A Modular Correctness Proof of IEEE 802.11i and SSL/TLS

CHANGHUA HE
Department of Electrical Engineering, Stanford University
and
MUKUND SUNDARARAJAN, ARNAB ROY, ANUPAM DATTA, ANTE DEREK and JOHN MITCHELL
Department of Computer Science, Stanford University

The IEEE 802.11i wireless networking protocol provides mutual authentication between a network access point and user devices, prior to user connectivity. The protocol consists of several parts, including an 802.1X authentication phase allowing TLS over EAP, the 4-Way Handshake to establish a fresh session key, and an optional Group Key Handshake for group communications. Motivated by previous vulnerabilities in related wireless protocols and changes in 802.11i to provide better security, we carry out a formal proof of correctness using a Protocol Composition Logic previously used for other protocols. The proof is modular, comprising a separate proof for each protocol section and providing insight into the networking environment in which each section can be reliably used. Further, the proof holds for a variety of failure recovery strategies and other implementation and configuration options. Since SSL/TLS is widely used apart from 802.11i, the security proof for SSL/TLS has independent interest.

Categories and Subject Descriptors: C.2.2 [**Computer-Communication Networks**]: Network Protocols

General Terms: Security

Additional Key Words and Phrases: IEEE802.11i, TLS, Protocol Composition Logic

1. INTRODUCTION

Security is a concern in many wireless networks, because intruders can potentially access a network without physically entering the buildings in which it is used. While intended to provide security, the Wired Equivalent Privacy (WEP) [802 1999] protocol lacks good key management and suffers from significant cryptographic problems [Borisov et al. 2001], to the extent that FBI agents have publicly demonstrated that they can break a 128-bit WEP key in about three minutes [Cheung 2005]. For these reasons, the *IEEE Task Group i* has developed the 802.11i Standard [802 2004], ratified in June 2004, to provide confidentiality, integrity, and mutual authentication. 802.11i provides authentication protocols, key management protocols, and data confidentiality protocols that may execute concurrently over a network in which other protocols are also used.

In this paper, we present a formal correctness proof of the 802.11i protocols using *Protocol Composition Logic (PCL)* [Durgin et al. 2001; 2003; Datta et al. 2003; 2004c; 2004a; 2004b; Meadows and Pavlovic 2004]. In previous work, PCL has been proved sound for protocol runs that use any number of principals and sessions, over both symbolic models and (for a subset of the logic at present) over more traditional cryptographic assumptions [Datta et al. 2005]. Therefore, while

there are previous studies [He and Mitchell 2004; 2005] using finite-state analysis to find errors in 802.11i with bounded configurations, we believe that this is the first complete proof of 802.11i for an unbounded number of participants and sessions. Furthermore, the formal proof for the SSL/TLS protocol has independent interest since SSL/TLS is widely used independent of 802.11i (e.g. [vis 2004]).

Our proof consists of separate proofs of specific security properties for 802.11i components - the TLS authentication phase, the 4-Way Handshake protocol and the Group Key Handshake protocol. We then combine the component proofs to show that any staged use of the protocol components achieves the stated security goals. It follows that the components compose securely for a range of failure recovery control flows, including the improvements proposed in [He and Mitchell 2005]. The general result also proves security for other configurations presented in the 802.11i Standards document, including the use of a Pre-Shared Key (PSK) or cached Pair-wise Master Key (PMK). In addition to defining a new composition operation to handle error recovery, we also extend the logic to reason about sequence numbers. The Group Key Handshake protocol uses sequence numbers to prevent replay attacks.

An advantage of PCL is that each proof component identifies not only the local reasoning that guarantees the security goal of that component, but also the environmental conditions that are needed to avoid destructive interference from other protocols that may use the same certificates or key materials. These environment assumptions are then proved for the steps that require them, giving us an invariant that is respected by the protocol. In formulating the proof, we identify the precise conditions that will allow other protocols to be executed in the same environment without interfering with 802.11i. Moreover, our proof provides certain insights into the component protocols. For example, one proof step in showing authentication for the 4-Way Handshake protocol reveals that the *authenticator* (the access point) must be a different principal from the *supplicant* (typically a laptop); without this separation a reflection attack is possible. While this condition is entirely reasonable in standard enterprise installations, it could be violated in an ad hoc network configuration and cause vulnerabilities. Briefly, our analysis yields the following suggestions for implementers:

—To prevent a reflection attack on the 4-Way Handshake and Group Key Handshake, no principal can be both an authenticator and supplicant.
—For TLS security, if a key associated with the CA-issued certificate is used in other protocols, all uses must conform to conditions enumerated in this paper.
—Failure recovery can roll back to the end of any completed component, respecting the invariants discussed in this paper.

The main result of this paper is that 802.11i and SSL/TLS protocols are secure against attacks in our attack model. Specifically, we show that the TLS and the 4Way Handshake protocols guarantee mutual authentication and establish shared secrets. We show that the keys established by the Group Key Handshake are secret and satisfy a key ordering property. We also show that 802.11i error handling and key caching strategies will not compromise its security.

Our results suggest that PCL is suitable for compositional analysis of large protocols, yielding assurance and guidelines for implementation and deployment. Among

the methods and security studies carried out in recent years, such as [Ryan et al. 2000; Burrows et al. 1990; Meadows 1994; Thayer-Fábrega et al. 1998; Millen and Shmatikov 2001; Mitchell et al. 1998; Mitchell et al. 1997; Mitchell 1998; Paulson 1998; Abadi and Gordn 1997], the closest to our study appear to be Paulson's investigations [Paulson 1999; 2001] of TLS and SET [SETCo 1997]. Some advantages of our approach over Paulson's inductive method are a compositional proof method and a higher level of abstraction. Paulson's method involves direct reasoning about an inductively defined set of protocol execution traces, built from the protocol specification and attacker actions. PCL, however, has an axiomatic system that abstracts away arguments about traces and eliminates the need to reason explicitly about attacker actions. This feature, along with the Floyd-Hoare style specification, make proofs in PCL more concise and more readable. Because the semantic soundness of PCL shows that each of the axioms and inference rules are correct for arbitrary protocol runs in the presence of a symbolic attacker, the PCL proof system provides the same semantic guarantees as Paulson's method, without requiring a set of lemmas to be reproved for each protocol that is studied.

The rest of the paper is organized as follows. Section 2 briefly describes the IEEE 802.11i Standard and the Protocol Composition Logic (PCL). Sections 3, 4 and 5 present the analysis of the the TLS protocol, 4-Way Handshake, and the Group Key Handshake, respectively. Section 6 describes the staged composition principle which takes into account the structure of complicated control flows and proves the safe composition of various components of 802.11i. Finally, Section 7 concludes the paper. We also include an Appendices A, B that contains the proof system. For the proof of soundness of the axioms and the rules, we refer the reader to [Datta et al.]. For proofs of soundness of the axioms and the rules in the secrecy framework, we refer the reader to [A. Roy 2006;].

2. OVERVIEW

2.1 Overview of 802.11i

The IEEE 802.11i Standard [802 2004] defines data confidentiality, mutual authentication, and key management protocols intended to provide enhanced security in the MAC layer of a wireless network. This set of protocols together defines a "Robust Security Network Association"(RSNA). The RSNA establishment procedure involves three entities called the *Supplicant* (the wireless station), the *Authenticator* (the access point), and the *Authentication Server* (typically a RADIUS server).

In this paper, we focus on the mutual authentication and key management protocols. A typical RSNA establishment procedure starts by executing an EAP authentication between the supplicant and the authentication server, typically using EAP-TLS, with the authenticator acting as a relay. After the successful completion of the EAP-TLS session, the supplicant and the authentication server have verified each other's identity and have agreed on a shared secret. Then the authentication server moves the secret to the authenticator; the authenticator and supplicant derive a shared Pair-wise Master Key (PMK) from this secret. Afterwards, the authenticator and the supplicant execute a session of the 4-Way Handshake protocol, from which a fresh Pair-wise Transient Key (PTK) is derived to secure subsequent data traffic. Note that, in practice, the authentication server can be implemented

Fig. 1. The 802.11i Control Flow

either in a single device with the authenticator, or through a separate server. In the latter case, it is assumed that the link between the authentication server and the authenticator is physically secure. Therefore, while modelling the protocol, we make the simplifying assumption that the authentication server and the authenticator are the same principal.

While the typical use of 802.11i described above is relatively straightforward, the complete specification is much more complicated due to additional optional components and alternative configurations. For example, 802.11i can adopt other EAP methods for authentication, such as password-based authentication, instead of EAP-TLS. Moreover, in the case of multicast applications, the authenticator can also distribute a fresh group key to all supplicants in a group after the PTK has been established.

In this paper, we focus on the complete RSNA establishment procedure that consists of four components: TLS, 4-Way Handshake, Group Key Handshake, and data sessions. These components are designed to be executed sequentially; however, in order to prove security properties of this procedure, we also have to consider all other possible executions. For example, 4-Way Handshakes can be periodically re-run to refresh the PTK. Failure of one component also leads to other possible execution sequences. In the original 802.11i specification, the entire sequence is restarted if one component fails, as shown in Figure 1(a). As observed in [He and Mitchell 2005], this failure recovery mechanism is inefficient and may be improved as shown in Figure 1(b). We therefore formulate our proof in a way that demonstrates the desired security properties for both control flow graphs.

2.2 Overview of Proof Method

We use Protocol Composition Logic (PCL) [Durgin et al. 2001; Datta et al. 2003; 2004c; 2004b] to prove correctness of the 802.11i protocols. This section contains a brief discussion of PCL relevant to this analysis.

Modelling protocols. A protocol is defined by a set of roles, each specifying a sequence of actions to be executed by an honest agent. In PCL, protocol roles are represented using a simple "protocol programming language" based on *cords* [Durgin et al. 2001]. Figure 2 shows a simple two message protocol in the informal arrows-and-messages notation and the formal programs for roles of the same protocol using the cords notation. Program **Init** describes the actions of a *thread* X executing the initiator role in the protocol with *agent* $\hat{Y}$ as the responder. The possible protocol actions include nonce generation, signatures and encryption, communication steps, and decryption and signature verification via pattern matching. Programs can also depend on input parameters (typically determined by context or the result of set-up operations) and provide output parameters to subsequent operations.

Protocol logic and the proof system. We include an Appendix A that contains the proof system. For the proof of soundness of the axioms and the rules, we

$$X \rightarrow Y \ : \ x$$
$$Y \rightarrow X \ : \ x, SIG_Y(X, x)$$

$$\textbf{Init} \ = \ (X, \hat{Y})[\texttt{new}\ x; \texttt{send}\ \hat{X}, \hat{Y}, x; \texttt{receive}\ \hat{Y}, \hat{X}, x, s; \texttt{match}\ s/SIG_Y(X, x);]_X$$
$$\textbf{Resp} \ = \ (Y)[\texttt{receive}\ \hat{X}, \hat{Y}, x; \texttt{send}\ \hat{Y}, \hat{X}, SIG_Y(X, x);]_Y$$

Fig. 2. Arrows-and-messages vs cords

refer the reader to [Datta et al. ; Durgin et al. 2003; Datta et al. 2004b]. Most protocol proofs use formulas of the form $\theta[P]_X\phi$, which means that starting from a state where formula θ is true, after actions P are executed by the thread X, the formula ϕ is true in the resulting state. Formulas ϕ and ψ typically make assertions about temporal order of actions (useful for stating authentication) and/or the data accessible to various principals (useful for stating secrecy).

The proof system extends first-order logic with axioms and proof rules for protocol actions, temporal reasoning, knowledge, and a specialized form of invariance rule called the *honesty rule*. The honesty rule is essential for combining facts about one role with inferred actions of other roles, in the presence of attackers. Intuitively, if Alice receives a response from a message sent to Bob, the honesty rule captures Alice's ability to use properties of Bob's role to reason about how Bob generated his reply. In short, if Alice assumes that Bob is honest, she may use Bob's role to reason from this assumption.

Compositional proof method. Following the modular design of the protocol, we extensively use the compositional approach developed in [Datta et al. 2004c; 2004b] and prove properties of the whole protocol by combining proofs of its parts.

We separately prove security guarantees of the form $\Gamma \vdash \theta[P]_X\phi$ for 4-Way Handshake, TLS, and the Group Key Handshake in Sections 3, 4, and 5 respectively. Here, the set of formulas Γ includes invariants of the specific protocol component. Assuming that these invariants are satisfied, the respective components possess the stated properties. For each component, invariants are proved using the honesty rule.

In order to prove properties of the complete protocol, we combine guarantees provided by the different components. For example, the 4-Way Handshake provides authentication assuming that the Pair-wise Master Key established by TLS is a shared secret between the supplicant and the authenticator. The combined protocol consisting of a TLS session followed by a 4-Way Handshake therefore provides authentication.

Technically, sequential composition involves matching a *postcondition* of one protocol to a *precondition* of the other, as well as checking that the two protocols satisfy each other's invariants. Ensuring that the protocol components compose safely given the error handling mechanisms in Figure 1 requires an extension of sequential composition. The new notion of composition as well as its application to 802.11i is presented in Section 6.

We use the framework developed in [A. Roy 2006;] to prove the secrecy of the keys generated by TLS, the 4-Way Handshake and the Group key protocols. The proof system for the framework is included for reference in Appendix B. For proofs

$\mathbf{TLS} : \mathbf{Client} = (X, \hat{Y}, V_x)[$
$\quad \texttt{new}\ n_x;\ \texttt{send}\ \hat{X}.\hat{Y}.n_x.V_x;$
$\quad \texttt{receive}\ \hat{Y}.\hat{X}.n_y.V_y;$
$\quad \texttt{new}\ secret;$
$\quad encky := \texttt{pkenc}\ secret, \hat{Y};$
$\quad sigterm := \hat{X}.\hat{Y}.n_x.V_x \cdot \hat{Y}.\hat{X}.n_y.V_y \cdot encky;$
$\quad sigvx := \texttt{sign}\ sigterm, \hat{X};$
$\quad hc2 := \texttt{hash}\ \hat{X}.\hat{Y}.n_x.V_x \cdot \hat{Y}.\hat{X}.n_y.V_y \cdot encky \cdot sigvx \cdot$ *"client"*$, secret;$
$\quad \texttt{send}\ \hat{X}.\hat{Y}.encky.sigvx.hc2;$
$\quad \texttt{receive}\ \hat{Y}.\hat{X}.hs'';$
$\quad hs' := \hat{X}.\hat{Y}.n_x.V_x \cdot \hat{Y}.\hat{X}.n_y.V_y \cdot \hat{X}.\hat{Y}.encky \cdot sig \cdot$ *"server"*$;$
$\quad \texttt{verifyhash}\ hs'', hs', secret;$
$\quad]_X\langle X, \hat{Y}, secret\rangle$

$\mathbf{TLS} : \mathbf{Server} = (Y, V_y)[$
$\quad \texttt{receive}\ \hat{X}.\hat{Y}.n_x.V_x;\ \texttt{new}\ n_y;$
$\quad \texttt{send}\ \hat{Y}.\hat{X}.n_y.V_y;$
$\quad \texttt{receive}\ \hat{X}.\hat{Y}.encky.sig.hc2'';$
$\quad sigterm := \hat{X}.\hat{Y}.n_x.V_x \cdot \hat{Y}.\hat{X}.n_y.V_y \cdot encky;$
$\quad \texttt{verify}\ sig, sigterm, \hat{X};$
$\quad secret := \texttt{pkdec}\ encky, \hat{Y};$
$\quad hc2' := \hat{X}.\hat{Y}.n_x.V_x \cdot \hat{Y}.\hat{X}.n_y.V_y \cdot encky \cdot sig \cdot$ *"client"*$;$
$\quad \texttt{verifyhash}\ hc2'', hc2', secret;$
$\quad hs := \texttt{hash}\ \hat{X}.\hat{Y}.n_x.V_x \cdot \hat{Y}.\hat{X}.n_y.V_y \cdot \hat{X}.\hat{Y}.encky \cdot sig \cdot$ *"server"*$, secret;$
$\quad \texttt{send}\ \hat{Y}.\hat{X}.hs;$
$\quad]_Y\langle Y, \hat{X}, secret\rangle$

Table I. TLS: Client and Server Programs

of soundness of the axioms and the rules in the secrecy framework, we refer the reader to [A. Roy 2006;].

