

C.3.1 SEC $\mathsf{Honest}(\hat{X}) \wedge \diamondminus \mathsf{Decrypt}(Y, \{\!|n|\!\}_X) \supset (\hat{Y} = \hat{X})$

Informally, **SEC** says that if a principal $\hat{X}$ is honest, and some principal $\hat{Y}$ has decrypted a message $\{\!|n|\!\}_X$ (i.e. a message encrypted with X's public key), then $\hat{Y}$ must be $\hat{X}$. In other words, if $\hat{X}$ is honest, then nobody but $\hat{X}$ knows $\hat{X}$'s private key.

For a detailed soundness proof of this axiom see [18].

C.3.2 VER $\mathsf{Honest}(\hat{X}) \wedge \diamondminus \mathsf{Verify}(Y, \{\!|n|\!\}_{\overline{X}}) \supset$
$\exists X.\exists m.(\diamondminus \mathsf{Send}(X, m) \wedge \mathsf{Contains}(m, \{\!|n|\!\}_{\overline{X}}))$

Informally, **VER** says that if a principal $\hat{X}$ is honest, and some principal $\hat{Y}$ has verified a signature $\{\!|n|\!\}_X$ (i.e. a message signed with X's private key), then $\hat{X}$ must have send the signature out in some process 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 $\hat{X}$.

The soundness of this axioms follows from the assumption that honest principals do not reveal their public keys, from the semantics of "Has", "Verifies", "$\diamondminus$" and from Lemma C.1.

C.4 Uniqueness of Nonces

This set of axioms is used for reasoning about nonces.

C.4.1 N1 $\diamondminus \mathsf{New}(X, n) \wedge \diamondminus \mathsf{New}(Y, n) \supset (X = Y)$

Informally, this axiom says that nonces are unique across sessions. If some processes 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".

C.4.2 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 a session. If some processes 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".

C.4.3 F1 $\diamondminus \mathsf{Fresh}(X, n) \wedge \diamondminus \mathsf{Fresh}(Y, n) \supset (X = Y)$

Informally, this axiom says that the freshness is local. If some nonce n is fresh in two different processes 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

C.5.1 CON $\mathsf{Contains}((x, y), x) \wedge \mathsf{Contains}((x, y), y)$

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

C.6.1 P1 $\mathsf{Persist}(X, t)[a]_X \mathsf{Persist}(X, t)$ where $\mathsf{Persist} \in \{\diamondminus\ \phi, \mathsf{Has}\}$

Informally this axiom says that the for some formulas stay valid when an agent does additional actions.

Since the semantics of the predicate "Has" is based on the existence of a certain event in a run, adding additional events to the run cannot make this predicates false.

Also, the fact that some formula was true in the past remains valid when add additional actions to the run.

C.6.2 P3 $\mathsf{HasAlone}(X, t)[a]_X \mathsf{HasAlone}(X, t)$, where $(t \not\subseteq_v a) \vee (a \neq \langle m \rangle)$

Informally this axiom says that the for some t 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 Lemma C.1.

C.6.3 F $\mathsf{Fresh}(X, t)[\langle m \rangle]_X \neg \mathsf{Fresh}(X, t)$, where $(t \subseteq m)$

Informally, this axiom talks about loss of freshnes. 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 Axioms for temporal ordering

C.7.1 T1 $\diamondminus\ (\phi \wedge \psi) \supset (\diamondminus\ \phi \wedge \diamondminus\ \psi)$

C.7.2 T2 $\diamondminus\ (\phi \vee \psi) \supset (\diamondminus\ \phi \vee \diamondminus\ \psi)$

C.7.3 T3 $\ominus \neg \phi \leftrightarrow \neg \ominus \phi$

These three axions describe the relation between temporal operators "$\diamondminus$", "$\ominus$", and logical conectives.

Since they are standard axioms from temporal logic, they can be verified easily and We omit the proof here.

C.7.4 AF1 $\theta[a_1 \dots a_n]_X$ $\mathsf{After}(\mathsf{a_1}, \mathsf{a_2}) \wedge \dots \wedge \mathsf{After}(\mathsf{a_{n-1}}, \mathsf{a_n})$

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

The soundness of this axiom follows directly from the definition of "After" and semantics of temporal operators.

