A Derivation System for Security Protocols and its Logical Formalization

Anupam Datta Ante Derek John C. Mitchell Dusko Pavlovic

Computer Science Dept.
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) and related protocols, deriving all members of the family from two basic protocols using a small set of refinements and protocol transformations. As initial steps toward associating logical derivations with protocol derivations, we extend a previous security protocol logic with preconditions and temporal assertions. Using this logic, we prove the security properties of the standard signature based Challenge-Response protocol and the Diffie-Hellman key exchange protocol. The ISO-9798-3 protocol is then formally derived by composing these two simple protocols.

1 Introduction

While many historical authentication and secrecy protocols, such as those cataloged by Clark and Jacob [7], may be analyzed independently, modern protocols often have a number of different subprotocols and interrelated modes. The Internet Key Exchange (IKE) protocol [13], 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, three transformations, and seven refinements. The two protocol components are Diffie-Hellman key exchange [10] 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 refining them by features and functions. Examples of this intuitive and appealing idea appear in [9]. Efforts to formalize the practice, and produce modular, reusable derivations of families of protocols, go back to [6]. More recently, Bellare, Canetti and Krawczyk [3] have studied two interesting protocol transformations, which they call *authenticators*, which generically add authentication to a given protocol scheme.

While the protocol derivation system clearly provides a way to start with simple protocols and make them more complicated, the appeal is that derivations accumulate desirable security properties. 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 assign to each protocol component, refinement and transformation a logical formula, expressing its meaning, and formally tied to it by a semantical relation in the style of [12]. 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 initial steps toward associating logical derivations with protocol derivations, 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 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. 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. Finally, in Section 5, 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. This is "puts them together" in a quite general sense. 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 [9], standard Challenge-Response protocol [19], JFKi, JFKr, ISO-9798-3 protocol [2], and the core of the IKE protocol [13].

3 Derivation of the STS Family

In this section we present a derivation of the STS protocol family. 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. The security properties relevant to the STS family of protocols include key secrecy, mutual authentication, denial-of-service protection, identity protection and computational efficiency. The specific way in which computational efficiency is achieved is 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. In order to keep the paper within page limits, 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 [10] 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 initialization part 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 [19])

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

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

Consider the signature-based authenticator, C_2, described previously. It provides unilateral authentication, i.e., after executing the protocol, I is assured that the second message was generated by R in response to the first message. However, R has no way of concluding who she authenticated herself to. The goal of a mutual authentication protocol is to provide the authentication guarantee to both parties. The binding transformation is a mechanism for constructing a mutual authentication protocol from two instances of a unilateral authentication protocol.

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

Figure 1. An example of a binding transformation

An example of a binding transformation is shown in Figure 1. On the left hand side, is a protocol obtained by sequential composition of two instances of the unilateral authentication protocol. On the right is the result of applying the binding transformation to it. By including the two nonces generated by the two parties inside the signature, it is assured that m and n belong to the same session. I and R can therefore both conclude that the other generated the corresponding messages of the protocol in response to their messages. (However, this protocol does not guarantee mutual authentication in the conventional sense, i.e., whenever I completes a session supposedly with R, she cannot be assured that R actually believes that she completed the same session with I. In order to provide that guarantee, the peer's identity needs to be included inside the signatures. This point is 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 [8]. Here, we only touch on the main idea.

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

Figure 2. An example of a cookie transformation

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, she sends an unforgeable token (cookie) back to I which captures the local state, and resumes the protocol only after the cookie is returned by I. Here the cookie is a keyed hash of message m_1 and m_2^c. The key used for this purpose, HK_R, is known only to R. Since expensive computation and creation of state is deferred till it is established that the initiator can receive messages at the IP address which it claimed as its own, the resulting protocol is resistant to blind DoS attacks.

3.3 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 [23]), 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 [15].

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.

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 protcol. 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, I cannot conclude that R believes that she was talking to I since it's possible that an attacker M intercepted I's first message and forwarded it to R acting as M. R's response meant for M is then forwarded to I. I sends the third message of the protocol to R and believes that she has completed a session with R. But M intercepts this message and instead sends her own signature over the two nonces to R. Thus, R concludes that she