We do not present an analysis of the data confidentiality protocol that is used once keys are established, in this paper. Since the current logic is based on an execution model based on idealized cryptography, a correctness proof of the data transfer protocol is unlikely to be very informative. However, it could be prudent to study the data confidentiality protocol in another manner.

3. TLS

In this section we prove the correctness of the TLS [Dierks and Allen 1999] protocol. Broadly, TLS involves two principals called the TLS client and the TLS server. It guarantees mutual authetication and establishes a shared secret between them.

3.1 Modelling TLS

We first express the TLS protocol in our protocol programming language. TLS has several modes of operation. We restrict our attention to the mode where both the server and the client have certificates, since this mode satisfies the mutual authentication property requirement of the 802.11i Standard. The **TLS : Client** and **TLS : Server** programs are described in Table I. TLS guarantees mutual authentication and establishes a shared secret between the client and the server; the secret(also refered to as the 'master-key' in TLS related literature) is the term *secret* generated by the TLS client role and sent out encrypted under ther server's public key. TLS also includes negotiation for the TLS protocol version and the cipher suite used for the encryption; V_x (V_y) represents the Client's (Server's) preferred protocol version and cipher suite configuration. TLS uses a combination of signatures, keyed hashes and nonces to ensure that both the client and the server have a consistent view of a protocol execution.

We briefly describe the intended execution the TLS client role. The client starts by sending a nonce n_x and its configuration information V_x to the server. It then receives a nonce n_y and the server's configuration information, V_y. It now generates a nonce, *secret*, encrypts it under the server's public key and sends this to the server. It also concatenates all the messages it has seen so far, and sends its signature over this concatentation to the server; this allows the server to verify that the clients view of the protocol execution matches its own. Finally, the client receives from the server a keyed hash (keyed by the term *secret*) of the concatenation of all the messages the server has seen; this allows client to verify that its view of the protocol run matches the server's view. These steps use the `verify` and `verifyhash` actions to check signatures and keyed hashes respectively.

3.2 Security Properties

We now formulate the desired security properties in PCL.

The TLS standard [Dierks and Allen 1999] requires that TLS satisfies the following authentication and secrecy properties:

(1) The principals agree on each other's identity, protocol completion status, the values of the protocol version, cryptographic suite, and the *secret* that the client sends to the server. The authentication property for TLS is formulated in terms of matching conversations [Bellare and Rogaway 1994]. The basic idea of matching conversations is that on execution of the server role, we prove that there exists a role of the intended client with a corresponding view of the interaction. For server $\hat{Y}$, communicating with client $\hat{X}$, matching conversations is formulated as $\phi_{tls,auth}$ defined below. Note that the receive action corresponding to the last message sent by the server is not part of the guarantee as the server recieves no acknowledgement for this message.

$$\phi_{tls,auth} ::= \mathsf{Honest}(\hat{X}) \wedge \mathsf{Honest}(\hat{Y}) \supset \exists X.\mathsf{Has}(X, secret) \wedge$$
$$(\mathsf{Send}(X, tlsm1) < \mathsf{Receive}(Y, tlsm1)) \wedge (\mathsf{Receive}(Y, tlsm1) < \mathsf{Send}(Y, tlsm2))$$
$$\wedge(\mathsf{Send}(Y, tlsm2) < \mathsf{Receive}(X, tlsm2)) \wedge (\mathsf{Receive}(X, tlsm2) < \mathsf{Send}(X, tlsm3))$$
$$\wedge(\mathsf{Send}(X, tlsm3) < \mathsf{Receive}(Y, tlsm3)) \wedge (\mathsf{Receive}(Y, tlsm3) < \mathsf{Send}(Y, tlsm4))$$

$\Gamma_{tls,1}$	$\mathsf{Honest}(\hat{X}) \wedge \mathsf{Sign}(X, sigterm) \supset$ $(\mathsf{Send}(X, tlsm1) < \mathsf{Receive}(X, tlsm2) < \mathsf{Send}(X, tlsm3))$ $\wedge\mathsf{FirstSend}(X, n_x, tlsm1) \wedge \mathsf{FirstSend}(X, secret, tlsm3))$ $\wedge\mathsf{New}(X, secret)$
$\Gamma_{tls,2}$	$\mathsf{Sign}(X, sigterm) \supset \mathsf{New}(X, secret) \wedge (\mathsf{PkEnc}(X, secret, \hat{Z}) \supset \hat{Z} = \hat{Y})$ $\wedge(\mathsf{Send}(X, m) \supset \neg\mathsf{ContainsOpen}(m, secret))$
$\Gamma_{tls,3}$	$\forall$basic sequence P of roles in the system. $\mathsf{SafeNet}(secret, \{\bar{k_Y}\})\ [P]_W\ \mathsf{Hon}(\hat{X}, \hat{Y}) \wedge \supset \mathsf{SendsSafeMsg}(W, secret, \{\bar{k_Y}\})$

Table II. TLS Invariants; Invariants use terms defined in Section 3.2

We use $tlsm1$, tlsm2, $tlsm3$, $tlsm4$ as abbreviations for various TLS messages:
$tlsm1 := \hat{X}.\hat{Y}.n_x.V_x$, $tlsm2 := \hat{Y}.\hat{X}.n_y.V_y$,
$tlsm3 := \hat{X}.\hat{Y}.ENC_Y(secret).sigvx.hc2$, $tlsm4 := \hat{Y}.\hat{X}.hs$.
The terms $sigvx$, $hc2$, hs are defined in Table I.

(2) The *secret* that the client generates should not be known to any principal other than the client and the server. For server $\hat{Y}$ and client $\hat{X}$, this property is formulated as $\phi_{tls,sec}$ defined as:

$$\phi_{tls,sec} ::= \mathsf{Honest}(\hat{X}) \wedge \mathsf{Honest}(\hat{Y}) \wedge \mathsf{Has}(\hat{Z}, secret) \supset \hat{Z} = \hat{X} \vee \hat{Z} = \hat{Y}$$

3.3 Security Guarantee

We now state the main theorem of this section. It has two parts that reflects our assume-guarantee approach. The first part says that if the formulae listed in Table II hold, we have the secrecy and authentication properties at the conclusion of an execution of the TLS server role. The second part of the theorem says that the formulae listed in Table II are invariants of the TLS protocol. Note that the guarantee holds when the server $\hat{Y}$ and the client $\hat{X}$, who it communicates with, follow the protocol, i.e. both principals are *honest*. On the other hand, the attacker(s) may perform arbitrary actions subject to the constraints of the symbolic model(for instance the attacker can only decrypt a public key encryption if it has the associated private key). The guarantee holds even when there are multiple, concurrent sessions of TLS.

Theorem 3.1 TLS Server Guarantee.
(i) On execution of the server role, key secrecy *and* session authentication *are guaranteed if the formulas in Table II hold.*
Formally, $\Gamma_{tls,1} \wedge \Gamma_{tls,2} \vdash [TLS{:}Server]_X \phi_{tls,auth} \wedge \phi_{tls,sec}$
(ii) The formulas in Table II are invariants of TLS. Formally, $TLS \vdash \Gamma_{tls,1} \wedge \Gamma_{tls,2}$.

3.4 Operating Environment

Besides providing a proof of guarantee for the protocol in isolation, an important consequence of our assume-guarantee approach is that the protocol invariants help characterize the class of protocols and deployment scenarios that preserve the security properties of the protocol under consideration. Note that if the formulae in

Table II are invariants of all the protocols in the system, the proofs from in this section imply the secrecy and authentication properties of TLS. We now discuss how these formulae provide insight about settings in which attacks on TLS may exist.

$\Gamma_{tls,1}$ states that if a message containing a signature of a certain format should is sent out, then the principal must have executed certain actions consistent with the TLS client. A principal running a protocol that signs messages indiscriminately could cause this invariant to fail. Such an attack would only be possible if the client certificate used by TLS were shared with other protocols. On the other hand, most well designed protocols would probably never sign a message with that exact format, and thus the invariant would be true for these.

$\Gamma_{tls,2}$ and $\Gamma_{tls,3}$ rule out an undesirable sequence of actions that may allow an intruder to learn the shared secret. Here are some implications of $\Gamma_{tls,2}$, $\Gamma_{tls,3}$. First, the client, which generates the secret, should not send it out in the clear. Second, if the server is tricked into decrypting a term containing the secret using its private key, after which it sends out the contents of the encryption, the *secrecy property* of TLS may be lost. Clearly, if principals use an exclusive public/private key pair for TLS , such an attack is not possible. However, since another protocol (or another stage of 802.11i) may uses the same public/private key pair as TLS, it is important to check that these formulas are invariants of any other protocol. Note that $\Gamma_{tls,3}$ is precisely what needs to hold about the basic sequences of the protocols in the system for proof of secrecy of the term *secret* to go through. For 802.11i, our secrecy proof establishes this; we use $\Gamma_{tls,2}$ in the process.

3.5 Proof of Security of TLS

We now prove Theorem 3.1. We start with the proof of the authentication, $\phi_{tls,auth}$. The first part of the argument uses the authentication property of signatures along with the protocol invariant $\Gamma_{tls,1}$ to argue that there must be a thread of client $\hat{X}$, the intended peer of server $\hat{Y}$, which must have performed certain actions corresponding to the TLS client role. The proof is summarized by the following steps below. For instance, the first line of the proof uses the axioms $\mathbf{AA1}, \mathbf{P1}, \mathbf{AA4}$ to conclude that the TLS Server has performed a certain sequence of actions. The next three steps use the invariant $\Gamma_{tls,1}$ and the server's signature verfication action to conclude that a thread X^0 of server's intended peer X must have performed certain actions.

$$\mathbf{AA1}, \mathbf{P1}, \mathbf{AA4} \quad [\mathbf{TLS} : \mathbf{Server}]_Y (\mathsf{Receive}(Y, tlsm1) < \mathsf{Send}(Y, tlsm2) < \mathsf{Receive}(Y, tlsm3) < \mathsf{Send}(Y, tlsm4)) \tag{1}$$

$$\mathbf{AA1}, \mathbf{P1} \quad [\mathbf{TLS} : \mathbf{Server}]_Y \mathsf{Verify}(Y, SIG_{\hat{X}}\{sigterm\}) \tag{2}$$

$$(-1), \mathbf{VER} \quad [\mathbf{TLS} : \mathbf{Server}]_Y \mathsf{Honest}(\hat{X}) \wedge \hat{X} \neq \hat{Y} \supset \exists X.\mathsf{Sign}(X, sigterm) \tag{3}$$

$$(-1), \Gamma_{tls,1} \text{ and Inst } X \text{ to } X^o \quad [\mathbf{TLS} : \mathbf{Server}]_Y \mathsf{Honest}(\hat{X}) \wedge \hat{X} \neq \hat{Y} \supset (\mathsf{Send}(X^o, \hat{X}, \hat{Y}, m1) < \mathsf{Receive}(X^o, \hat{Y}, \hat{X}, m2) < \mathsf{Send}(X^o, \hat{X}, \hat{Y}, m3)) \wedge \mathsf{New}(X^o, secret) \wedge \mathsf{FirstSend}(X^o, n_x, tlsm1)) \wedge \mathsf{FirstSend}(X^o, secret, tlsm3)) \tag{4}$$

The second part of the proof uses the first part of the proof along with properties of nonces to order the actions of the client thread X^o and the server thread Y; we also conclude that the client thread must have the secret as we are guaranteed that it generated it. This proves the authentication guarantee.

$$\mathbf{AN3}, \mathbf{FS1}, \mathbf{P1} \quad [\mathbf{TLS} : \mathbf{Server}]_Y \mathsf{FirstSend}(Y, n_y, tlsm2)) \tag{5}$$

$$(-1), (4), \mathbf{FS2} \quad [\mathbf{TLS} : \mathbf{Server}]_Y \mathsf{Honest}(\hat{X}) \wedge \hat{X} \neq \hat{Y} \supset \mathsf{Send}(Y, tlsm2) < \mathsf{Receive}(X^o, tlsm2) \tag{6}$$

$$(4), (1), \mathbf{FS2} \quad [\mathbf{TLS} : \mathbf{Server}]_Y \mathsf{Honest}(\hat{X}) \wedge \hat{X} \neq \hat{Y} \supset (\mathsf{Send}(X^o, tlsm1) < \mathsf{Receive}(Y, tlsm1)) \wedge (\mathsf{Send}(X^o, tlsm3) < \mathsf{Receive}(Y, tlsm3)) \tag{7}$$

$$(1), (4), (6), (7), \mathbf{ORIG} \quad [\mathbf{TLS} : \mathbf{Server}]_Y \mathsf{Honest}(\hat{X}) \wedge \hat{X} \neq \hat{Y} \supset \exists X.(\mathsf{Send}(X, tlsm1) < \mathsf{Receive}(Y, tlsm1)) \wedge (\mathsf{Receive}(Y, tlsm1) < \mathsf{Send}(Y, tlsm2)) \wedge (\mathsf{Send}(Y, tlsm2) < \mathsf{Receive}(X, tlsm2)) \wedge (\mathsf{Receive}(X, tlsm2) < \mathsf{Send}(X, tlsm3)) \wedge (\mathsf{Send}(X, tlsm3) < \mathsf{Receive}(Y, tlsm3)) \wedge (\mathsf{Receive}(Y, tlsm3) < \mathsf{Send}(Y, tlsm4)) \wedge \mathsf{Has}(X, secret) \tag{8}$$

We now prove the secrecy property of TLS protocol. Recall that the secrecy guarantee is stated informally as: Suppose the server completes the protocol with the intended client, who is honest, then the only principals who possibly know the secret are the server and the intended client. Let $\mathcal{K} = \{k_Y^-\}$, the server $\hat{Y}$'s private key.