C.7.5 AF2 $\mathsf{Honest}(\hat{X}) \wedge \mathsf{Honest}(\hat{Y}) \wedge \mathsf{Honest}(\hat{Z}) \supset$ $(\mathsf{After}(\mathsf{a_1}(X), \mathsf{a_2}(Y)) \wedge \mathsf{After}(\mathsf{a_2}(\mathsf{Y}), \mathsf{a_3}(Z)) \supset \mathsf{After}(\mathsf{a_1}(X), \mathsf{a_3}(Z)))$

This axiom says that the relation After is transitive when actions of honest principals are considered.

It is trivially sound since we assumed that all actions of honest participants are distinct.

C.7.6 AF3 $(\diamondminus\, (\mathsf{Send}(X, m) \wedge \ominus \mathsf{Fresh}\,(X, n)) \wedge \diamondminus\, \mathsf{a}(Y) \wedge$ $\mathsf{Contains}(m, n) \wedge \mathsf{Contains}(a, n) \wedge (Y \neq X)) \supset \mathsf{After}(\mathsf{Send}(X, m), \mathsf{a}(Y))$

Informally, this axiom says that the all actions a involving the term n which was fresh at some point, must have happened after the first time that n was send out.

The soundness of this axioms follows from the semantics of the predicate "Fresh", semantics of temporal operators and Lemma C.1.

C.8 Axioms Diffie-Hellman key exchange

This set of axioms are used to reason about properties of Diffie-Hellman key exchange. Remember, Computes is just a shortcut.

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

C.8.1 DH1 $\mathsf{Computes}(X, g^{ab}) \supset \mathsf{Has}(X, g^{ab})$

Informally, this axiom says that if some agent 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".

C.8.2 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".

C.8.3 DH3 $(\diamondminus\, \mathsf{Receive}(X, m) \wedge \mathsf{Contains}(m, g^{ab})) \supset$ $\exists Y, m'.(\mathsf{Computes}(Y, g^{ab}) \wedge \diamondminus\, \mathsf{Send}(Y, m') \wedge \mathsf{Contains}(m', g^{ab}))$

Informally, this axiom says that if someone receives a Diffie-Hellman shared secret then there must be some agent 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 C.1, 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 $\mathbf{DH2}$ that means that Y must have computed g^{ab} himself.

C.8.4 $\mathbf{DH4}$ $\mathsf{Fresh}(X, a) \supset \mathsf{Fresh}(X, g^a)$

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

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

C.9.1 Generic rule G1

$\mathbf{G1}$ 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 "$\wedge$".

C.9.2 Generic rule G2

Similarly, $\mathbf{G2}$ follows from the semantics of the logical connectives.

C.9.3 Generic rule G3

$\mathbf{G3}$ is valid because if ϕ is true after any run, then ϕ is true after a specific run that contains actions P.

C.10 Temporal generalization rule

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

This is a standard proof rule from temporal logic and it can be verified easily. We omit the proof here.

C.11 Composition rules

This set of rules is used for reasoning about protocol composition.

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

C.11.1 The weakening rule W

The weakening rule $\mathbf{W}$ states that a formula ϕ which is provable from a set of hypotheses, Γ, remains provable if additional formulas are added to the set of hypotheses.

This rule is trivially sound since $\Gamma \models \phi$ implies $\Gamma \cup \Gamma' \models \phi$.

C.11.2 Sequential composition rule C1

The protocol composition rule $\mathbf{C1}$ gives us a way of sequentially composing two roles P and P' when post-condition of P, matches the pre-condition or P'.

Assume that Q is a protocol and Γ is the set of formulas such that $\Gamma \models \phi_1[P]_A\phi_2$ and $\Gamma \models \phi_2[P']_A\phi_3$. We need to prove that $\Gamma \models \phi_1[P;P']_A\phi_3$. When $Q \not\models \Gamma$ this is trivially true. Assume that $Q \models \Gamma$, now it has to be that $Q \models \phi_1[P]_A\phi_2$ and $Q \models \phi_2[P']_A\phi_3$. Let $R = R_0R_1R_2$ be a run of Q such that R_1 matches $P;P'|_A$ under substitution σ and $Q, R_0 \models \sigma\phi_1$. Run R can be written as $R = R_0R_1'R_1''R_2$ where R_1' matches $P|_A$ under σ and R_1'' matches $P'|A$ under σ. It follows that $Q, R_0R_1' \models \sigma\phi_2$ and therefore $q, R_0R_1'R_1'' \models \sigma\phi_3$.