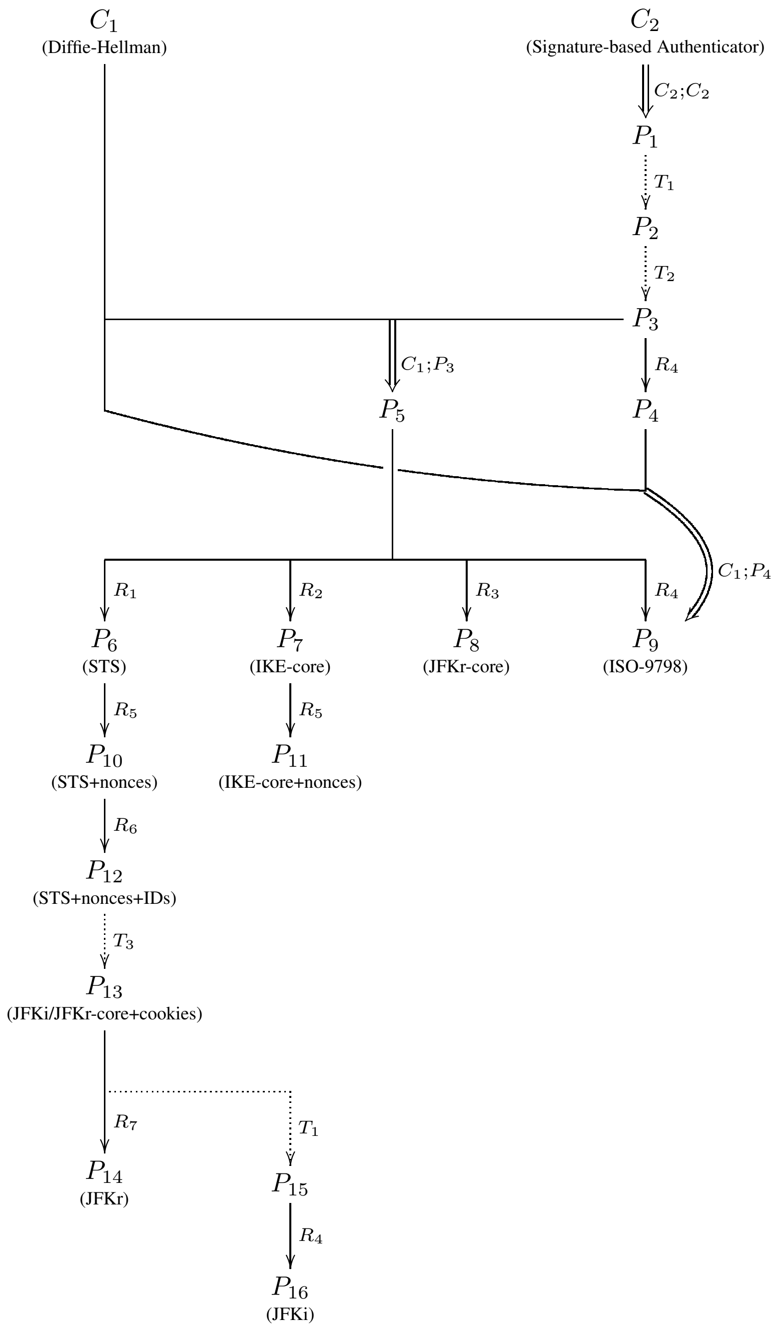

Figure 3. Derivation graph of the STS protocol family

has completed the session with M and not 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.

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 [14] and JFKr; the fourth path includes the ISO-9798-3 protocol. We describe the first derivation path in detail here. The refinements and transformations used in the other paths can be seen in Figure 3. The details are omitted due to space constraints. Also included is a derivation of the *ISO-9798-3* protocol. This derivation is formalized in the next section.

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 [16]. 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}.

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

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 nearly JFKr.

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

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 [16] doesn't work against this protocol.

Path 1.2: JFKi

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

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

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 nearly JFKi.

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

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 [16]. P_{16} retains all the properties of P_{15}.

Derivation of the ISO-9798-3 Protocol

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

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

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

P_3 is refined so that the peer's identity is included inside the signatures. The person-in-the-middle attack on P_3 doesn't succeed against P_4 for exactly the same reason that it doesn't work against P_9 above. 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 usually allow 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. 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. 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. This is in keeping with the main idea of the derivation system and provides evidence that a complete formalization of the derivation system presented in the previous section may be achievable.

The formal system employed towards this end is based on earlier work [12]. 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. Due to space constraints, we only sketch the extensions here, deferring the full technical development to another paper. The most significant extension is a rule for composing protocols. This rule plays 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. While the order in which the actions within a single process are carried out is easily determined, inferring the ordering of actions carried out by different processes requires some thought. Our formalization is based on the idea that if a fresh value n is generated and sent out for the first time as a subterm of message m, then any action carried out by any other principal which involves n must have occurred after the send action.

4.1 Cord Calculus and Security Protocols

Cords [12] are the formalism we use to represent protocols and their parts. They form an action calculus [20, 21, 24], related to π-calculus [22] and spi-calculus [1]. The cords formalism is also similar to the approach of the Chemical Abstract Machine formalism [5], 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 [12]; a brief summary is included in Appendix A.

4.2 A Logic for Protocol Analysis and Derivation

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 N are terms and names. We use ϕ and ψ to indicate predicate formulas, and m to indicate a generic term we call a "message". Each message has the form (source, destination, content), providing source and destination fields in addition to 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 [9] which requires that whenever A and B accept each other's identities at the end of a run, their records of the run should match. Including the source and destination fields in the message allows us to match up send-receive actions.