We now perform induction on the basic sequences of various protocol roles to show that honest principals do not perform actions that compromise the secrecy of the

term *secret*. Informally, each induction step asserts that if, at the beginning of the basic sequence, if the term *secret* has not already been compromised($\mathsf{SafeNet}(secret, \mathcal{K})$ holds), then a basic sequence executed by a thread Z does not perform any actions that compromise the secrecy of the term secret($\mathsf{SendSafeMsg}(Z, secret, \mathcal{K})$).

In the proof that follows, the basic sequences $\mathbf{TLS} : \mathbf{Client_3}$ and $\mathbf{TLS} : \mathbf{Server_2}$ are the easy cases; either they are contain no `send` actions or the terms sent out cannot possibly contain the *secret* in the clear.

$$\begin{aligned}\text{Let } [\mathbf{TLS} : \mathbf{Client}_3]_{X'} : [&\texttt{receive } \hat{Y}'.\hat{X}'.hs';\\ &\texttt{verifyhash } hs', \hat{X}'.\hat{Y}'.n'_x.V'_x \cdot \hat{Y}'.\hat{X}'.n'_y.V'_y \cdot \hat{X}'.\hat{Y}'.encky' \cdot sig' \cdot \text{“}server\text{”},\\ &\quad secret';]_{X'}\end{aligned}$$

$$\mathbf{NET1}\quad \mathsf{SafeNet}(s, \mathcal{K})[\mathbf{TLS} : \mathbf{Client}_3]_{X'}\ \mathsf{SendsSafeMsg}(X', secret, \mathcal{K}) \tag{9}$$

$$\begin{aligned}\text{Let } [\mathbf{TLS} : \mathbf{Server}_2]_{Y'} : [&\texttt{receive } \hat{X}'.\hat{Y}'.encky'.sig'.hc2';\\ &sigterm' := \hat{X}'.\hat{Y}'.n'_x.V'_x \cdot \hat{Y}'.\hat{X}'.n'_y.V'_y \cdot encky';\\ &\texttt{verify } sig', sigterm', \hat{X}';\\ &\texttt{verifyhash } hc1'', \hat{X}'.\hat{Y}'.n'_x.V'_x \cdot \hat{Y}'.\hat{X}'.n'_y.V'_y \cdot encky';\\ &secret' := \texttt{pkdec } encky', \hat{Y}';\\ &\texttt{verifyhash } hc2', \hat{X}'.\hat{Y}'.n'_x.V'_x \cdot \hat{Y}'.\hat{X}'.n'_y.V'_y \cdot encky' \cdot sig' \cdot \text{“}client\text{”}, secret';\\ &hs' := \texttt{hash } \hat{X}'.\hat{Y}'.n'_x.V'_x \cdot \hat{Y}'.\hat{X}'.n'_y.V'_y \cdot \hat{X}'.\hat{Y}'.encky' \cdot sig' \cdot \text{“}server\text{”}, secret';\\ &\texttt{send } \hat{Y}'.\hat{X}'.hs';]_{Y'}\end{aligned}$$

$$\begin{aligned}\mathbf{SAF}*\quad &\mathsf{SafeMsg}(HASH[secret'](\hat{X}'.\hat{Y}'.n'_x.V'_x \cdot \hat{Y}'.\hat{X}'.n'_y.V'_y \cdot \hat{X}'.\hat{Y}'.encky' \cdot sig' \cdot\\ &\quad \text{“}server\text{”}), secret, \mathcal{K}) \qquad (10)\\ (-1), \mathbf{NET3}\quad &\mathsf{SafeNet}(s, \mathcal{K})[\mathbf{TLS} : \mathbf{Server}_2]_{Y'}\ \mathsf{SendsSafeMsg}(Y', secret, \mathcal{K}) \qquad (11)\end{aligned}$$

We need to reason more carefully for other basic sequences that do send out nonces by either arguing that the nonces are not equal to the *secret*; or that the secret appears encrypted uder the key $\bar{k_y}$. These arguments can be made if a certain formula Φ hold at *every* point in the run of the protocol; Φ is a conjunction of the following:

$$\Phi^0_{secret} : \forall W.\ \mathsf{New}(W, secret) \supset W = X$$
$$\Phi^1_{secret} : \forall W, \hat{Z}.\ \mathsf{New}(W, secret) \wedge \mathsf{PkEnc}(W, secret, \hat{Z}) \supset \hat{Z} = \hat{Y}$$
$$\Phi^2_{secret} : \forall W, m.\ \mathsf{New}(W, secret) \wedge \mathsf{Send}(W, m) \supset \neg\mathsf{ContainsOpen}(m, secret)$$

A practical proof strategy is starting the induction without figuring out a Φ at the outset and construct parts of the Φ as we do induction over an individual basic sequence. Note that Φ may be easily established using the invariant $\Gamma_{tls,2}$ and line 3 of the authentication proof. The proof of Φ^0_{secret} also additionally uses the axiom **AN1**.

We now use Φ to complete the secrecy proof. The motivation for this structure of the Φ parts is that many of the basic sequences generate new nonces and send them out unprotected or protected under a set of keys different from $\mathcal{K}$. The Φ parts tell us that this is not the way the secret in consideration was sent out. For

example consider one of the parts, Φ^2_{secret} : $\forall W, m.\ \mathsf{New}(W, secret) \wedge \mathsf{Send}(W, m) \supset \neg\mathsf{ContainsOpen}(m, secret)$ - this tells us that the generator of *secret* did not send it out unprotected in the open. The predicate $\mathsf{ContainsOpen}(m, a)$ asserts that a can be obtained from m by a series of unpairings only; no decryption is required.

$$\begin{aligned}
&\text{Let } [\mathbf{TLS} : \mathbf{Server}_1]_{Y'} : [\texttt{receive } \hat{X}'.\hat{Y}'.n'_x.V'_x;\\
&\qquad\qquad \texttt{new } n'_y;\\
&\qquad\qquad \texttt{send } \hat{Y}'.\hat{X}'.n'_y.V'_y;]_{Y'}
\end{aligned}$$

$$\begin{aligned}
\mathbf{AA1} \quad & [\mathbf{TLS} : \mathbf{Server}_1]_{Y'}\ \mathsf{New}(Y', n'_y) \wedge \mathsf{Send}(Y', \hat{Y}'.\hat{X}'.n'_y.V'_y) \wedge \\
& \quad \mathsf{ContainsOpen}(\hat{Y}'.\hat{X}'.n'_y.V'_y, n'_y) && (12)\\
(-1), \Phi^2_{secret}, \mathbf{NEW3} \quad & n'_y \neq secret && (13)\\
\mathbf{SAF}* \quad & [\mathbf{TLS} : \mathbf{Server}_1]_{Y'}\ \mathsf{SafeMsg}(\hat{Y}'.\hat{X}'.n'_y.V'_y, secret, \mathcal{K}) && (14)\\
(-1), \mathbf{NET3} \quad & \mathsf{SafeNet}(secret, \mathcal{K})[\mathbf{TLS} : \mathbf{Server}_1]_{Y'}\ \mathsf{SendsSafeMsg}(Y', secret, \mathcal{K}) && (15)
\end{aligned}$$

$$\begin{aligned}
&\text{Let } [\mathbf{TLS} : \mathbf{Client}_1]_{X'} : [\texttt{new } n'_x;\\
&\qquad\qquad \texttt{send } \hat{X}'.\hat{Y}'.n'_x.V'_x;]_{X'}
\end{aligned}$$

$$\begin{aligned}
\Phi^2_{secret} \quad & n'_x \neq secret && (16)\\
(-1), \mathbf{SAF0} \quad & [\mathbf{TLS} : \mathbf{Client}_1]_{X'}\ \mathsf{SafeMsg}(\hat{X}'.\hat{Y}'.n'_x.V'_x, secret, \mathcal{K}) && (17)\\
(-1), \mathbf{NET3} \quad & \mathsf{SafeNet}(s, \mathcal{K})[\mathbf{TLS} : \mathbf{Client}_1]_{X'}\ \mathsf{SendsSafeMsg}(X', secret, \mathcal{K}) && (18)
\end{aligned}$$

$$\begin{aligned}
&\text{Let } [\mathbf{TLS} : \mathbf{Client}_2]_{X'} : [\texttt{receive } \hat{Y}'.\hat{X}'.n'_y.V'_y;\\
&\qquad\qquad \texttt{new } secret';\\
&\qquad\qquad ency k' := \texttt{pkenc } secret', \hat{Y}';\\
&\qquad\qquad hc1' := \texttt{hash } \hat{X}'.\hat{Y}'.n'_x.V'_x \cdot \hat{Y}'.\hat{X}'.n'_y.V'_y \cdot ency k';\\
&\qquad\qquad sigvx' := \texttt{sign } hc1', \hat{X}';\\
&\qquad\qquad hc2' := \texttt{hash } \hat{X}'.\hat{Y}'.n'_x.V'_x \cdot \hat{Y}'.\hat{X}'.n'_y.V'_y \cdot ency k' \cdot sigvx' \cdot \text{“client”}, secret';\\
&\qquad\qquad \texttt{send } \hat{X}'.\hat{Y}'.ency k'.sigvx'.hc2';]_{X'}
\end{aligned}$$

$$\begin{aligned}
\text{Case} : secret' \neq secret \quad & && (19)\\
\mathbf{SAF}* \quad & [\mathbf{TLS} : \mathbf{Client}_2]_{X'}\ \mathsf{SafeMsg}(E_{pk}[\hat{Y}'](secret'), secret, \mathcal{K}) && (20)\\
(-1), \mathbf{SAF}*, \mathbf{NET3} \quad & \mathsf{SafeNet}(s, \mathcal{K})[\mathbf{TLS} : \mathbf{Client}_2]_{X'}\ \mathsf{SendsSafeMsg}(X', secret, \mathcal{K}) && (21)
\end{aligned}$$

$$\begin{aligned}
\text{Case} : secret' = secret \quad & && (22)\\
& [\mathbf{TLS} : \mathbf{Client}_2]_{X'}\ \mathsf{PkEnc}(X', secret, \hat{Y}') && (23)\\
(-1), \Phi^1_{secret} \quad & \hat{Y}' = \hat{Y} && (24)\\
(-1), \mathbf{SAF}* \quad & [\mathbf{TLS} : \mathbf{Client}_2]_{X'}\ \mathsf{SafeMsg}(E_{pk}[\hat{Y}](secret), secret, \mathcal{K}) && (25)\\
(-1), \mathbf{NET3} \quad & \mathsf{SafeNet}(s, \mathcal{K})[\mathbf{TLS} : \mathbf{Client}_2]_{X'}\ \mathsf{SendsSafeMsg}(X', secret, \mathcal{K}) && (26)
\end{aligned}$$

We can now use the **NET** rule to conclude that $\mathsf{SafeNet}(secret, \mathcal{K})$ is always true and then conclude the proof by using the **POS** axiom, which asserts that we have the secrecy property if all the actions performed by honest principals are *safe* in the sense decribed above. From 15, 11, 18, 21, 26, 9 and by application of the **NET** rule and the **POS** axiom, we have the proof of secrecy:

$\mathbf{4WAY} : \mathbf{AUTH} = (X, \hat{Y}, pmk)$	$\mathbf{4WAY} : \mathbf{SUPP} = (Y, pmk)$
$[\texttt{new}\ x;\ \texttt{send}\ \hat{X}.\hat{Y}.x.\text{“}msg1\text{”};$	$[\texttt{receive}\ \hat{X}.\hat{Y}.x.\text{“}msg1\text{”};$
$\texttt{receive}\ \hat{Y}.\hat{X}.y.\text{“}msg2\text{”}.mic1;$	$\texttt{new}\ y;\ ptk := \texttt{hash}\ x.y, pmk;$
$ptk := \texttt{hash}\ x.y, pmk;$	$mic1 := \texttt{hash}\ y.\text{“}msg2\text{”}, ptk;$
$\texttt{verifyhash}\ mic1, y.\text{“}msg2\text{”}, ptk;$	$\texttt{send}\ \hat{Y}.\hat{X}.y.\text{“}msg2\text{”}.mic1;$
$mic2 := \texttt{hash}\ x.\text{“}msg3\text{”}, ptk;$	$\texttt{receive}\ \hat{X}.\hat{Y}.x.\text{“}msg3\text{”}.mic2;$
$\texttt{send}\ \hat{X}.\hat{Y}.x.\text{“}msg3\text{”}.mic2;$	$\texttt{verifyhash}\ mic2, x.\text{“}msg3\text{”}, ptk;$
$\texttt{receive}\ \hat{Y}.\hat{X}.\text{“}msg4\text{”}.mic3;$	$mic3 := \texttt{hash}\ \text{“}msg4\text{”}, ptk;$
$\texttt{verifyhash}\ mic3, \text{“}msg4\text{”}, ptk;]_X$	$\texttt{send}\ \hat{Y}.\hat{X}.\text{“}msg4\text{”}.mic3;]_Y$

Table III. 4-Way Handshake Program

$$\Phi \wedge \mathsf{Honest}(\hat{X}) \wedge \mathsf{Honest}(\hat{Y}) \supset (\mathsf{Has}(Z, secret) \supset \hat{Z} = \hat{X} \vee \hat{Z} = \hat{Y}) \qquad (27)$$

4. 4-WAY HANDSHAKE

In this section, we prove security properties of the 4-Way Handshake protocol proposed in the 802.11i standard and the modified 4-Way Handshake(that fixes a DoS attack on the 4Way Handshake) proposed in [He and Mitchell 2004; 2005].

The 4-Way Handshake guarantees mutual authentication and establishes a shared secret called Pairwise Temporary Key (PTK) between the authenticator and the supplicant. This key is used for for data confidentiality protocols and the Group Key Handshake Protocol. The protocol needs a pre-established secret shared called the Pair-wise Master Key (PMK), which may be set up via mutual authentication protocols (de facto EAP-TLS, this is precisely the term *secret* from the previous section.The name 'PMK' is consistent with 802.11i terminology.); or it may be pre-configured as a Pre-Shared Key (PSK). The 4Way Handshake may be re-executed using the same PMK to generate a new PTK.