C.11.3 Parallel composition meta-rule C2

The protocol composition rule $\mathbf{C2}$ states that all invariants provable in both Q and Q' are provable in their composition $Q \circ Q'$. Remember that we are only considering invariants that are provable using one application of the honesty rule.

Suppose that the formula ϕ can proved in Q and Q' using only one application of the honesty rule. By the definition of the honesty rule $\forall \rho \in \mathcal{Q} \cup \mathcal{Q}'.\forall P \epsilon BS(\rho).\ \phi\ [P]_X\ \phi$.

Every basic sequence of a role in $Q \circ Q'$ is either a basing sequence P of $Q \cup Q'$ or a concatenation of two basic sequences in $Q \cup Q'$, in the first case it trivially follows that $Q \circ Q' \models \phi[P]_X\phi$, in the second case the same follows by the application of the composition rule $\mathbf{C1}$.

C.12 The Honesty rule

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

Assume that $Q \models [\ ]_X\phi$ and $Q \models \phi\ [P]_X\ \phi$ for all roles $\rho \in Q$ and for all basic sequences $P \in BS(\rho)$.

We must show that $\mathcal{Q}, R \models \mathsf{Honest}(\hat{X}) \supset \phi$, for any run R. Assume $\mathcal{Q}, R \models \mathsf{Honest}(X)$. Then from the semantics of Honest and Lemma C.2, it has to be that $R|_X$ is an interleaving of traces of roles of $\mathcal{Q}$ carried out by X and therefore a concatenation of basic sequences. $Q \models \phi$ now follows from the composition rule $\mathbf{C1}$.

D Formal Correctness Proofs of Protocols

A complete proof of the authentication property for the initiator role of the Challenge-Response protocol ($\mathbf{Init_{CR}}$) is given in Table 10. The proof of the shared secret property of *ISO-9798-3* is given in Table 11. This proof uses composition ideas and the structure of the proof has been discussed in the body of the paper. Below, we discuss the structure of the proof of the Challenge-Response protocol and provide some insight on the proof technique used to prove authentication results in this logic.

Proof Structure of Challenge-Response Protocol: The formal proof in Table 10 naturally breaks down into three parts:

- Lines (1)–(5) assert what actions were executed by Alice in the initiator role as well as the order in which those actions occured. Specifically, in this part of the proof, the order in which Alice executed the send-receive actions is proved, i.e., ActionsInOrder(Send(A,msg1), Receive(A,msg2)).
- In lines (6)–(10), we first use the fact that the signatures of honest parties are unforgeable (axiom $\mathbf{VER}$), to conclude that Bob must have sent out some message containing his signature since Alice received the signature in msg2. 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 msg2 of the protocol and must have previously received msg1 from Alice. Thus, the order in which Bob executed his actions are proved, i.e., ActionsInOrder(Receive(B,msg1), Send(B,msg2)).
- Finally, in lines (11)–(13), the temporal ordering rules are used to establish a total ordering among the send-receive actions of Alice and Bob. Line (11) concludes that Bob must have received msg1 after Alice sent it since msg1 contains a fresh nonce. Line (12) uses the same argument for msg2 sent by Bob. Finally, line (13) uses the transitivity axiom to conclude that the following formula is true:
ActionsInOrder(Send(A,msg1), Receive(B,msg1), Send(B,msg2), Receive(A,msg2)).
This formula means that Alice and Bob have matching conversations. This is precisely the formulation of authentication that we needed to prove.