The η in the predicate formulas serves to identify the instance of a role in which that predicate is true. For example, a principal A could simultaneously engage in two sessions in the initiator role. It is necessary to distinguish between the predicates that are true in the two role instances. Most protocol proofs use formulas of the form $[P]_{X,\eta}\phi$, which means that after X executes actions P in the role instance identified by η, formula ϕ is true about the resulting state of X in η. Here are the informal interpretations of the predicates, with the basis for defining precise semantics discussed in the next section:

The formula $\mathsf{Has}(X, x, \eta)$ means that principal X possesses information x in the role identified by η. 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 $\mathsf{Send}(X, m, \eta)$ means that the last action in a run of the protocol corresponds to principal X sending message m. $\mathsf{Receive}(X, m, \eta)$, $\mathsf{New}(X, t, \eta)$, $\mathsf{Decrypt}(X, t, \eta)$, and $\mathsf{Verify}(X, t, \eta)$ are similarly associated with the receive, new, decrypt and signature verification actions of a protocol. $\Diamond\!\!\!\!-\ \mathsf{a}$ means that in the past, action a was carried out. As the name suggests, $\mathsf{After}(\mathsf{a}, \mathsf{a})$ means that the second action happened after the first action in runs of a protocol. $\mathsf{Fresh}(X, t, \eta)$ means that the term t generated by 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. Finally, the formula $\mathsf{Honest}(X)$ means that the actions of principal 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, X assumes some set of roles and does exactly the actions prescribed by them. We note here that the temporal operator $\Diamond\!\!\!\!-$ and some of the predicates (e.g., Send, Receive) bear semblance to those used in NPATRL [25], the temporal requirements language for the NRL Protocol Analyzer [17, 18]. However, unlike NPATRL, which is used only for specifying protocol requirements, our logic is also used to infer properties of protocols.

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.

The inductive definition of $Q, R \models \phi$ is omitted due to lack of space (see [12] to get a general sense of the approach). 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, 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. 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 $\Diamond\!\!\!\!-\ \mathsf{a}$ 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. Note that the a in axiom $\mathbf{AA1}$ is any one of the 5 actions and a is the corresponding predicate in the logic. $\mathbf{AA1}$ therefore 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. If principal X generates a new value n and does no further actions in this role, then axiom $\mathbf{AN2}$ says that X knows n, $\mathbf{AN1}$ says that no one else knows n, and $\mathbf{AN3}$ says that n is fresh. $\mathbf{AR1}$ says that if X has received a message m, then she knows m. Axiom $\mathbf{AM1}$ is about the binding of static variables of a protocol role. In this case the x_i in $(x_1 \ldots x_n)[\ ]_{X,\eta}$, is a value determined when the roles for each participant were assigned. Typically this will be the identity of the participant, and possibly the identity of other participants an initiator will try to talk to, along with shared keys, etc.

4.3.2 Axioms relating Atomic Predicates

Table 3 lists axioms relating various propositional properties, most of which follow naturally from the semantics of propositional formulas. For example, if X decrypts $\{|n|\}_K$, then X knows n because that is the result of the decryption, and if a principal knows a tuple x, y then he also knows x and y. An important axiom is $\mathbf{N1}$ which states that if a principal X has generated a value n in some role, then that value is distinct from all other values generated in all other roles. $\mathbf{N2}$ states that values generated by different actions within a role are distinct. $\mathbf{N1}$ and $\mathbf{N2}$ together capture the intuition that fresh nonces are unique. $\mathbf{VER}$ and $\mathbf{SEC}$ respectively refer to the unforgeability of signatures and the need to possess the private key in order to decrypt a message encrypted with the corresponding public key. The additional condition requiring principal X to be honest guarantees that the intruder is not in possession of the private keys.

4.3.3 Inference Rules

Table 4 collects the other inference rules. It is clear that most predicates are preserved by additional actions. However, the Fresh predicate is not preserved if the freshly generated value n is sent out in a message (see $\mathbf{F}$). Perhaps, the most interesting inference rule is the protocol composition rule $\mathbf{C}$. It gives us a way of sequentially composing two roles P and P' when the logical formula guaranteed by the execution of P, i.e., the post-condition of P, matches the

Action formulas

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

Formulas

$\phi \quad ::= \quad \diamondminus \mathsf{a} \mid \mathsf{Has}(N, t, \eta) \mid \mathsf{Fresh}(N, t, \eta) \mid \mathsf{Honest}(N) \mid \mathsf{After}(\mathsf{a}, \mathsf{a}) \mid \phi \wedge \phi \mid \neg\phi$