4.1 Modelling 4-Way

During the handshake, the authenticator and supplicant generate fresh nonces, then derive a fresh PTK based on the shared PMK (synonymous with the term *secret* from the previous section), the nonces, and their MAC addresses. They authenticate the key material generated using keyed hashes. The authenticator and supplicant roles of the 4-Way Handshake, expressed formally in our programming language, are listed in Table III. We describe an intended execution of the authenticator program **4WAY** : **AUTH** below.

This program has three input parameters - the authenticator and the supplicant identifiers, and the pre-established shared secret,PMK, represented by the variable pmk. The first action executed by the authenticator X involves generating a fresh nonce x using the action `new` . Then X sends out the first message to Y, which

contains the nonce x and the string "msg1". In practice, message indicators are represented by sequences of bits, but we use strings here for readability. Authenticator X waits to receive a response from Y and then checks for the integrity of the using the `verifyhash` action; this action takes three arguments, the hashed term to be verified, the term which it is supposedly the keyed hash of, and the key. If the second message is valid, X sends out the third message including the nonce x, the string "msg3" and the MIC, and waits for the response. Once a valid fourth message is received and verified, X completes the 4-Way Handshake.

Note that in the 802.11i specifications, the PTK is divided into several parts: KCK (Key Confirmation Key) for computing MIC, KEK (Key Encryption Key) for encrypting the group key, and TK (Temporary Key) for protecting data packets. For expository convenience, we use *ptk* to refer to all of these parts.

4.2 Security Properties

The desired security properties for the 4-Way Handshake as identified by the standard (see Section 8.4.8 in [802 2004]) are:

(1) Confirm the existence of the PMK at the peer.
(2) Ensure that the security association key (PTK) is fresh.
(3) Synchronize the installation of session keys into the MAC.
(4) Transfer the GTK from the Authenticator to the Supplicant.
(5) Confirm the selection of cipher suites.

The definitions below formalize two security properties called *key secrecy* and *session authentication*. Items 1, 2, 3, and 5 are captured by *session authentication*; moreover, *session authentication* can be asserted only when the *key secrecy* is guaranteed. The authenticator may distribute a Group Temporary Key (GTK) to supplicants for use in multicast applications. The third and fourth message of the 4-Way Handshake may optionally set up this key distribution. We discuss item 4 in Section 5. We formalize the secrecy of of the shared secret established by the 4-Way Handshake as follows. For an authenticator $\hat{X}$ communicating with a supplicant $\hat{Y}$, the secrecy property, $\phi_{4way,sec}$ defined below, says that the only principals that could possibly have the term *ptk* are the authenticator and its intended supplicant:

$$\phi_{4way,sec} ::= \mathsf{Honest}(\hat{X}) \wedge \mathsf{Honest}(\hat{Y}) \supset ((\mathsf{Has}(\hat{Z}, ptk) \supset \hat{Z} = \hat{X} \vee \hat{Z} = \hat{Y}))$$

As in the case of TLS, We formalizes a standard notion of authentication called *matching conversations* [Bellare and Rogaway 1994]. It guarantees that the two principals have consistent views of protocol runs. It follows that they agree on terms such as the cipher suite and the freshly generated PTK. We formulate this guarantee for an authenticator thread X communicating with a supplicant $\hat{Y}$:

$$\begin{aligned}
&\phi_{4way,auth} ::= \mathsf{Honest}(\hat{X}) \wedge \mathsf{Honest}(\hat{Y}) \supset \exists Y. \\
&(\mathsf{Send}(X, m1) < \mathsf{Receive}(Y, m1)) \wedge (\mathsf{Receive}(Y, m1)) < \mathsf{Send}(Y, m2)) \\
&\wedge(\mathsf{Send}(Y, m2) < \mathsf{Receive}(X, m2)) \wedge (\mathsf{Receive}(X, m2)) < \mathsf{Send}(X, m3)) \\
&\wedge(\mathsf{Send}(X, m3) < \mathsf{Receive}(Y, m3)) \wedge (\mathsf{Receive}(Y, m3)) < \mathsf{Send}(Y, m4)) \\
&\wedge(\mathsf{Send}(Y, m4) < \mathsf{Receive}(X, m4)) \wedge \mathsf{Has}(Y, ptk)
\end{aligned}$$

$\Gamma_{4way,1}$	$:= \mathsf{Honest}(\hat{X}) \wedge \mathsf{Receive}((\hat{X},\eta), m1) \supset \neg\exists\eta'.\mathsf{Send}((\hat{X},\eta'), m3) \wedge$ $\mathsf{Honest}(\hat{X}) \wedge \mathsf{Send}((\hat{X},\eta), m1) \supset \neg\exists\eta'.(\mathsf{Send}((\hat{X},\eta'), m2) \wedge \mathsf{Send}((\hat{X},\eta'), m4)$
$\Gamma_{4way,2}$	$:= \mathsf{Honest}(\hat{Y}) \wedge \mathsf{Send}(Y,m) \wedge \mathsf{Contains}(m, HASH_{ptk}(\text{“}msg4\text{”})) \supset$ $m = m4 \wedge \mathsf{FirstSend}(Y, HASH_{ptk}(y, \text{“}msg2\text{”}), m2) \wedge \mathsf{New}(Y,y)$ $\wedge(\mathsf{Receive}(Y,m1) < \mathsf{Send}(Y,m2) < \mathsf{Receive}(Y,m3) < \mathsf{Send}(Y,m4))$
$\Gamma_{4way,3}$	$:= \mathsf{Honest}(\hat{X}) \wedge \mathsf{Send}(X,m) \wedge \mathsf{Contains}(m, HASH_{ptk}(\text{“}msg3\text{”})) \supset$ $m = m3 \wedge \mathsf{New}(X,x)$
$\Gamma_{4way,4}$	$\forall$basic sequence P of roles in the system. $\mathsf{SafeNet}(ptk)\ [P]_W\ \mathsf{Hon}(\hat{X},\hat{Y}) \wedge \supset \mathsf{SendsSafeMsg}(W, ptk)$

Table IV. 4-Way Handshake Precondition and Invariants. Invariants use terms defined in Section 4.2

Here, $m1 \ldots m4$ are abbreviations for the messages of the 4-Way handshake.

$$m1 := \hat{X}.\hat{Y}.x.\text{“}msg1\text{”}, \quad m2 := \hat{Y}.\hat{X}.y.\text{“}msg2\text{”}.HASH_{ptk}(y, \text{“}msg2\text{”})$$
$$m3 := \hat{X}.\hat{Y}.x.\text{“}msg3\text{”}.HASH_{ptk}(x, \text{“}msg3\text{”}), \quad m4 := \hat{Y}.\hat{X}.\text{“}msg4\text{”}.HASH_{ptk}(\text{“}msg4\text{”})$$

The main result of this section is Theorem 4.1, which states the security guarantees for the authenticator.

Theorem 4.1 4Way Authenticator Guarantee.
(i) On execution of the authenticator role, key secrecy *and* session authentication *are guaranteed if the formulas in Table IV hold. Formally,*

$$\Gamma_{4way,1} \wedge \Gamma_{4way,2} \wedge \Gamma_{4way,3} \vdash [4WAY{:}AUTH]_X \phi_{4way,auth} \wedge \phi_{4way,sec}$$

(ii) $\Gamma_{4way,2}$, $\Gamma_{4way,3}$ *are invariants of the 4-Way Handshake;* $\Gamma_{4way,1}$ *is an assumption on the environment. Formally,*

$$4WAY \vdash \Gamma_{4way,2} \wedge \Gamma_{4way,3}$$

The theorem states that if the authenticator role is executed, then in the resulting state the desired authentication and secrecy properties are guaranteed. The authenticator deduces its security properties based on the actions that it performs, the properties of certain cryptographic primitives and knowledge of the behavior of an honest supplicant. By definition, an honest principal behaves in accordance with the protocol. In the case of the 4-Way Handshake, the expected behavior of an honest principal is captured by the formulas $\Gamma_{4way,1}$, $\Gamma_{4way,2}$ and $\Gamma_{4way,3}$ listed in Table IV. As in the case of the TLS security guarantee, the theorem has two parts. The first statement of the theorem states that, assuming these formulas hold, the security property is guaranteed. The second part of the theorem states that $\Gamma_{4way,2}$,$\Gamma_{4way,3}$ are an invariants of the 4-Way Handshake protocol. The formula $\Gamma_{4way,1}$ states that no principal performs the roles of both the supplicant and the authenticator. This is a reasonable assumption for a typical 802.11i deployment. This also necessary; as noted in [He and Mitchell 2005], there is a reflection attack if this condition is violated.

4.3 Operating Environment

As discussed above, in order to provide the *key secrecy* and *session authentication* properties, the 4-Way Handshake must be executed in an environment where the formulas listed in Table IV are satisfied. As in the case of TLS, we now relate the protocol invariants to deployment considerations.

$\Gamma_{4way,2}$ and $\Gamma_{4way,3}$ state that if a message containing a keyed hash of a certain format should is sent out, then the principal must have executed certain actions consistent with the 4-Way Handshake. A principal running a protocol provides an oracle that constructs keyed hashes using the same key as the PTK may cause this invariant to fail. On the other hand, most well designed protocols would probably include enough information in the keyed hashes such that the term has a different format.

$\Gamma_{4way,1}$ states the requirement that an honest principal does not execute both the authenticator and the supplicant roles. It expresses this by requiring that a principal that performs actions corresponding to one role does not perform actions corresponding to the other. If $\Gamma_{4way,2}$ does not hold, a simple reflection attack can be demonstrated [He and Mitchell 2005]. It is highly unlikely that this condition would be violated in a wireless LAN environment, as we do not expect a laptop to play the role of an authenticator, or an access point to play the role of a supplicant. However, 802.11i may be deployed in an ad-hoc network environment, in which case it is conceivable that nodes play both roles and violate this assumption.

$\Gamma_{4way,4}$ is precisely what needs to hold for the basic sequences of the protocols in the system for proof of secrecy of ptk to go through. For 802.11i, we can prove $\Gamma_{4way,4}$ by induction over the basic sequences as in the proof of secrecy of the 4Way Handshake. On the other hand, a protocol that has access to the ptk and sends it out in the clear would both cause $\Gamma_{4way,4}$ to fail and break the proof of secrecy.

4.4 Proof of Security of the 4Way Handshake

We now prove Theorem 4.1. We start by proving the secrecy of the term ptk, $\phi_{4Way,sec}$. We then use it to prove the authentication property $\phi_{4Way,auth}$.

We assume that the term pmk is known only to the authenticator X and the supplicant Y, formalized below. This assumption can be discharged by the TLS secrecy property when TLS is used to establish pmk is established by TLS; on the other hand it is a set-up assumption if pmk is a pre-shared secret.

$$\mathsf{Honest}(\hat{Y}) \wedge \mathsf{Honest}(\hat{X}) \wedge \mathsf{Has}(Z, pmk) \supset (\hat{Z} = \hat{X} \vee \hat{Z} = \hat{Y}$$

Intuitively, the proof of the secrecy guarantee is as follows. We first argue that the basic sequences of the 4-Way Handshake do not compromise the secrecy of ptk. This is similar to the proof of secrecy for TLS. The main difference is that unlike the pmk, the ptk is not a nonce, it is a keyed hash of a some nonces and the MAC addresses of the supplicant and the authenticator. The distinctive structure of ptk makes it easier to argue that there are no actions that could possibly compromise its secrecy. The proof of secrecy requires us to introduce new axioms into the proof system, listed in Table V. For instance, the first axiom, **SH0** encodes the non-invertibility of a keyed hash. We now prove that the no basic sequence of the 4way could possibly compromise ptk.

SH0	$\mathsf{Safe}(x, HASH[k](m))$, x is of atomic type
SH1	$\mathsf{Safe}(m'.m'', HASH[k](m)) \equiv \mathsf{Safe}(m', HASH[k](m)) \wedge \mathsf{Safe}(m'', HASH[k](m))$
SH2	$\mathsf{Safe}(SIG[k'](m'), HASH[k](m)) \equiv \mathsf{Safe}(m', HASH[k](m))$
SH3	$\mathsf{Safe}(E_{pk}[k'](m'), HASH[k](m)) \equiv \mathsf{Safe}(m', HASH[k](m))$
SH4	$\mathsf{Safe}(HASH[k'](m'), HASH[k](m)) \equiv k \neq k' \vee m' \neq m$
SH5	$\mathsf{Safe}(E_{sym}[k'](m'), HASH[k](m)) \equiv \mathsf{Safe}(m', HASH[k](m))$
HPOS	$\mathsf{SafeNet}(HASH[k](m)) \wedge \mathsf{Has}(X, HASH[k](m)) \supset \mathsf{Has}(X, k)$

Table V. Axioms used in the proof of secrecy of the 4Way Handshake

Let $[\mathbf{4WAY} : \mathbf{AUTH}_1]_{X'} : [\texttt{new } x'; \texttt{ send } \hat{X}'.\hat{Y}'.x'.\text{“}msg1\text{”};]_{X'}$

$$\mathbf{SH}* \quad [\mathbf{4WAY} : \mathbf{AUTH}_1]_{X'}\ \mathsf{Safe}(\hat{X}'.\hat{Y}'.x'.\text{“}msg1\text{”}, ptk) \tag{1}$$

$$(-1), \quad \mathsf{SafeNet}(ptk)[\mathbf{4WAY} : \mathbf{AUTH}_1]_{X'}\ \mathsf{SendsSafeMsg}(X', ptk) \tag{2}$$

Let $[\mathbf{4WAY} : \mathbf{AUTH}_2]_{X'} : [\texttt{receive } \hat{Y}'.\hat{X}'.y'.\text{“}msg2\text{”}.mic1';$
$ptk' := \texttt{hash } x'.y', pmk';$
$\texttt{verifyhash } mic1', y'.\text{“}msg2\text{”}, ptk';$
$mic2' := \texttt{hash } x'.\text{“}msg3\text{”}, ptk';$
$\texttt{send } \hat{X}'.\hat{Y}'.x'.\text{“}msg3\text{”}.mic2';]_{X'}$

$$\mathbf{SH4} \quad [\mathbf{4WAY} : \mathbf{AUTH}_2]_{X'}\ x'.\text{“}msg3\text{”} \neq x.y \supset \mathsf{Safe}(mic2', ptk) \tag{3}$$

$$\mathbf{SH}*, (-1) \quad [\mathbf{4WAY} : \mathbf{AUTH}_2]_{X'}\ \mathsf{Safe}(\hat{X}'.\hat{Y}'.x'.\text{“}msg3\text{”}.mic2', ptk) \tag{4}$$

$$(-1), \quad \mathsf{SafeNet}(ptk)[\mathbf{4WAY} : \mathbf{AUTH}_2]_{X'}\ \mathsf{SendsSafeMsg}(X', ptk) \tag{5}$$