This proof exemplifies a general methodology 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:

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

- 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 msg2 to Alice after having received msg1 from her).
- 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. They can often be translated into a natural language argument, similar to ones that a protocol designer might use to convince herself of the correctness of a protocol.

$$\begin{array}{rll}
\mathbf{AA1}, \mathbf{T1}, \mathbf{F} & \mathsf{Fresh}(A, m)[\langle \hat{A}, \hat{B}, m\rangle]_A & \\
& \quad \diamondminus(\mathsf{Send}(A, \{\hat{A}, \hat{B}, m\}) \wedge \ominus \mathsf{Fresh}(A, m)) & (1) \\
\mathbf{AA1}, \mathbf{T1} & [(\hat{B}, \hat{A}, n, \{\!|m, n, \hat{A}|\!\}_{\overline{B}})]_A \diamondminus \mathsf{Receive}(A, \{\hat{B}, \hat{A}, n, \{\!|m, n, \hat{A}|\!\}_{\overline{B}}\}) & (2) \\
\mathbf{AA1}, \mathbf{T1} & [(\{\!|m, n, \hat{A}|\!\}_{\overline{B}} / \{\!|m, n, \hat{A}|\!\}_B)]_A \ \diamondminus \mathsf{Verify}(A, \{\!|m, n, \hat{A}|\!\}_{\overline{B}}) & (3) \\
\mathbf{AA1}, \mathbf{T1} & [\langle \hat{A}, \hat{B}, \{\!|m, n, \hat{B}|\!\}_{\overline{A}}\rangle]_A \ \diamondminus \mathsf{Send}(A, \{\hat{A}, \hat{B}, \{\!|m, n, \hat{B}|\!\}_{\overline{A}}\}) & (4) \\
\mathbf{AF1}, \mathbf{AF2} & \mathsf{Fresh}(A, m)[\langle \hat{A}, \hat{B}, m\rangle(x)(x/\hat{B}, \hat{A}, n, \{\!|m, n, \hat{A}|\!\}_{\overline{B}}) & \\
& (\{\!|m, n, \hat{A}|\!\}_{\overline{B}} / \{\!|m, n, \hat{A}|\!\}_B)\langle \hat{A}, \hat{B}, \{\!|m, n, \hat{B}|\!\}_{\overline{A}}\rangle]_A & \\
& \quad \mathsf{ActionsInOrder}(& \\
& \qquad \mathsf{Send}(A, \{\hat{A}, \hat{B}, m\}), & \\
& \qquad \mathsf{Receive}(A, \{\hat{B}, \hat{A}, n, \{\!|m, n, \hat{A}|\!\}_{\overline{B}}\}), & \\
& \qquad \mathsf{Send}(A, \{\hat{A}, \hat{B}, \{\!|m, n, \hat{B}|\!\}_{\overline{A}}\})) & (5) \\
(5), \mathbf{F1}, \mathbf{P1}, \mathbf{G2} & \mathsf{Fresh}(A, m)[\mathbf{Init_{CR}}]_A \neg \diamondminus \mathsf{Fresh}(B, m) & (6) \\
\mathbf{VER} & \mathsf{Honest}(\hat{B}) \wedge \diamondminus \mathsf{Verify}(A, \{\!|m, n, \hat{A}|\!\}_{\overline{B}}) \supset & \\
& \quad \exists B. \exists m'.(\diamondminus \mathsf{Send}(B, m') \wedge \mathsf{Contains}(m', \{\!|m, n, \hat{A}|\!\}_{\overline{B}})) & (7) \\