Modal forms

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

Table 1. Syntax of the logic

$$
\begin{array}{rl}
\mathbf{AA1} & [a]_{X,\eta} \diamondminus \mathsf{a} \\
\mathbf{AN1} & [(\nu n)]_{X,\eta}\ \mathsf{Has}(Y, n, \eta') \supset (Y = X) \wedge (\eta' = \eta) \\
\mathbf{AN2} & [(\nu n)]_{X,\eta}\ \mathsf{Has}(X, n, \eta) \\
\mathbf{AN3} & [(\nu n)]_{X,\eta}\ \mathsf{Fresh}(X, n, \eta) \\
\mathbf{AR1} & [(m)]_{X,\eta}\ \mathsf{Has}(X, m, \eta) \\
\mathbf{AM1} & (x_1 \ldots x_n)[\ ]_{X,\eta}\ \mathsf{Has}(X, x_1, \eta) \wedge \ldots \wedge \mathsf{Has}(X, x_n, \eta)
\end{array}
$$

$$\mathsf{HasAlone}(X, n, \eta) \quad \equiv \quad \mathsf{Has}(X, n, \eta) \wedge (\mathsf{Has}(Y, n, \eta') \supset (Y = X) \wedge (\eta' = \eta))$$

Table 2. Axioms for protocol actions

$$
\begin{array}{rl}
\mathbf{DEC1} & \diamondminus \mathsf{Decrypt}(X, \{\!|n|\!\}_K, \eta) \supset \mathsf{Has}(X, \{\!|n|\!\}_K, \eta) \\
\mathbf{DEC2} & \diamondminus \mathsf{Decrypt}(X, \{\!|n|\!\}_K, \eta) \supset \mathsf{Has}(X, n, \eta) \\
\mathbf{PROJ1} & \mathsf{Has}(X, (x, y), \eta) \supset \mathsf{Has}(X, x, \eta) \wedge \mathsf{Has}(X, y, \eta) \\
\mathbf{N1} & \diamondminus \mathsf{New}(X, n, \eta) \wedge \diamondminus \mathsf{New}(Y, n, \eta') \supset (X = Y \wedge \eta = \eta') \\
\mathbf{N2} & \diamondminus \mathsf{New}(X, n, \eta) \wedge \diamondminus \mathsf{New}(X, n', \eta) \supset (n \neq n') \\
\mathbf{SEC} & \mathsf{Honest}(X) \wedge \diamondminus \mathsf{Decrypt}(Y, \{\!|n|\!\}_X, \eta) \supset (Y = X) \\
\mathbf{VER} & \mathsf{Honest}(X) \wedge \diamondminus \mathsf{Verify}(Y, \{\!|n|\!\}_{\overline{X}}, \eta) \supset \\
 & \exists \eta'. \exists m. \diamondminus \mathsf{CSend}(X, m, \eta') \wedge (\{\!|n|\!\}_{\overline{X}} \subseteq m)
\end{array}
$$

$$\diamondminus \mathsf{CSend}(X, \{\!|n|\!\}_K, \eta) \quad \equiv \quad \mathsf{Has}(X, n, \eta) \wedge \mathsf{Has}(X, K, \eta) \wedge \diamondminus \mathsf{Send}(X, \{\!|n|\!\}_K, \eta)$$

Table 3. Relationship between properties

pre-condition required in order to ensure that P' achieves some property. 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 and parallels the way that protocols are derived in the derivation system.

4.3.4 Axioms and Rules for Temporal Ordering

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. $\mathbf{AF3}$ uses the freshness of nonces to reason about the ordering of actions carried out by different principals. Intuitively, it states that if a principal 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 principal 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

Intuitively, 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 suppose that Bob is honest and use properties of Bob's role to reason about how Bob generated his reply. The honesty rule is essentially an invariance rule for proving properties of all roles of a protocol. 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. The honesty rule depends on the protocol and so we write $\mathcal{Q} \vdash [P]\phi$ if $[P]\phi$ is provable using the honesty rule for $\mathcal{Q}$ and the other axioms and proof rules (see [12] for further discussion about the honesty rule).

The honesty rule is used in the proof of correctness of the Challenge-Response protocol that is given in full in Table 8. One place the honesty rule is used is to prove that if B sent out a message with his signature over the two nonces and $A's$ identity, then in the past B must have received a message with source field A and containing one of the nonces. This property (line (10) of Table 8) is an important step in proving that the challenge response protocol has the mutual authentication property.

4.4 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 presented in the previous section. It corresponds to the step in Figure 3, where P_9 is derived from C_1 and P_4 (note that P_9 is actually *ISO-9798-3* and C_1 and P_4 are DH_0 and CR respectively.)

The approach is to assign to each protocol a logical formula which captures (a) the security properties that it guarantees; and (b) the assumptions under which those properties hold. For example, the CR protocol guarantees mutual authentication under the assumption that the data exchanged by the principals are fresh nonces. The property associated with the DH_0 protocol is that the protocol principal possesses a fresh Diffie-Hellman exponential and moreover she is the only one who knows the exponent. It therefore guarantees a form of secrecy: if A's Diffie-Hellman exponent is a and someone else knows the Diffie-Hellman secret g^{ab}, then that person must know b. DH_0 does not require any additional assumptions to be made.