The first part of the proof argues that the first two basic sequences of the 4Way Authenticator role does not compromise the secret. The first basic sequence clearly does not send out any term that possibly has the structure of the *ptk*. Though the second basic sequences does send out a keyed hash(recall that the *ptk* is a hash keyed by the *pmk*), as the key to the key hash has a structure different from a nonce, we can use **SH4** to argue that the send action is safe. The proof for the remaining basic sequences follows a similar approach.

$$\text{Let } [\mathbf{4WAY} : \mathbf{AUTH}_3]_{X'} : [\texttt{receive } \hat{Y}'.\hat{X}'.\text{“}msg4\text{”}.mic3'; \texttt{verifyhash } mic3', \text{“}msg4\text{”}, ptk';]_{X'}$$

$$\mathsf{SafeNet}(ptk)[\mathbf{4WAY} : \mathbf{AUTH}_3]_{X'}\ \mathsf{SendsSafeMsg}(X', ptk) \tag{6}$$

$$\text{Let } [\mathbf{4WAY} : \mathbf{SUPP}_1]_{Y'} : [\texttt{receive } \hat{X}'.\hat{Y}'.x'.\text{“}msg1\text{”}; \texttt{new } y'; \ ptk' := \texttt{hash } x'.y', pmk'; \ mic1' := \texttt{hash } y'.\text{“}msg2\text{”}, ptk'; \texttt{send } \hat{Y}'.\hat{X}'.y'.\text{“}msg2\text{”}.mic1';]_{Y'}$$

$$\mathbf{SH4} \quad [\mathbf{4WAY} : \mathbf{SUPP}_1]_{Y'}\ y'.\text{“}msg2\text{”} \neq x.y \supset \mathsf{Safe}(mic1', ptk) \tag{7}$$

$$\mathbf{SH}*, (-1) \quad [\mathbf{4WAY} : \mathbf{SUPP}_1]_{Y'}\ \mathsf{Safe}(\hat{Y}'.\hat{X}'.y'.\text{“}msg2\text{”}.mic1', ptk) \tag{8}$$

$$(-1), \quad \mathsf{SafeNet}(ptk)[\mathbf{4WAY} : \mathbf{SUPP}_1]_{Y'}\ \mathsf{SendsSafeMsg}(Y', ptk) \tag{9}$$

$$\text{Let } [\mathbf{4WAY} : \mathbf{SUPP}_2]_{Y'} : [\texttt{receive } \hat{X}'.\hat{Y}'.x'.\text{“}msg3\text{”}.mic2'; \texttt{verifyhash } mic2', x'.\text{“}msg3\text{”}, ptk'; \ mic3' := \texttt{hash } \text{“}msg4\text{”}, ptk'; \texttt{send } \hat{Y}'.\hat{X}'.\text{“}msg4\text{”}.mic3';]_{Y'}$$

$$\mathbf{SH4} \quad [\mathbf{4WAY} : \mathbf{SUPP}_2]_{Y'}\ \text{“}msg4\text{”} \neq x.y \supset \mathsf{Safe}(mic3', ptk) \tag{10}$$

$$\mathbf{SH}*, (-1) \quad [\mathbf{4WAY} : \mathbf{SUPP}_2]_{Y'}\ \mathsf{Safe}(\hat{Y}'.\hat{X}'.\text{“}msg4\text{”}.mic3', ptk) \tag{11}$$

$$(-1), \quad \mathsf{SafeNet}(ptk)[\mathbf{4WAY} : \mathbf{SUPP}_2]_{Y'}\ \mathsf{SendsSafeMsg}(Y', ptk) \tag{12}$$

From 2, 5, 6, 9, 12 and by application of the **NET** rule and the **HPOS** axiom we have the following:

$$\mathsf{Honest}(\hat{Y}) \wedge \mathsf{Honest}(\hat{X}) \wedge \mathsf{Has}(Z, ptk) \supset \mathsf{Has}(Z, pmk)$$

Intuitively the **HPOS** axiom states that if a keyed hash term never appeared on the network, only an agent that knows the key possibly has the term. Using the secrecy of pmk, $\phi_{sec,pmk}$, we conclude the proof of secrecy and have that:

HASH0	$\mathsf{Hash}(X,m,k) \supset \mathsf{Has}(X,m) \wedge \mathsf{Has}(X,k)$
HASH2	$\top[\texttt{verifyhash}\ m', m, k;]_X\ m' = HASH[k](m)$
HASHSRC	$(\mathsf{Send}(X,m)) \wedge \mathsf{Contains}(m, HASH[k](t)) \supset$ $\exists Y.\exists m'.\mathsf{FirstSend}(Y, HASH[k](t), m') \wedge \mathsf{Hash}(Y,t,k)$
FS3	$\mathsf{FirstSend}(Y,t,m) \supset \mathsf{Send}(Y,m) \wedge \mathsf{Contains}(m,t)$

Table VI. Axioms used in the proof of authentication of the 4Way Handshake

$$\mathsf{Honest}(\hat{Y}) \wedge \mathsf{Honest}(\hat{X}) \wedge (\mathsf{Has}(Z, ptk) \supset (\hat{Z} = \hat{X} \vee \hat{Z} = \hat{Y})$$

We now discuss the proof of matching conversations. The first and second steps of the proof asserts that if a principal performs the authenticator role, it must have executed certain actions in a sequence. The third step uses the **HASHSRC** to assert that if a certain keyed hashed term appeared on the network, there must exist some principal who has the key, constructed the hash and sent it out. The fourth step uses the secrecy property to argue that no the only principals that could have sent the keyed hash term are the authenticator and its intended supplicant. The fifth step uses the deployment assumption $\Gamma_{4way,1}$ to argue that this must be the supplicant as the authenticator would never send such a message out. Note that the above proof steps require us to introduce axioms that formalize properties of keyed hashes; these are listed in Table VI.

$$\begin{aligned}
\mathbf{AA1}, \mathbf{P1}, \mathbf{HASH2} \quad & [\mathbf{4WAY} : \mathbf{AUTH}]_X \quad \mathsf{Send}(X, m1) < \mathsf{Receive}(X, m2) \\
& < (\mathsf{Send}(X,) < \mathsf{Receive}(X, m4) \qquad (1) \\
(-1) \quad & [\mathbf{4WAY} : \mathbf{AUTH}]_X \\
& \exists m.\mathsf{Receive}(X,m) \wedge \mathsf{Contains}(m, HASH_{ptk}(\textit{“msg4”})) \\
& \qquad (2) \\
(-1), \mathbf{HASHSRC}, \mathbf{HASH0} \quad & [\mathbf{4WAY} : \mathbf{AUTH}]_X \quad \exists Z.\exists m.\mathsf{Has}(Z, ptk) \\
& \wedge \mathsf{FirstSend}(Z, HASH_{ptk}(\textit{“msg4”}), m) \qquad (3) \\
(-1), \phi_{4way,sec} \quad & [\mathbf{4WAY} : \mathbf{AUTH}]_X \mathsf{Honest}(\hat{X}) \wedge \mathsf{Honest}(\hat{Y}) \supset \\
& \exists Z.\exists m., \quad (\hat{Z} = \hat{X} \vee \hat{Z} = \hat{Y}) \wedge \mathsf{Has}(Z, ptk) \\
& \wedge \mathsf{FirstSend}(Z, HASH_{ptk}(\textit{“msg4”}), m) \qquad (4) \\
(-1), (1), \Gamma_{4way,1}, \mathbf{FS3} \quad & [\mathbf{4WAY} : \mathbf{AUTH}]_X \mathsf{Honest}(\hat{X}) \wedge \mathsf{Honest}(\hat{Y}) \supset \\
& \exists Y.\exists m.\mathsf{FirstSend}(Y, HASH_{ptk}(\textit{“msg4”}), m) \qquad (5)
\end{aligned}$$

We now use the protocol invariant $\Gamma_{4way,2}$ to argue that if the supplicant sent out the term $HASH_{ptk}(\textit{“msg4”})$ it must also have performed actions consistent with the supplicant role.

$$
\begin{aligned}
(-1), \mathbf{FS3}, \Gamma_{4way,3} \quad & [\mathbf{4WAY} : \mathbf{AUTH}]_X \mathsf{Honest}(\hat{X}) \wedge \mathsf{Honest}(\hat{Y}) \supset \\
& \quad \exists Y. \mathsf{FirstSend}(Y, HASH_{ptk}(\textit{“msg4”}), m4) \\
& \wedge \mathsf{FirstSend}(Y, HASH_{ptk}(y, \textit{“msg2”}), m2) \\
& \wedge (\mathsf{Receive}(Y, m1) < \mathsf{Send}(Y, m2) \\
& < \quad \mathsf{Receive}(Y, m3) < \mathsf{Send}(Y, m4)) \qquad (6)
\end{aligned}
$$

To complete the proof we use the freshness of the nonces and certain hashed terms to order the actions of the authenticator with respect to the supplicant.

$$
\begin{aligned}
\mathbf{FS1}, \mathbf{AN3} \quad & [\texttt{new } x; \texttt{send } \hat{X}, \hat{Y}, x, \textit{“msg1”};]_X \\
& \mathsf{FirstSend}(X, x, \hat{X}.\hat{Y}.x.\textit{“msg1”}) \qquad (7) \\
\mathbf{AA1}, \mathbf{P1}, \mathbf{HASH}* \quad & [\mathbf{4WAY} : \mathbf{AUTH}]_X \mathsf{Send}(X, m3) \qquad (8) \\
(-1), \mathbf{HASHSRC}, \mathbf{HASH0} \quad & [\mathbf{4WAY} : \mathbf{AUTH}]_X \exists Z. \exists m. \mathsf{Has}(Z, ptk) \\
& \wedge \mathsf{FirstSend}(Z, HASH_{ptk}(\textit{“msg3”}), m) \qquad (9) \\
(-1), \phi_{4way,sec} \quad & [\mathbf{4WAY} : \mathbf{AUTH}]_X \mathsf{Honest}(\hat{X}) \wedge \mathsf{Honest}(\hat{Y}) \supset \\
& \exists Z. \exists m. \quad (\hat{Z} = \hat{X} \vee \hat{Z} = \hat{Y}) \wedge \mathsf{Has}(Z, ptk) \\
& \wedge \mathsf{FirstSend}(Z, HASH_{ptk}(\textit{“msg3”}), m) \qquad (10) \\
(6), \Gamma_{4way,1} \quad & [\mathbf{4WAY} : \mathbf{AUTH}]_X \mathsf{Honest}(\hat{X}) \wedge \mathsf{Honest}(\hat{Y}) \supset \\
& \exists X'. \exists m. \mathsf{FirstSend}(X, HASH_{ptk}(\textit{“msg3”}), m) \qquad (11) \\
\Gamma_{4way,3}, \text{Inst X' to } X^0 \quad & [\mathbf{4WAY} : \mathbf{AUTH}]_X \mathsf{Honest}(\hat{X}) \wedge \mathsf{Honest}(\hat{Y}) \supset \\
& \mathsf{FirstSend}(X^0, HASH_{ptk}(\textit{“msg3”}), m3) \qquad (12) \\
\Gamma_{4way,3}, \mathbf{AN1} \quad & [\mathbf{4WAY} : \mathbf{AUTH}]_X \mathsf{Honest}(\hat{X}) \wedge \mathsf{Honest}(\hat{Y}) \supset \\
& \mathsf{FirstSend}(X, HASH_{ptk}(\textit{“msg3”}), m3) \qquad (13) \\
(1, 6, 7, 13) \quad & [\mathbf{4WAY} : \mathbf{AUTH}]_X \\
& \mathsf{Honest}(\hat{X}) \wedge \mathsf{Honest}(\hat{Y}) \supset \exists Y. \\
& \quad (\mathsf{Send}(X, m1) < \mathsf{Receive}(Y, m1)) \\
& \wedge (\mathsf{Receive}(Y, m1)) < \mathsf{Send}(Y, m2)) \\
& \wedge (\mathsf{Send}(Y, m2) < \mathsf{Receive}(X, m2)) \\
& \wedge (\mathsf{Receive}(X, m2)) < \mathsf{Send}(X, m3)) \\
& \wedge (\mathsf{Send}(X, m3) < \mathsf{Receive}(Y, m3)) \\
& \wedge (\mathsf{Receive}(Y, m3)) < \mathsf{Send}(Y, m4)) \\
& \wedge (\mathsf{Send}(Y, m4) < \mathsf{Receive}(X, m4)) \\
& \wedge \mathsf{Has}(Y, ptk) \qquad (14)
\end{aligned}
$$

4.5 Improved 4-Way Handshake

The 4-Way Handshake suffers from a DOS vulnerability [He and Mitchell 2004; 2005]. The vulnerability results from the lack of any authentication in $Message\ 1$,

which allows the attacker to block the supplicant role. It is possible to work around this flaw by allowing an arbitrary number of sessions, but this may result in a memory exhaustion attack since the supplicant must store all the nonces that it generates for various sessions. A modification that involves nonce re-use is discussed in [He and Mitchell 2004; 2005] and has been adopted by the 802.11i standards committee. The modified supplicant program is listed in the pseudo-code that follows, while the authenticator program is the same as the one in Table III.

Algorithm 4.1: MOD-4-WAY:SUPP(Y, pmk)

```
new y;
repeat
 receive X̂.Ŷ.z;
 if match z as x.“msg1”;
   then
    match HASH_pmk(x.y) as ptk;
    send Ŷ.X̂.y.“msg2”.HASH_ptk(y.“msg2”);
until
   match z as x.“msg3”.HASH_ptk(x.“msg3”);
send Ŷ.X̂.“msg4”.HASH_ptk(“msg4”);
```

It is easy to see that the modified protocol cannot be blocked by the attacker. Also re-using the nonce until one 4-Way Handshake completes allows the supplicant to avoid storing state, which prevents memory exhaustion. Alternative methods of preventing DoS and a discussion about why the above modification to the 4Way Handshake was the best fix for the DoS attack is discussed [He and Mitchell 2004; 2005]. On the other hand since the fix involves nonce reuse, and nonces are generally used to provide freshness guarantees, it is not obvious that the *authentication* property is preserved under this modification. The main result of this section is the following theorem.

THEOREM 4.2 MODIFIED 4-WAY GUARANTEE.
For the authenticator, the session authentication *and the* key secrecy *guarantees for the modified protocol are identical to Theorem 4.1.*

Intuitively the authenticator guarantee continues to hold because nonce re-use occurs within the *repeat - until loop* of a single session. The formalization of authentication - matching conversations - usually refers to all the actions of a principal in a session. Since an attacker can inject a forged $Message$ 1 an arbitrary number of times, and the supplicant will respond to it. Even so, our guaratee is identical as all we prove is that there are actions of the intended supplicant that match the authenticator's session. The supplicant guarantees are identical to the original 4-Way Handshake.