(3), (7), \mathbf{P1}, \mathbf{G1-3} & \mathsf{Fresh}(A, m)[\mathbf{Init_{CR}}]_A \mathsf{Honest}(\hat{B}) \supset & \\
& \quad \exists B. \exists m'.(\diamondminus \mathsf{Send}(B, m') \wedge \mathsf{Contains}(m', \{\!|m, n, \hat{A}|\!\}_{\overline{B}})) & (8) \\
\mathbf{HON} & \mathsf{Honest}(\hat{B}) \supset (((\diamondminus \mathsf{Send}(B, m_0) \wedge & \\
& \quad \mathsf{Contains}(m_0, \{\!|m, n, \hat{A}|\!\}_{\overline{B}}) \wedge \neg \diamondminus \mathsf{Fresh}(B, m)) \supset & \\
& \qquad (\ m_0 = \{\hat{B}, \hat{A}, \{n, \{\!|m, n, \hat{A}|\!\}_{\overline{B}}\}\} \wedge & \\
& \qquad \diamondminus(\mathsf{Send}(B, \{\hat{B}, \hat{A}, \{n, \{\!|m, n, \hat{A}|\!\}_{\overline{B}}\}\}) \wedge \ominus \mathsf{Fresh}(B, n)) \wedge & \\
& \qquad \mathsf{ActionsInOrder}(\mathsf{Receive}(B, \{\hat{A}, \hat{B}, m\}), & \\
& \qquad\quad \mathsf{Send}(B, \{\hat{B}, \hat{A}, \{n, \{\!|m, n, \hat{A}|\!\}_{\overline{B}}\}\}))))) & (9) \\
(6), (8), (9), \mathbf{G1-3} & \mathsf{Fresh}(A, m)[\mathbf{Init_{CR}}]_A \mathsf{Honest}(\hat{B}) \supset & \\
& \quad \diamondminus(\mathsf{Send}(B, \{\hat{B}, \hat{A}, \{n, \{\!|m, n, \hat{A}|\!\}_{\overline{B}}\}\}) \wedge \ominus \mathsf{Fresh}(B, n)) \wedge & \\
& \quad \mathsf{After}(\mathsf{Receive}(B, \{\hat{A}, \hat{B}, m\}), \mathsf{Send}(B, \{\hat{B}, \hat{A}, \{n, \{\!|m, n, \hat{A}|\!\}_{\overline{B}}\}\})) & (10) \\
(1), (10), \mathbf{AF3} & \mathsf{Fresh}(A, m)[\mathbf{Init_{CR}}]_A \mathsf{Honest}(\hat{B}) \supset & \\
& \quad \mathsf{After}(\mathsf{Send}(A, \{\hat{A}, \hat{B}, m\}), \mathsf{Receive}(B, \{\hat{A}, \hat{B}, m\})) & (11) \\
(10), (5), \mathbf{AF3} & \mathsf{Fresh}(A, m)[\mathbf{Init_{CR}}]_A \mathsf{Honest}(\hat{B}) \supset & \\
& \quad \mathsf{After}(\mathsf{Send}(B, \{\hat{B}, \hat{A}, \{n, \{\!|m, n, \hat{A}|\!\}_{\overline{B}}\}\}), & \\
& \qquad \mathsf{Receive}(A, \{\hat{B}, \hat{A}, n, \{\!|m, n, \hat{A}|\!\}_{\overline{B}}\}) & (12) \\
(10), (11), (12), \mathbf{AF2} & \mathsf{Fresh}(A, m)[\mathbf{Init_{CR}}]_A \mathsf{Honest}(\hat{B}) \supset & \\
& \quad \mathsf{ActionsInOrder}(\mathsf{Send}(A, \{\hat{A}, \hat{B}, m\}), \mathsf{Receive}(B, \{\hat{A}, \hat{B}, m\}), & \\
& \qquad \mathsf{Send}(B, \{\hat{B}, \hat{A}, \{n, \{\!|m, n, \hat{A}|\!\}_{\overline{B}}\}\}), & \\
& \qquad \mathsf{Receive}(A, \{\hat{B}, \hat{A}, n, \{\!|m, n, \hat{A}|\!\}_{\overline{B}}\})) & (13)
\end{array}$$