The properties just described can be expressed as modal formulas in our logic where the assumptions appear as pre-conditions and the security properties as post-conditions to the protocol role. If the post-condition of a protocol role matches the pre-condition of another, then using the inference rule **C** in Table 4, we can obtain properties of a bigger protocol by composing the two. In the example that we consider here, the post-condition of DH_0 matches the pre-condition of CR (DH_0 furnishes the fresh data that CR requires). The result of applying the protocol composition rule is the *ISO-9798-3* protocol. In what follows, we express the properties of secrecy and authentication as logical formulas and show how the process of composition ties together the two properties. A direct consequence is that the *ISO-9798-3* protocol allows principals to compute a shared secret. We sketch the outline of the formal derivation relegating detailed proofs to Appendix B.

4.4.1 Challenge Response Protocol, CR

Our formulation of authentication is based on the concept of *matching conversations* [4] and is similar to the idea of proving authentication using *correspondence assertions* [26]. The same basic idea is also presented in [9] where it is referred to as *matching records of runs*. Simply put, it requires that whenever A and B accept each other's identities at the end of a run, their records of the run *match*, i.e., each message that A sent was received by B and vice versa, each send event happened before the corresponding receive event, and moreover the messages sent by each principal (A or B) appear in the same order in both the records. The formalization of authentication as used here is interesting in its own right. However, since the focus of this paper is on the compositional aspect of security protocols and space is limited, we will present the details elsewhere.

Generic Rules:

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

Preservation Rules: (For Persist $\in \{\mathsf{Has}, \diamondminus\, \mathsf{a}\}$)

$$\frac{[P]_{A,\eta}\mathsf{Persist}(X,t,\eta')}{[Pa]_{A,\eta}\mathsf{Persist}(X,t,\eta')}\ \mathbf{P1} \qquad \frac{[P]_{A,\eta}\mathsf{Fresh}(A,n,\eta)}{[Pa]_{A,\eta}\mathsf{Fresh}(A,n,\eta)}\ (\mathbf{n} \not\subseteq \mathbf{a})\ \mathbf{P2}$$

$$\frac{[P]_{X,\eta}\mathsf{HasAlone}(X,n,\eta)}{[Pa]_{X,\eta}\mathsf{HasAlone}(X,n,\eta)}\ (\mathbf{a} \neq \langle \mathbf{m} \rangle)\mathbf{P3} \qquad \frac{[P]_{X,\eta}\mathsf{HasAlone}(X,n,\eta)}{[P\langle m \rangle]_{X,\eta}\mathsf{HasAlone}(X,n,\eta)}\ (\mathbf{n} \not\subseteq_{\mathbf{v}} \mathbf{m})\ \mathbf{P4}$$

Freshness Loss and Composition Rules:

$$\frac{[P]_{A,\eta}\mathsf{Fresh}(A,n,\eta)}{[P\langle m \rangle]_{A,\eta}\neg\mathsf{Fresh}(A,n,\eta)}\ (\mathbf{n} \subseteq \mathbf{m})\ \mathbf{F} \qquad \frac{\phi_1 [P]_{A,}}{\phi_1}$$

Table 4. Inference Rules

$$\mathbf{AF1} \qquad [a_1 \ldots a_n]_{X,\eta}\ \mathsf{After}(\mathsf{a_1},\mathsf{a_2}) \wedge \ldots \wedge \mathsf{After}(\mathsf{a_{n-1}},\mathsf{a_n})$$

$$\mathbf{AF2} \qquad \mathsf{After}(\mathsf{a_1},\mathsf{a_2}) \wedge \mathsf{After}(\mathsf{a_2},\mathsf{a_3}) \supset \mathsf{After}(\mathsf{a_1},\mathsf{a_3})$$

Inter-process ordering rule:

$$\frac{[P]_{X,\eta}\mathsf{Fresh}(X,n,\eta) \quad [Pa_1]_{X,\eta}\neg\mathsf{Fresh}(X,n,\eta) \quad [P'a_2]_{Y,\eta'}}{\mathsf{After}(\mathsf{a_1},\mathsf{a_2})}\ (\mathbf{Y} \neq \mathbf{X} \vee \eta \neq \eta') \wedge (\mathbf{n} \subseteq \mathbf{a_2})\ \mathbf{AF3}$$