5. GROUP KEY HANDSHAKE

The authenticator may distribute a Group Temporary Key (GTK) to supplicants for use in multicast applications. The authenticator runs the Group Key Handshake protocol periodically to update the GTK. In this section we prove correctness of the Group Key Handshake.

5.1 Modelling Group Key Handshake

The programs for the Group Key Handshake are listed in Table VII. We briefly describe the authenticator role. Unlike other protocol roles in 802.11i, the authenticator role communicates with multiple peers (supplicants); the role takes as input the list of supplicants and a list of pair-wise keys (PTKs), established by the 4-Way handshake, that the authenticator shares with them. The group key is encrypted under a key encryption key and the messages of the protocol are integrity protected by the a key conformation key, both derived from the PTK; we refer to both keys as *ptk* for expository convenience. The supplicant then confirms receipt of the GTK. The authenticator monotonically increases the sequence number for every key exchange message sent to prevent replay attacks; note that unlike the other protocols, nonces are not used to guarantee freshness. The sequence number comparison `isLess` (a, b) is used by the supplicant to check that $a < b$; if the check fails, the role is aborted.

The authenticator repeatedly generates new group keys and sends them out to all the supplicants in its input list; though a *loop* is a natural way of modeling this, we choose to *unroll* the loop rather than introduce a new construct into the programming language. The proof approach that we detail is extensible to an arbitrary but fixed number of key distributions to a arbitrary but fixed set of supplicants.

5.2 Security Properties

The properties we prove for the Group Key Handshake are listed below.

(1) The Supplicant is assured that the GTK received in the current Group Key Handshake was sent by the Authenticator, and was generated by the Authenticator after the GTK that the supplicant holds from a previous Group Key Handshake or 4-Way Handshake. This is called the *key ordering* property, and is formalized in Definition 5.1.

(2) The Authenticator is assured that the principals with knowledge of the GTK must have executed a 4-Way Handshake with the Authenticator. This is called the *key secrecy* property, and formalized in Definition 5.2.

DEFINITION 5.1 GROUP KEY ORDERING.
For a supplicant $\hat{Y_1}$, the Group Key Handshake is said to provide key ordering if $\phi_{gk,ord}$ holds, where

$$\begin{aligned}&\phi_{gk,ord} ::= \mathsf{Honest}(\hat{X}) \wedge \mathsf{Honest}(\hat{Y}) \supset \exists X.\\&(\mathsf{Send}(X, GrpMessage1) < \mathsf{Receive}(Y, GrpMessage1)) \wedge\\&(\mathsf{Send}(X, GrpMessage1') < \mathsf{Receive}(Y, GrpMessage1')) \wedge\\&(\mathsf{Send}(X, GrpMessage1) < \mathsf{Send}(X, GrpMessage1'))\end{aligned}$$

Here $GrpMessage1'$ and $GrpMessage1$ are abbreviations the messages containing group keys that the supplicant receives.

$$\begin{aligned}GrpMessage1 &:= \hat{X}.\hat{Y}.seqno1.\text{“grp1”}.ENC_{ptk}(gtk1), HASH_{ptk}(seqNo1.\text{“grp2”})\\GrpMessage1' &:= \hat{X}.\hat{Y}.seqno2.\text{“grp1”}.ENC_{ptk}(gtk2), HASH_{ptk}(seqNo2.\text{“grp2”})\end{aligned}$$

$\mathbf{GK} : \mathbf{AUTH} = (X, \hat{Y_1}, \hat{Y_2}, ptk1, ptk2)$ [
$seqNo1 := 0; \texttt{new}\ gtk1;$
$enc11 := \texttt{symenc}\ gtk, ptk1; mic11 := \texttt{hash}\ seqNo1.\text{“}grp1\text{”}.enc11, ptk1;$
$\texttt{send}\ \hat{X}.\hat{Y_1}.seqNo1.\text{“}grp1\text{”}.enc11.mic11;$
$enc21 := \texttt{symenc}\ gtk, ptk2; mic21 := \texttt{hash}\ seqNo1.\text{“}grp1\text{”}.enc21, ptk2;$
$\texttt{send}\ \hat{X}.\hat{Y_2}.seqNo1.\text{“}grp1\text{”}.enc21.mic21;$
$\texttt{receive}\ \hat{Y_1}.\hat{X}.seqNo1.\text{“}grp2\text{”}.mic11'; \texttt{verifyhash}\ mic11', seqNo1.\text{“}grp2\text{”}, ptk1;$
$\texttt{receive}\ \hat{Y_2}.\hat{X}.seqNo1.\text{“}grp2\text{”}.mic21'; \texttt{verifyhash}\ mic21', seqNo1.\text{“}grp2\text{”}, ptk2;$

$seqNo2 = \texttt{inc}\ (seqNo1); \texttt{new}\ gtk2;$
Distribute gtk2 to Y_1 and Y_2 as above
$]_X$

$\mathbf{GK} : \mathbf{SUPP} = (Y, \hat{X}, ptk)$ [
$seqNo1 := 0; \texttt{receive}\ \hat{X}.\hat{Y}.seqno1.\text{“}grp1\text{”}.enc1.mic1;$
$\texttt{verifyhash}\ mic1, seqNo1.\text{“}grp1\text{”}.enc1, ptk;$
$gtk := \texttt{symdec}\ enc1, ptk;$
$mic1' := \texttt{hash}\ seqNo1.\text{“}grp2\text{”}, ptk;$
$\texttt{send}\ \hat{Y}.\hat{X}.seqNo1.\text{“}grp2\text{”}.mic1';$

$\texttt{receive}\ \hat{X}.\hat{Y}.seqno2.\text{“}grp1\text{”}.enc2.mic2;$
$\texttt{verifyhash}\ mic2, seqNo2.\text{“}grp1\text{”}.enc2, ptk;$
$\texttt{isLess}\ seqNo1, seqNo2;$
$gtk2 := \texttt{symdec}\ enc2, ptk;$
$mic2' := \texttt{hash}\ seqNo2.\text{“}grp2\text{”}, ptk;$
$\texttt{send}\ \hat{Y}.\hat{X}.seqNo2.\text{“}grp2\text{”}.mic2';\ ]_Y$

Table VII. Group Key Handshake Programs

DEFINITION 5.2 GROUP KEY SECRECY.
For an authenticator $\hat{X}$, the Group Key Handshake is said to provide key secrecy if $\phi_{gk,sec}$ holds, where

$$\phi_{gk,sec} ::= \mathsf{Honest}(\hat{X}) \wedge \mathsf{Honest}(\hat{Y_1}) \wedge \mathsf{Honest}(\hat{Y_2}) \supset$$
$$\mathsf{Has}(\hat{Z}, gtk1) \supset \hat{Z} = \hat{X} \vee \hat{Z} = \hat{Y_1} \vee \hat{Z} = \hat{Y_2}$$

$\Gamma_{gk,1}$:= $\mathsf{Honest}(\hat{X}) \wedge \mathsf{Send}(X, GrpMessage1') \wedge \mathsf{Send}(X, GrpMessage1)$
$\wedge \mathsf{Contains}(GrpMessage1, seqNo1) \wedge \mathsf{Contains}(GrpMessage1', seqNo1)$
$\wedge \mathsf{IsLess}(seqNo1, seqNo2) \supset \mathsf{Send}(X, GrpMessage1) < \mathsf{Send}(X, GrpMessage1')$

$\Gamma_{gk,2}$:= $\mathsf{Honest}(\hat{X}) \wedge \mathsf{Send}((\hat{X}, \eta), GrpMessage\ 1) \supset \neg\exists\eta'\mathsf{Send}((\hat{X}, \eta'), GrpMessage\ 2)$

$\Gamma_{gk,3}$:= $\mathsf{Honest}(\hat{X}) \wedge \mathsf{Send}(X, m) \wedge \mathsf{Contains}(m, mic1) \supset m = GrpMessage1$

$\Gamma_{gk,4}$ $\forall$basic sequence P of roles in the system.
$\mathsf{SafeNet}(gtk, \{ptk1, ptk2\})\ [P]_W\ \mathsf{Hon}(\hat{X}, \hat{Y_1}, \hat{Y_2}) \wedge \Phi \supset \mathsf{SendsSafeMsg}(W, gtk)$

Table VIII. Group-Key Protocol Precondition and Invariants

Note that $\hat{Y_1}$, $\hat{Y_2}$ is the set of supplicants that the authenticator sends the GTK to. Though the 802.11i standard is not very clear with regards to the supplicant guarantee, two types of properties are conceivable - key freshness and key secrecy. However, key freshness cannot be achieved for the supplicant since a message could be lost; hence, we consider key ordering instead, which is a weaker requirement. Obviously the authenticator knows the key ordering since it generates the keys. Key secrecy can be guaranteed only for the authenticator because the supplicants do not have knowledge of the other supplicants in the group. Theorem 1 states the guarantee for the supplicant and the authenticator.

Theorem 1 Group Key Guarantee.
(i) After execution of the supplicant role, key ordering *is guaranteed if the formulas in Table VIII hold. Formally,*

$$\Gamma_{gk,1} \wedge \Gamma_{gk,2} \wedge \Gamma_{gk,3} \vdash$$
$$[GK{:}SUPP]_Y \phi_{gk,ord}$$

(ii) After execution of the authenticator role, key secrecy *is guaranteed if the formulas in Table VIII hold. Formally,*

$$\Gamma_{gk,1} \wedge \Gamma_{gk,2} \wedge \Gamma_{gk,3} \vdash$$
$$[GK{:}AUTH]_X \phi_{gk,sec}$$

(iii) $\Gamma_{gk,1}$ *and* $\Gamma_{gk,2}$ *are invariants of the Group Key Handshake;* $\Gamma_{gk,3}$ *is an assumption on the environment. Formally,* $GK \vdash \Gamma_{gk,1} \wedge \Gamma_{gk,2}$

5.3 Operating Environment

In this section, we discuss the relationship between deployment scerios and the protocol invariants needed for the proof of security of the Group Key Handshake. Table VIII lists the protocol invariants. As in Sections 4.3, 3.4, we regard these formulas as specifications for a safe operating environment.

$\Gamma_{gk,1}$ states that the authenticator should increase the sequence counter monotonically to guarantee the key ordering property for the supplicants. As pointed out in [He and Mitchell 2004], the sequence numbers play an important role in the Group Key Handshake, but are redundant in the 4-Way Handshake. We need the axioms listed in Table IX to establish this invariant.

SQ1	$[b = \texttt{inc}\ \ a]\quad \mathsf{IsLess}(a, b)$
SQ2	$\mathsf{IsLess}(a, b) \wedge \mathsf{IsLess}(b, c) \supset \mathsf{IsLess}(a, c)$
SQ3	$\mathsf{IsLess}(a, b) \wedge \neg\mathsf{IsLess}(b, a)$
SQ4	$\neg\mathsf{IsLess}(a, a)$

Table IX. Axioms related to Sequence Numbers

Like the 4-Way Handshake, there is a restriction on a principal to not play both the authenticator and the supplicant roles. This is represented by the formula $\Gamma_{gk,2}$. This is an assumption about the operating environment which should be carefully considered in implementations.

5.4 Proof of Security of the Group Key Handshake

We now prove the key ordering property for the supplicant. Note that the proof assumes that the *ptk* is secret and known only the the supplicant and its intended authenticator. This can be discharged by 4Way secrecy proof. As in the proof of authentication of the 4Way Handshake protocol, the first five steps use the **HASHSRC** axiom, the secrecy of the *ptk* and the deployment assumption that no principal plays the role of both authenticator and supplicant to argue that the intended authenticator must indeed have generated and sent out the first group key received by the supplicant. To conclude we order the send of the authenticator with the receive of the supplicant.

$$\mathbf{AA1}, \mathbf{P1}, \mathbf{HASH2} \quad [\mathbf{GK} : \mathbf{SUPP}]_Y \quad \mathsf{Receive}(Y, GrpMessage1) \tag{1}$$

$$(-1) \quad [\mathbf{GK} : \mathbf{SUPP}]_Y \exists m.\mathsf{Receive}(Y, m) \wedge \mathsf{Contains}(m, mic1) \tag{2}$$

$$(-1), \mathbf{HASHSRC}, \mathbf{HASH0} \quad [\mathbf{GK} : \mathbf{SUPP}]_Y \quad \exists Z.\exists m.\mathsf{Has}(Z, ptk) \wedge \mathsf{FirstSend}(Z, mic1, m) \tag{3}$$

$$(-1), \phi_{4way,sec} \quad [\mathbf{GK} : \mathbf{SUPP}]_Y \mathsf{Honest}(\hat{X}) \wedge \mathsf{Honest}(\hat{Y}) \supset \exists Z.\exists m.,$$
$$(\hat{Z} = \hat{X} \vee \hat{Z} = \hat{Y_1}) \wedge \mathsf{Has}(Z, ptk) \wedge \mathsf{FirstSend}(Z, mic1, m) \tag{4}$$

$$(-1), (1), \Gamma_{gk,2}, \mathbf{FS3} \quad [\mathbf{GK} : \mathbf{SUPP}]_Y \mathsf{Honest}(\hat{X}) \wedge \mathsf{Honest}(\hat{Y}) \supset$$
$$\exists X.\exists m.\mathsf{FirstSend}(X, mic1, m) \tag{5}$$

$$(-1), \mathbf{FS2}, \Gamma_{gk,3} \quad [\mathbf{GK} : \mathbf{SUPP}]_Y \mathsf{Honest}(\hat{X}) \wedge \mathsf{Honest}(\hat{Y}) \supset$$
$$\exists X.\mathsf{Send}(X, Grpmessage1) < \mathsf{Receive}(Y, GrpMessage1) \tag{6}$$

$$(-1), \text{Inst } X \text{ to } X^0 \quad [\mathbf{GK} : \mathbf{SUPP}]_Y \mathsf{Honest}(\hat{X}) \wedge \mathsf{Honest}(\hat{Y}) \supset$$
$$\mathsf{Send}(X^0, Grpmessage1) < \mathsf{Receive}(Y, GrpMessage1) \tag{7}$$

By a similar argument we now establish a similar guarantee for the second GTK received by the supplicant.

$$[\mathbf{GK} : \mathbf{SUPP}]_Y \mathsf{Honest}(\hat{X}, \hat{Y}) \supset Send(X', GrpMessage1') < \mathsf{Receive}(Y, GrpMessage1') \tag{8}$$

It remains to assert that the authenticator distributed the two gtks in order of their associated sequence numbers. This relies on the authenticators incrementing the sequence number between two successive key distributions. We need a technical assumpstion that that the authenticator does not use the same ptk in two different threads. This is a set-up assumption that we do not discharge.

$$\mathsf{Send}((\hat{X}, \eta), GrpMessage1) \wedge \mathsf{Send}((\hat{X}, \eta'), GrpMessage1') \supset \eta = \eta' \tag{9}$$

The authentication property $\phi_{gk,auth}$ follows easily from $7, 8, 9$ and the invariant $\Gamma_{gk,1}$, which says that an authenticator thread always uses sequence numbers in an ascending manner.