Table 10. Deductions of $\hat{A}$ executing Init role of Challenge-Response Protocol

$$
\begin{array}{rll}
\mathbf{P3} & \mathsf{HasAlone}(A,a) \wedge \mathsf{Fresh}(A,g^a)[\mathbf{Init_{CR'}}]_A\ \mathsf{HasAlone}(A,a) & (1)\\
CR & \mathsf{HasAlone}(A,a) \wedge \mathsf{Fresh}(A,g^a)[\mathbf{Init_{CR'}}]_A \mathsf{Honest}(\hat{B}) \supset & \\
 & \quad \mathsf{ActionsInOrder}(& \\
 & \quad\quad \mathsf{Send}(A,\{\hat{A},\hat{B},g^a\}), & \\
 & \quad\quad \mathsf{Receive}(B,\{\hat{A},\hat{B},g^a\}), & \\
 & \quad\quad \mathsf{Send}(B,\{\hat{B},\hat{A},\{n,\{\![g^a,n,\hat{A}]\!\}_{\overline{B}}\}\}), & \\
 & \quad\quad \mathsf{Receive}(A,\{\hat{B},\hat{A},n,\{\![g^a,n,\hat{A}]\!\}_{\overline{B}}\})) & (2)\\
\mathbf{HON} & \mathsf{Honest}(\hat{B}) \wedge \diamondminus \mathsf{Send}(B,\{\hat{B},\hat{A},\{n,\{\![g^a,n,\hat{A}]\!\}_{\overline{B}}\}\}) \supset & (3)\\
 & \quad \exists b'.(n = g^{b'} \wedge \mathsf{HasAlone}(B,b')) & \\
(2),(3) & \mathsf{HasAlone}(A,a) \wedge \mathsf{Fresh}(A,g^a)[\mathbf{Init_{CR'}}]_A \mathsf{Honest}(\hat{B}) \supset & \\
 & \quad \exists b.(n = g^b \wedge \mathsf{HasAlone}(B,b)) & (4)\\
\mathbf{AA1},\mathbf{REC},\mathbf{PROJ},\mathbf{P1} & \mathsf{HasAlone}(A,a) \wedge \mathsf{Fresh}(A,g^a)[\mathbf{Init_{CR'}}]_A \mathsf{Has}(A,n) & (5)\\
(1),(4),(5),\mathbf{Computes} & \mathsf{HasAlone}(A,a) \wedge \mathsf{Fresh}(A,g^a)[\mathbf{Init_{CR'}}]_A \mathsf{Honest}(\hat{B}) \supset & \\
 & \quad \exists b.(n = g^b \wedge \mathsf{Computes}(A,g^{ab})) & (6)\\
(1),(4),\mathbf{Computes} & \mathsf{HasAlone}(A,a) \wedge \mathsf{Fresh}(A,g^a)[\mathbf{Init_{CR'}}]_A \mathsf{Honest}(\hat{B}) \supset & \\
 & \quad \exists b.(n = g^b \wedge (\mathsf{Computes}(X,g^{ab}) \supset (X = A \vee X = B))) & (7)\\
(6),(7) & \mathsf{HasAlone}(A,a) \wedge \mathsf{Fresh}(A,g^a)[\mathbf{Init_{CR'}}]_A \mathsf{Honest}(\hat{B}) \supset & \\
 & \quad \exists b.(n = g^b \wedge \mathsf{Computes}(A,g^{ab}) \wedge (\mathsf{Computes}(X,g^{ab}) \supset & \\
 & \quad\quad (X = A \vee X = B)))) & (8)\\
\mathbf{DH2},\mathbf{DH3} & \mathsf{Has}(X,g^{ab}) \supset (\mathsf{Computes}(X,g^{ab}) \vee \exists Y,m'. & \\
 & (\mathsf{Computes}(Y,g^{ab}) \wedge \diamondminus\ \mathsf{Send}(Y,m') \wedge \mathsf{Contains}(m',g^{ab})) & (9)\\
\mathbf{HON} & \mathsf{Honest}(Y) \supset (\mathsf{Computes}(Y,g^{ab}) \supset & \\
 & \neg\exists m'.(\diamondminus\ \mathsf{Send}(Y,m') \wedge \mathsf{Contains}(m',g^{ab}))) & (10)\\
(8),(9),(10) & \mathsf{HasAlone}(A,a) \wedge \mathsf{Fresh}(A,g^a)[\mathbf{Init_{CR'}}]_A \mathsf{Honest}(\hat{B}) \supset & \\
 & \quad \exists b.(n = g^b \wedge \mathsf{Computes}(A,g^{ab}) \wedge (\mathsf{Has}(X,g^{ab}) \supset & \\
 & \quad\quad (X = A \vee X = B)))) & (11)
\end{array}
$$

Table 11. Deductions of $\hat{A}$ executing Init role of Challenge-Response Protocol with g^a substituted for m