Table 5. Axioms and rule for temporal ordering of actions

DH1	$\mathsf{Has}(X, a, \eta) \wedge \mathsf{Has}(X, g^b, \eta) \supset \mathsf{Has}(X, g^{ab}, \eta)$
DH2	$\mathsf{Has}(X, g^{ab}, \eta) \supset \mathsf{Has}(X, g^{ba}, \eta)$
DH3	$\mathsf{Has}(X, g^{ab}, \eta) \supset (\mathsf{Has}(X, a, \eta) \wedge \mathsf{Has}(X, g^b, \eta)) \vee (\mathsf{Has}(X, b, \eta) \wedge \mathsf{Has}(X, g^a, \eta))$
DH4	$\mathsf{Fresh}(X, a, \eta) \supset \mathsf{Fresh}(X, g^a, \eta)$

Table 6. Rules for Diffie-Hellman key exchange

The logical formula specifying the initiator role of the CR protocol is of the form: *precondition [actions] postcondition*, where:

$$\begin{aligned}
pre = \quad & (A\ B\ m\ \eta)\ \mathsf{Fresh}(A, m, \eta) \\
actions = \quad & [\langle A, B, m\rangle(B, A, n, \{\!|m, n, A|\!\}_{\overline{B}}/B, A, y, z)(z/\{\!|m, y, A|\!\}_B)\langle A, B, \{\!|m, y, B|\!\}_{\overline{A}}\rangle]_{A,\eta} \\
post = \quad & \mathsf{Honest}(B) \supset \exists\eta'.(\mathsf{ActionsInOrder}(\mathsf{Send}(A, \{A, B, m\}, \eta), \mathsf{Receive}(B, \{A, B, m\}, \eta'), \\
& \mathsf{Send}(B, \{B, A, \{n, \{\!|m, n, A|\!\}_{\overline{B}}\}\}, \eta'), \mathsf{Receive}(A, \{B, A, \{n, \{\!|m, n, A|\!\}_{\overline{B}}\}\}, \eta))
\end{aligned}$$

The predicate $\mathsf{ActionsInOrder}(\mathsf{a_n}, \mathsf{a_n}, \ldots, \mathsf{a_n})$ means that the actions $a_1, a_2, \ldots, a_n$ were executed in that order. It can be defined in a natural away using the transitivity axiom about the After predicate, $\mathbf{AF2}$.

Let us try to understand the three parts of the formula. The *precondition* includes the static parameter list $(A\ B\ m\ \eta)$ with the parameters identifying the initiator, responder, the data sent out by the initiator and the role instance identifier, and the predicate $\mathsf{Fresh}(A, m, \eta)$ capturing the assumptions made about the static parameters. The sequence of *actions* corresponds to the initiator cord (from now on denoted by $\mathbf{Init_{CR}}$). These are the actions executed by a principal in the initiator role. Finally, the *postcondition* (denoted ϕ_{auth} from now on) captures the security property associated with the initiator role. Intuitively, this formula states that if A initiates a session of the CR protocol with B with fresh data m, then after completing the actions in $\mathbf{Init_{CR}}$, A can be assured of the security property ϕ_{auth}. Note that ϕ_{auth} effectively captures the notion of authentication discussed above. A total ordering is established among all send and receive actions. Moreover, for each 'matching' send-receive action pair, the send action precedes the receive action.

A complete proof of the formula above is presented in Table 8 in Appendix B.

4.4.2 Base Diffie Hellman Protocol, DH_0

In this section, we take a closer look at protocol DH_0 that forms the essence of the standard Diffie-Hellman protocol. Our goal is to assign to DH_0 a logical formula characterizing its security properties and making explicit any assumptions. The first step is to enrich the term language and the protocol logic to allow reasoning about Diffie-Hellman computation. The terms $g(a)$ and $h(a, b)$, respectively representing the Diffie-Hellman exponential $g^a \, mod\, p$ and the Diffie-Hellman secret $g^{ab} \, mod\, p$, are added to the term language. To improve readability, we will use g^a and g^{ab} instead of $g(a)$ and $h(a, b)$. Table 6 presents the rules specific to the way that Diffie-Hellman secrets are computed. $\mathbf{DH1}$ captures the way that a Diffie-Hellman secret is computed from an exponent and an exponential. $\mathbf{DH2}$ captures the commutativity of exponentiation. $\mathbf{DH3}$ captures the hardness of the discrete log problem by stating that in order to compute the Diffie-Hellman secret, at least one exponent and the other exponential needs to be known. $\mathbf{DH4}$ captures the intuition that if a is fresh at some point of a run, then g^a is also fresh at that point.

The 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. The logical formula specifying the initiator role of the DH_0 protocol is again of the form *precondition [actions] postcondition* and is given below.

$$(A\ B\ \eta)\ [(\nu a)]_{A,\eta}\ \langle A\ B\ g^a\ \eta\rangle \mathsf{Fresh}(A, g^a, \eta) \wedge \mathsf{HasAlone}(A, a, \eta)$$

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. Note that the precondition only includes the static parameters. No additional assumptions are made. The postcondition includes a cord calculus construct - the *output interface* $\langle A\ B\ g^a\ \eta\rangle$ - which we haven't encountered till this point. The purpose of this construct will become clear during the composition process.