We now prove the secrecy property of Group Key protocol. Without loss of generality we focus on one specific gtk. As in the proof of TLS secrecy, we perform induction on the basic sequences of various protocol roles to show that honest principals do not perform actions that compromise the secrecy of the term gtk. We prove:

$$\Phi \wedge \mathsf{Honest}(X) \wedge \wedge_i \mathsf{Honest}(Y_i) \supset \mathsf{SafeNet}(gtk, \mathcal{K})$$

where,

$$\mathcal{K} = \{ptk_1, ptk_2, \cdots, ptk_n\}$$
$$\Phi \equiv \mathsf{SymEnc}(Z, gtk, ptk') \wedge \mathsf{New}(Z, gtk) \supset ptk' \in \mathcal{K}$$

The proof of secrecy is general, we assume that the authenticator only sent out the GTK to one of these n supplicants that it shares a PTK with. Formally, each induction step asserts that if, at the beginning of the basic sequence, the term gtk has not appeared on the network either in the clear of encrypted under the key $\bar{k_y}$ ($\mathsf{SafeNet}(gtk, \mathcal{K})$), then a basic sequence executed by a thread Z does not perform any actions that compromise the secrecy of the term secret($\mathsf{SendSafeMsg}(Z, gtk, \mathcal{K})$). We start with the basic sequences of the supplicant. W.l.o.g. we fix an arbitrary basic sequence of the supplicant thread, as all basic sequences of the supplicant are similar; they receive a new gtk, check that the new sequence number is higher than the current sequence number and then install the new gtk. It is easy to show that the basic sequence does not violate secrecy as the only term it sends is a keyed hash term.

$$\text{Let } [\mathbf{GK} : \mathbf{SUPP}_i]_{Y'} : [\texttt{receive } \hat{X}'.\hat{Y}'.NewSeqNo'.\text{“}grp1\text{”}.encg'.micg1';$$
$$\texttt{verifyhash } micg1', NewSeqNo'.\text{“}grp1\text{”}.encg', ptk';$$
$$gtk' := \texttt{symdec } encg', ptk';$$
$$\texttt{isLess } OldSeqNo', NewSeqNo';$$
$$micg2' := \texttt{hash } NewSeqNo'.\text{“}grp2\text{”}, ptk';$$
$$\texttt{send } \hat{Y}'.\hat{X}'.NewSeqNo'.\text{“}grp2\text{”}.micg2';]_{Y'}$$

$$\mathbf{NET}* \quad \mathsf{SafeNet}(gtk, \mathcal{K})[\mathbf{GK} : \mathbf{SUPP}_i]_{X'} \mathsf{SendsSafeMsg}(X', gtk, \mathcal{K}) \tag{10}$$

$$(18, 19, 10, \cdots), \mathbf{NET} \quad \Phi \supset \mathsf{SafeNet}(gtk, \mathcal{K}) \tag{11}$$

Though the authenticator has several basic sequences, there are only two basic types. First, we have the basic sequence that generates and sends a new GTK. We use the formula Φ to show that such a basic sequence does not compromise the secrecy of the term gtk.

$$\text{Let } [\mathbf{GK} : \mathbf{AUTH}_1]_{X'} : [NewSeqNo' := \texttt{inc } CurrSeqNo';$$
$$encg' := \texttt{symenc } gtk', ptk';$$
$$micg1' := \texttt{hash } NewSeqNo'.\text{“}grp1\text{”}.encg', ptk';$$
$$\texttt{send } \hat{X}'.\hat{Y}'.NewSeqNo'.\text{“}grp1\text{”}.encg'.micg1';]_{X'}$$

$$\text{Case } 1 : gtk' \neq gtk \tag{12}$$
$$\mathbf{SAF}* \quad [\mathbf{GK} : \mathbf{AUTH}_1]_{X'} \mathsf{SafeMsg}(\hat{X}'.\hat{Y}'.NewSeqNo'.\text{“}grp1\text{”}.encg'.micg1', gtk, \mathcal{K}) \tag{13}$$

$$\text{Case } 2 : gtk' = gtk \tag{14}$$
$$\mathbf{AA1} \quad [\mathbf{GK} : \mathbf{AUTH}_1]_{X'} \mathsf{SymEnc}(X', gtk, ptk') \tag{15}$$
$$\Phi, (-1) \quad ptk' \in \mathcal{K} \tag{16}$$
$$\mathbf{SAF}*, (-1) \quad [\mathbf{GK} : \mathbf{AUTH}_1]_{X'} \mathsf{SafeMsg}(\hat{X}'.\hat{Y}'.NewSeqNo'.\text{“}grp1\text{”}.encg'.micg1', gtk, \mathcal{K}) \tag{17}$$
$$(-5, -1), \mathbf{NET}* \quad \mathsf{SafeNet}(gtk, \mathcal{K})[\mathbf{GK} : \mathbf{AUTH}_1]_{X'} \mathsf{SendsSafeMsg}(X', gtk, \mathcal{K}) \tag{18}$$

Finally we have the basic sequence that receives the acknowledgement from the supplicant and checks its integrity. This case is easy as there are no send actions.

$$\text{Let } [\mathbf{GK} : \mathbf{AUTH}_2]_{X'} : [\texttt{receive } \hat{Y}'.\hat{X}'.NewSeqNo'.\text{“}grp2\text{”}.micg2';$$
$$\texttt{verifyhash } micg2', NewSeqNo'.\text{“}grp2\text{”}, ptk';]_{X'}$$

$$\mathbf{NET}* \quad \mathsf{SafeNet}(gtk, \mathcal{K})[\mathbf{GK} : \mathbf{AUTH}_2]_{X'} \mathsf{SendsSafeMsg}(X', gtk, \mathcal{K}) \tag{19}$$

This concludes the proof of the secrecy property.

6. COMPOSITION

802.11i consists of multiple protocols,TLS, 4-Way Handshake, Group Key Handshake, that share keying material. Thus far we have proved that each of these protocols are secure in isolation. When protocols share keying material, protoocols that are secure in isolation may no longer be secure when they are concurrently in

the same environment; Section 6.1 discusses an early version of TLS that illustrates this. Further, 802.11i provides an error-handling strategy that reacts to failures and timeouts by restarting execution from the beginning. Furthermore, more efficient error-handling strategies have also been proposed (see Figure 1) to improve efficiency. It is not clear that the individual protocol components are secure with such restarts. Section 6.1 discusses the ISO-9798-3 protocol which has such a flaw. Finally, 802.11i allows different modes of operation which use Pre-Shared Keys or reuse previously cached PMK's; this is similar to restarting the protocol with the same keying material.

In this section, we prove that the various components of 802.11i compose safely. Section 6.1 discusses flaws in protocols that have occurred in the past and discusses why 802.11i may possibly suffer from similar flaws, which inspires our analysis. Section 6.2 is a technical section that introduces a new notion of composition, called staged composition, needed to tackle the 802.11i error handling strategy. Section 6.3 applies staged composition to model 802.11i and proves thatvarious components of 802.11i compose safely, for a general class of error-handling mechanisms, and for different operation modes mentioned above.

6.1 Protocol Interactions

In this section, we discuss some flaws due to protocol interactions and complicated control flows. Compositional flaws are protocol flaws that cannot be discovered when a protocol is analyzed in isolation. Such flaws have been discovered in published protocol systems in the past [Meadows and Pavlovic 2004; J. Benaloh and Yee. 1995; F. J. Thayer and Guttman. 1999]. For instance, the authors of [J. Benaloh and Yee. 1995; Meadows 2001] discuss a flaw in the previous versions of SSL, that results from the confusion of authenticators in the main and the resumption protocols. In SSL, client authentication is achieved by the client signing certain terms. However, in the flawed SSL version, the authenticator does not include the negotiated cipher suite or indicates whether the undergoing protocol is the main protocol or the resumption protocol. This leads to an attack on the main protocol, which uses the resumption protocol as a signature oracle. As a result, an intruder who can break a weak cipher, is able to gain access to keys that the server believes that it has established with the client using strong cryptography. This attack will be captured during the analysis. In Section 6.3 we show that the invariants used to establish properties of protocols in isolation are preserved by *all* the protocols in the system. Proofs that involve signature based authentication involve invariants are generally of the form $\phi \supset \psi$, where ϕ refers to the action of sending a signature term, and ψ refers to having performed a specific set of actions corresponding to the protocol being analyzed; for instance consider the invariant $\Gamma_{tls,1}$ in Table II). In the case of SSL 2.0, it is impossible to establish such an invariant, which discloses the flaw. The flaw also demonstrates that when protocols share keying material, it is important to analyze their interactions. Fortunately, in the subsequent SSL 3.0, the signature is over the entire conversation history, which includes adequate version information. Though we do not consider the resumption protocol, this difference in structures in the main and resumption protocols to have different structures would allow us to prove that the invariant is satisfied by the resumption protocol.

Next, we consider an instance of a protocol where non-linear control flows may

introduce vulnerabilities in an otherwise secure system. In [Datta et al. 2004b], the authors construct the ISO-9798-3 protocol by sequentially composing a protocol-DH_0 which generates a Diffie-Hellman exponent and a protocol-CR which does signature-based challenge response. Each role of the signature-based challenge response takes as input a nonce that needs to be fresh. Assuming that prior to the execution of the protocol the precondition of nonce freshness is satisfied, we can prove that on completion of the protocol, the peer is authenticated. Since the Diffie-Hellman exponent generated by the first protocol is fresh, the sequential composition theorem allows us to prove mutual authentication for the ISO-9798-3 protocol. Now consider a situation with restarts. This corresponds to allowing the reuse of the Diffie-Hellman exponent multiple times. As we might expect, this results in a replay attack. Since the Diffie-Hellman exponent is no longer fresh once it has been sent out, it is clear that freshness of the nonce is not preserved by roles of the CR protocol, which includes these terms in its messages.

IEEE 802.11i is an instance of the system that one protocol establishes keys and subsequent protocols use the key, just like the situation in SSL with the main and resumptions protocols discussed above. Further, there are proposals [He and Mitchell 2005] that suggest error handling strategies causing complex control flows. It is possible that these error handling flows cause 802.11i not to be secure, just as in the case of ISO-9798-3. Therefore, a compositional analysis of 802.11i is important.

6.2 Staged Composition

The sequential composition theorem [Datta et al. 2004c; 2004b] allows us to combine the proofs of sub-protocols into a proof of a composed protocol. However, sequential composition only works for control flows where the components execute one after the other in sequence. An examination of the control flow graphs in Figure 1 indicates that the intended sequence of execution of the sub-protocols forms a chain and the error-handling strategy introduces a set of backward arcs. This makes the sequential composition theorem stated in [Datta et al. 2004c; 2004b] and the notion of sequential composition inadequate for our purpose. We now introduce the notion of Staged Composition that extends sequential composition to handle control flow graphs with an arbitrary set of backward edges.

Recall that a protocol is a set of roles. For instance, the 4-Way Handshake consists of two roles – the authenticator and the supplicant. Each role is a sequence of protocol steps (`send` , `receive` , `new` and `verify` actions), possibly depending on an input parameter list and providing an output parameter list. In a correct execution, we expect the role to run to completion. However, in reality the execution might be interrupted due to unexpected failures. Therefore, we partition a role into atomic steps, during each of which the execution does not block and cannot be interrupted. As only `receive` actions require one thread to wait for another, roles are broken into atomic steps at receive points.[1] The following definition captures the intuition that a protocol role may terminate at the end of any of its atomic protocol steps.

[1]Stricly speaking, roles could also terminate when `verify` acitons do not succeed. In such cases we can assume that the receive action that preceeds the verify did not occur.

Definition 6.1 Strand Prefix.
The set of prefix strands of a strand $[R]_X$, $Pref([R]_X) := \bigcup_{0\leq i\leq k} \{[b_0 \dots b_i]_X\} \bigcup\{[]_X\} \bigcup\{[R]_X\}$, *where strand* $R = [b_0 \dots b_k]_X$.

We next define an operation on roles called *box* composition, which models a complete execution of the first role followed by several incomplete executions of the second role. The definition makes precise the substitutions involved for an incomplete executions; in particular each execution of the second role effectively ignores all previous incomplete executions. We also define box composition of *sets* of roles in terms of box composition of roles.

Definition 6.2.
Given two roles R_1 *and* R_2, $R_1 = (\overrightarrow{x_1})[r_1]_X\langle\overrightarrow{y_1}\rangle$ *and* $R_2 = (\overrightarrow{x_2})[r_2]_X\langle\overrightarrow{y_2}\rangle$. *Their box composition* A *is defined as follows. A role* $R \in A$ *if and only if* $R = (\overrightarrow{x_1})[s_1; s_2; \dots s_k]\langle\overrightarrow{y}\rangle$, *and there exists a* σ *such that* $\sigma\overrightarrow{y_1} = \overrightarrow{x_2}$.

(1) $s_1 = r_1$
(2) $i > 1$, $s_i \in Pref(\sigma r_2)$,
(3) $y = \sigma\overrightarrow{y_2}$ *if* $s_k = \sigma r_2$, *else* $y = \{\}$.

Definition 6.3.
The box composition A *of two sets of roles* B *and* C, *is defined as follows. A role* $R \in A$ *if and only if* $R \in R_1 \Box R_2$, *where* $R_1 \in B$, $R_2 \in C$.

A role in the composed protocol corresponds to a possible execution path in the control flow graph. A backward edge in a control flow graph from role R_i to role R_j causes control to proceed from the end of an atomic protocol step of R_i to the beginning of R_j. The following definition captures the set of possible executions of a role in a composed protocol.

Definition 6.4 Staged Role.
The staged composition of a sequence of roles $\langle R_1, R_2 \dots R_k\rangle$ *is the set of roles,* $RComp(\langle R_1, R_2 \dots R_k\rangle)$, *defined in terms of box composition as:* $[]_X \Box (R_1 \Box (R_2 \Box (\dots R_k)))$.