4.4.3 Composing the Protocols

The final step of the derivation is to obtain a logical formula characterizing the security properties of the *ISO-9798-3* protocol. We sketch an outline of the proof here.

Let us go back and look at the form of the logical formulas characterizing DH_0 and CR:

$$\begin{aligned}
DH_0: \quad & (A\ B\ \eta)\ [\mathbf{Init_{DH_0}}]\ \langle A\ B\ g^a\ \eta\rangle \mathsf{Fresh}(A, g^a, \eta) \\
CR: \quad & (A\ B\ m\ \eta)\ \mathsf{Fresh}(A, m, \eta)\ [\mathbf{Init_{CR}}]\ \phi_{auth}
\end{aligned}$$

Note that the post-condition of DH_0 matches the precondition of CR. We can therefore compose the two formulas by applying the composition rule C. The resulting formula is:

ISO-9798-3 (authentication) : $\quad (A\ B\ \eta)\ [\mathbf{Init_{DH_0}}; \mathbf{Init_{CR}}]\ \phi_{auth}$

The composition operator ';' is defined so that the output parameters of the first cord are substituted into the input static parameter list of the second cord. Composition therefore provides a method for substitution. In this specific example, 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.

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.

$$DH_0: \quad (A\ B\ \eta)\,[\mathbf{Init_{DH_0}}]\,\langle A\ B\ g^a\ \eta\rangle \mathsf{HasAlone}(A, a, \eta)$$
$$CR: \quad (A\ B\ g^a\ \eta)\ \mathsf{HasAlone}(A, a, \eta)\,[\mathbf{Init_{CR}}]\ \mathsf{HasAlone}(A, a, \eta)$$

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

$$\textit{ISO-9798-3 (secrecy)}: \quad (A\ B\ \eta)\,[\mathbf{Init_{DH_0}}; \mathbf{Init_{CR}}]\ \mathsf{HasAlone}(A, a, \eta)$$

The rest of the proof uses the properties of Diffie-Hellman secret computation to prove the following logical formula:

$$\begin{aligned}\textit{ISO-9798-3 (shared-secret)}: \quad & (A\ B\ \eta)\,[\mathbf{Init_{DH_0}}; \mathbf{Init_{CR}}] \\ & \mathsf{Honest(B)} \supset \exists b.\exists \eta'.(\ \phi_{auth} \wedge (h = g(b)) \wedge \\ & \mathsf{Has}(A, g^{ab}), \eta) \wedge (\mathsf{Has}(X, g^{ab}), \eta'') \supset \\ & ((X = A \wedge \eta'' = \eta) \vee (X = B \wedge \eta'' = \eta')))\end{aligned}$$

Intuitively, the property proved is that if B is honest, then A and 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 a shared secret.

5 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 *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 that of 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. Also, 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 interesting to extend the logical system to allow formal reasoning about all the protocol refinement and transformation steps that have been presented in this paper. Finally, in the example that we worked out, we have shown that under certain restrictions, security properties can be preserved under composition. A formal characterization of the conditions under which security protocols can be composed would be a significant result.

Acknowledgements This work was partially supported by ONR, contract numbers 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] 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.

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

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

[6] 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.

[7] 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.

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

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

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

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

[12] 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.

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

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

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

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

[17] 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.

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

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

[20] 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.

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

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

[23] 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.

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

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

[26] 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

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 X, Y, Z, or constants A, B, 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 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.

A.2 Cord Spaces and Runs

A *cord space* is a multiset of cords that may interact via communication. The *runs* of a protocol arise as reaction sequences of cord spaces. The basic reactions within a cord space are shown in Table 7, with the required side conditions for each reaction shown below them. Reaction (1) is a send and receive interaction, showing the simultaneous

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

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 7. Basic reaction steps

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.

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 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. Both the principal name and the role-id appear as subscripts on the square brackets delimiting a cord.

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 [23] and a model presented by Dolev and Yao [11]. 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, that property is preserved.

B Formal Proof of Challenge-Response Protocol

A complete proof of the challenge-response protocol is presented in Table 8.

$\mathbf{AM1}$	$(A\ B\ \eta)[\ ]_{A,\eta}\ \mathsf{Has}(A, A, \eta) \wedge \mathsf{Has}(A, B, \eta)$	(1)
$\mathbf{AN3}$	$[(\nu m)]_{A,\eta}\ \mathsf{Fresh}(A, m, \eta)$	(2)
$\mathbf{AA1}$	$[\langle A, B, m\rangle]_{A,\eta}\ \Diamond\,\mathsf{Send}(A, \{A, B, m\}, \eta)$	(3)
$\mathbf{AA1}$	$[(B, A, n, \{\!\|m, n, A\|\!\}_{\overline{B}})]_{A,\eta}$ $\Diamond\mathsf{Receive}(A, \{B, A, n, \{\!\|m, n, A\|\!\}_{\overline{B}}\}, \eta)$	(4)
$\mathbf{AA1}$	$[(\{\!\|m, n, A\|\!\}_{\overline{B}}/\{\!\|m, n, A\|\!\}_{B})]_{A,\eta}\ \Diamond\,\mathsf{Verify}(A, \{\!\|m, n, A\|\!\}_{\overline{B}}, \eta)$	(5)
$\mathbf{AA1}$	$[\langle A, B, \{\!\|m, n, B\|\!\}_{\overline{A}}\rangle]_{A,\eta}\ \Diamond\,\mathsf{Send}(A, \{A, B, \{\!\|m, n, B\|\!\}_{\overline{A}}\}, \eta)$	(6)
$\mathbf{AF1}, \mathbf{AF2}$	$(A\ B\ \eta)[(\nu m)\langle A, B, m\rangle(x)(x/B, A, n, \{\!\|m, n, A\|\!\}_{\overline{B}})$ $(\{\!\|m, n, A\|\!\}_{\overline{B}}/\{\!\|m, n, A\|\!\}_{B})\langle A, B, \{\!\|m, n, B\|\!\}_{\overline{A}}\rangle]_{A,\eta}$ $\mathsf{ActionsInOrder}(\mathsf{Send}(A, \{A, B, m\}, \eta), \mathsf{Receive}(A, \{B, A, n, \{\!\|m, n, A\|\!\}_{\overline{B}}\}, \eta),$ $\mathsf{Send}(A, \{A, B, \{\!\|m, n, B\|\!\}_{\overline{A}}\}, \eta))$	(7)
$\mathbf{N1}$	$\Diamond\,\mathsf{New}(A, m, \eta) \supset \neg\Diamond\,\mathsf{New}(B, m, \eta')$	(8)
$5, \mathbf{VER}$	$\mathsf{Honest}(B) \wedge \Diamond\,\mathsf{Verify}(A, \{\!\|m, n, A\|\!\}_{\overline{B}}, \eta) \supset$ $\exists\eta'.\exists m'.(\Diamond\,\mathsf{CSend}(B, m', \eta') \wedge (\{\!\|m, n, A\|\!\}_{\overline{B}} \subseteq m'))$	(9)
$\mathbf{HON}$	$\mathsf{Honest}(B) \supset (\exists\eta'.\exists m'.((\Diamond\,\mathsf{CSend}(B, m', \eta') \wedge$ $\{\!\|m, n, A\|\!\}_{\overline{B}} \subseteq m' \wedge \neg\Diamond\,\mathsf{New}(B, m, \eta')) \supset$ $(\ m' = \{B, A, \{n, \{\!\|m, n, A\|\!\}_{\overline{B}}\}\} \wedge \Diamond\,\mathsf{Receive}(B, \{A, B, m\}, \eta') \wedge$ $\mathsf{ActionsInOrder}(\mathsf{Receive}(B, \{A, B, m\}, \eta'), \mathsf{New}(B, n, \eta'),$ $\mathsf{Send}(B, \{B, A, \{n, \{\!\|m, n, A\|\!\}_{\overline{B}}\}\}, \eta')))))$	(10)
$2, 3, 11, \mathbf{AF3}$	$\mathsf{Honest}(B) \supset \mathsf{After}(\mathsf{Send}(A, \{A, B, m\}, \eta),$ $\mathsf{Receive}(B, \{A, B, m\}, \eta'))$	(11)
$11, \mathbf{AF2}$	$\mathsf{Honest}(B) \supset \mathsf{After}(\mathsf{Receive}(B, \{A, B, m\}, \eta'),$ $\mathsf{Send}(B, \{B, A, \{n, \{\!\|m, n, A\|\!\}_{\overline{B}}\}\}, \eta'))$	(12)
$11, 4, \mathbf{AF3}$	$\mathsf{Honest}(B) \supset \mathsf{After}(\mathsf{Send}(B, \{B, A, \{n, \{\!\|m, n, A\|\!\}_{\overline{B}}\}\}, \eta'),$ $\mathsf{Receive}(A, \{B, A, \{n, \{\!\|m, n, A\|\!\}_{\overline{B}}\}\}, \eta))$	(13)
$10 - 13, \mathbf{AF2}$	$\mathsf{Honest}(B) \supset \exists\eta'.(\mathsf{ActionsInOrder}(\mathsf{Send}(A, \{A, B, m\}, \eta),$ $\mathsf{Receive}(B, \{A, B, m\}, \eta'), \mathsf{Send}(B, \{B, A, \{n, \{\!\|m, n, A\|\!\}_{\overline{B}}\}\}, \eta'),$ $\mathsf{Receive}(A, \{B, A, \{n, \{\!\|m, n, A\|\!\}_{\overline{B}}\}\}, \eta))$	(14)

Table 8. Deductions of A executing Init role of CR