Given a chain of roles $\langle R_1, R_2 \dots R_n\rangle$, execution always starts at R_1; progress down the chain happens on successful completion of a role, with error-handling causing control to flow to the beginning of an earlier block in the chain. Failure of a role R_{ak} can happen at multiple points as characerized by the set $Pref(R_{a_k})$ defined above. Box composition implies that staged roles are obtained by substitutions as follows: Variables in the input parameter list of a role-instance R_{a_i} are renamed by corresponding variables in the output parameter list of the *last, complete* execution of R_{a_i-1} that occurs prior to R_{a_i} in the list: $R_{a_1}; R_{a_2}; \dots; R_{a_l}$. Note that such subsitutions are consistent with the use-defintion behavior that non-linear control flows cause in real-world protocol systems. Consider two protocols: Protocol A establishes a shared secret, and Protocol B uses this shared secret as a symmetric key to protect data transfers. Principals may periodically re-execute Protocol A to get a fresh key. Any instance of Protocol B would use the shared key established by the latest completed run of Protocol A.

To prove that formulas are invariants of staged roles, we use the following variant of the **Honesty** rule(see Section 2, Appendix A). The following rule is intended to

be applied to protocol systems where honest principals execute exactly a staged role in a single role composition $RComp(\langle R_1 \ldots R_n \rangle)$. Note that this staged composition induces a natural order on the the basic sequences of the staged role; this is precisely the sequence of execution of the basic sequences in a run where the role runs to completion without any restarts. We index the basic sequences in this order.

$$\frac{\begin{array}{l}\mathsf{Start}(X)\ []_X\ \phi \wedge \theta_0 \\ \forall i \geq 0.\ \phi \wedge \theta_{P_i}\ [P_{i+1}]_X\ \phi \wedge \theta_{P_{i+1}} \\ \forall i, j, j > i.\ \theta_{P_j} \supset \theta_{P_i}\end{array}}{\mathsf{Honest}(\hat{X}) \supset \phi}$$

Informally the rule states that if three sets of conditions are satified, the formula ϕ holds at every point in the execution of the system. First, the formula ϕ must be provable at start of the thread; Second, for every basic sequence P_i of the role, if ϕ is true at the start of start of the basic sequence and a certain pre-condition $\theta_{P_{i-1}}$ holds, we can show that ϕ and a certain post-condition θ_{P_i} hold at the end of the execution of the basic sequence. The third set of conditions are motivated by the restarts in execution of the staged role, it asserts a sufficient condition that implies the precondition $\theta_{P_{i-1}}$ whenever P_i is executed. We omit the proof of soundness of the rule; it follows from a modification of the proof of soundness the standard honesty rule.

6.3 Composition of 802.11i

We now describe our model of 802.11i using combination of parallel composition and staged composition. A cord of 802.11i is either in $RComp(\langle TLS : Client, 4Way : Supplicant \rangle)$ or in $RComp(\langle TLS : Server, 4Way : Authenticator \rangle)$ or one of the $Groupkey : Authenticator$ or $Groupkey : Supplicant$ roles.

Note that our treatment of the Group Key protocol is not uniform: The authenticator role of the group key protocol communicates with multiple supplicants using PTKs established by several runs of the 4Way Handshake one for each supplicant. This makes it hard to define an 802.11i role as a staged composition of one role each of the TLS, 4Way Handshake and the Group Key Handshake protocols. We would have to define a variant of staged composition composition that binds several 4Way Handshake threads to a single Group Key thread; we do not see a straightforward way of doing this. Further, we will soon observe that allowing the Group key protocol to restart breaks its key ordering property.

We now state the main theorem of this paper. The theorem has two parts that reflect the assume guarantee approach. The first part asserts that 802.11i satisfies all the invariants used in proofs of security of TLS, 4Way and the Group Key protocols. The second part asserts that if all these invariants are satisfied by the protocols in the system, we have security of the TLS, 4Way and the Group Key Handshake protocols; the precise security properties are those listed in Sections 3, Sections 4 and Sections 5.

Theorem 6.5 802.11i Guarantee.

(i) Suppose no principal plays the role of both supplicant and authenticator, then 802.11i satisfies $\Gamma_{4way,1}, \Gamma_{gk,2}$.
Further, $802.11i \vdash \Gamma_{tls,1}, \Gamma_{tls,2}, \Gamma_{tls,3}, \Gamma_{4way,2}, \Gamma_{4way,3}, \Gamma_{4way,4}, \Gamma_{gk,1}, \Gamma_{gk,3}, \Gamma_{gk,4}$
Thus 802.11i satisfies assumptions in Table IV, II, VIII.
(ii) The security properties of the components listed in sections 4.2, 3.2, 5.2 are guaranteed in all modes of IEEE 802.11i, if the assumptions in Table IV, II, VIII are satisfied.

Statements (i) and (ii) together imply that 802.11i satisfies the properties listed in the standard. Additionally, using Pre-Shared Keys or cached PMKs is also secure.

We now prove the theorem. Note that if the first part of the theorem is true, the second part is an easy consequence of part (i) of Theorem 3.1, part (i) of Theorem 4.1, part (i),(ii) of Theorem 1, which state that the relevant security properties are true if the required invariants hold. We now focus on proving the first part of the theorem.

$802.11i \vdash \Gamma_{4way,1}, \Gamma_{gk,2}$ are assumptions on the deployment scenario that no principal plays both the authenticator and the supplicant roles, we do not have to discharge these assumptions by proof.

We now consider the invariants $802.11i \vdash \Gamma_{tls,1}, \Gamma_{tls,2}, \Gamma_{4way,2}, \Gamma_{4way,3}, \Gamma_{gk,3}$. All these invariants are of the form $\phi \supset \psi$, where ϕ refers to the action of sending a or a keyed hash term, and ψ refers to having performed a specific set of actions corresponding to the protocol being analyzed. The proofs for the various formule are similar.

We now describe the proof process. We first prove that the above formulae are invariants of each role in the system. We establish these invariants by induction on the basic sequences and by application of an honesty rule. For the staged roles, we use the honesty rule from the previous section. Recall that 802.11i is a parallel composition of four roles, two of which are staged roles. We use the *parallel composition theorem* from [Datta et al.] to show that the formulae are invariants of 802.11i.

All the proofs are similar and fairly mechanical; we list part of the proof of $\Gamma_{tls,2}$ as a representative example. Recall the formula $\Gamma_{tls,2} \equiv \quad \mathsf{Sign}(X, sigterm) \supset \mathsf{New}(X, secret) \wedge (\mathsf{PkEnc}(X, secret, \hat{Z}) \supset \hat{Z} = \hat{Y})$
$\wedge (\mathsf{Send}(X, m) \supset \neg\mathsf{ContainsOpen}(m, secret))$. We use the honesty rule from the previous section to show that for any role R in $RComp(\langle TLS : Client, 4Way : Supplicant\rangle)$, $R \vdash \Gamma_{tls,2}$. Recall that the we need to establish three sets of conditions before we can apply the honesty rule. It is easy to show that $\mathsf{Start}(X)\ []_X\ \Gamma_{tls,2}$, using the axiom **AA2**. We omit the easy proofs that establish(part of the second antecedent) and preserve the post conditions (the third set of conditions). We focus on showing that if $\Gamma_{tls,2}$ is true before a basic sequence starts, it is also true at the end of the basic sequence. We now prove that $\Gamma_{tls,2} \wedge \theta_{P_{i-1}}\ [P_i]_X\ \Gamma_{tls,2}$ for the basic sequences of the TLS client role. For the proof that follows, define $\theta_{TLS:Client_0} \equiv \theta_{TLS:Client_1} \equiv \theta_{TLS:Client_2} \equiv \theta_{TLS:Client_3} \equiv \top$; Though the induction steps sometimes require certain preconditions to hold, we do not need any for this proof. Note that the first basic sequence of TLS generates and sends out a nonce n_x. The proof has two easy cases; first, if the thread previously signed a term with the structure of $sigterm$, then as the invariant is initially satisfied, the

relevant nonce must already have been generated and therefore cannot be the nonce n_x, which is generated within the basic sequence.

Sequence: $[\mathbf{TLS} : \mathbf{Client_1}']_X$

Case: $\mathsf{Sign}(X, sigterm)$

$\Gamma_{tls,2}$ $\mathsf{New}(X, secret)$ (1)

$\mathsf{New}(X, secret)\ [\mathbf{TLS} : \mathbf{Client_1}']_X\ n'_x \neq secret$ (2)

(-1) $\mathsf{New}(X, secret)\ [\mathbf{TLS} : \mathbf{Client_1}']_X\ \neg\mathsf{ContainsOpen}(\hat{X}.\hat{Y}.n'_x.V_x, secret)$ (3)

$(-1), \mathbf{AA3}$ $\Gamma_{tls,2} \wedge \theta_{TLS:Client_0} \wedge \mathsf{Sign}(X, sigterm)\ [\mathbf{TLS} : \mathbf{Client_1}']_X\ \Gamma_{tls,2}$ (4)

Case: $\neg\mathsf{Sign}(X, sigterm)$

AA3 $\neg\mathsf{Sign}(X, sigterm)\ [\mathbf{TLS} : \mathbf{Client_1}']_X\ \neg\mathsf{Sign}(X, sigterm)$ (5)

(-1) $\Gamma_{tls,2} \wedge \theta_{TLS:Client_0} \wedge \neg\mathsf{Sign}(X, sigterm)\ [\mathbf{TLS} : \mathbf{Client_1}']_X\ \Gamma_{tls,2}$ (6)

We now list the proof for the other basic sequences of the TLS client role.

Sequence: $[\mathbf{TLS} : \mathbf{Client_2}']_X$

Case: $\mathsf{Sign}(X, sigterm)$

$\Gamma_{tls,2}$ $\mathsf{New}(X, secret)$ (7)

$\mathsf{New}(X, secret)\ [\mathbf{TLS} : \mathbf{Client_2}']_X\ secret' \neq secret$ (8)

(-1) $\mathsf{New}(X, secret)\ [\mathbf{TLS} : \mathbf{Client_2}']_X$
$\neg\mathsf{ContainsOpen}(\hat{X}.\hat{Y}.encky'.sigvx'.hc2', secret)$ (9)

$(-1), \mathbf{AA3}$ $\Gamma_{tls,2} \wedge \theta_{TLS:Client_1} \wedge \mathsf{Sign}(X, sigterm)\ [\mathbf{TLS} : \mathbf{Client_1}']_X\ \Gamma_{tls,2}$ (10)

Case: $\neg\mathsf{Sign}(X, sigterm)$

AA3 $\neg\mathsf{Sign}(X, sigterm)\ [\mathbf{TLS} : \mathbf{Client_2}']_X\ sigterm' \neq sigterm \supset$
$\neg\mathsf{Sign}(X, sigterm)$ (11)

$\neg\mathsf{Sign}(X, sigterm)\ [\mathbf{TLS} : \mathbf{Client_2}']_X\ sigterm' = sigterm \supset$
$\mathsf{Sign}(X, sigterm) \wedge \mathsf{New}(X, secret) \wedge \mathsf{PkEnc}(X, secret, \hat{Y}) \wedge$
$(\mathsf{Send}(X, m) \supset \neg\mathsf{ContainsOpen}(m, secret))$ (12)

(-1) $\Gamma_{tls,2} \wedge \theta_{TLS:Client_1} \wedge \neg\mathsf{Sign}(X, sigterm)\ [\mathbf{TLS} : \mathbf{Client_2}']_X\ \Gamma_{tls,2}$ (13)

Sequence: $[\mathbf{TLS} : \mathbf{Client_3}']_X$

$$\mathbf{AA3} \quad \Gamma_{tls,2} \wedge \theta_{TLS:Client_2}\ [\mathbf{TLS} : \mathbf{Client_3}']_X\ \Gamma_{tls,2} \wedge \theta_{TLS:Client_3} \tag{14}$$

We now prove that $\Gamma_{tls,2} \wedge \theta_{P_{i-1}}\ [P_i]_X\ \Gamma_{tls,2}$ for the basic sequences of the 4Way Supplicant role. Both basic sequences of the supplicant send out a nonce n_x; for the second basic sequence, we need a non-trivial, but easy to establish precondition. We define $\theta_{4Way:Supplicant_0} \equiv \theta_{4Way:Supplicant_2} \equiv \top$, $\theta_{4Way:Supplicant_1} \equiv \neg(Sign(X, sigterm) \wedge secret = n'_x)$

$$\mathbf{AN4} \quad \mathsf{New}(X, x)[\texttt{new}\ n;]_X \quad n \neq x$$

Sequence: $[\mathbf{4Way} : \mathbf{Supplicant_1}']_X$

Case: $\mathsf{Sign}(X, sigterm)$

$(-1), \Gamma_{tls,2}$ $\mathsf{New}(X, secret)$

$(-1), \mathbf{AN4}$ $[\ldots \texttt{new}\ n'_x; \ldots]_X\ secret \neq n'_x$

(-1) $[\mathbf{4Way} : \mathbf{Supplicant_1}']_X\ \neg\mathsf{ContainsOpen}(m2, secret)$ (15)

Case: $\neg\mathsf{Sign}(X, sigterm)$

$\mathbf{AA3}$ $\neg\mathsf{Sign}(X, sigterm)\ [\mathbf{4Way} : \mathbf{Supplicant_1}']_X\ \neg\mathsf{Sign}(X, sigterm)$ (16)

$(15), (16)$ $\Gamma_{tls,2} \wedge \theta_{4Way:Supplicant_0}\ [\mathbf{4Way} : \mathbf{Supplicant_1}']_X\ \Gamma_{tls,2}$ (17)

Sequence: $\theta_{4Way:Supplicant_1}[\mathbf{4Way} : \mathbf{Supplicant_2}']_X$

Case: $\mathsf{Sign}(X, sigterm)$

$(-1),$ $\theta_{4Way:Supplicant_1}[\mathbf{4Way} : \mathbf{Supplicant_2}']_X\ secret \neq n'_x$

(-1) $\theta_{4Way:Supplicant_1}[\mathbf{4Way} : \mathbf{Supplicant_2}']_X\ \neg\mathsf{ContainsOpen}(m4, secret)$ (18)

Case: $\neg\mathsf{Sign}(X, sigterm)$

$\mathbf{AA3}$ $\neg\mathsf{Sign}(X, sigterm)\ [\mathbf{4Way} : \mathbf{Supplicant_2}']_X\ \neg\mathsf{Sign}(X, sigterm)$ (19)

$(15), (16)$ $\Gamma_{tls,2} \wedge \theta_{4Way:Supplicant_1}\ [\mathbf{4Way} : \mathbf{Supplicant_2}']_X\ \Gamma_{tls,2}$ (20)

We now discuss the remaining invariants briefly. The secrecy proofs can be modified to show that $802.11i \vdash \Gamma_{tls,3}, \Gamma_{4way,4}, \Gamma_{gk,4}$. The proof of $802.11i \vdash \Gamma_{gk,1}$ is easy to establish but requires the following additional axioms about sequence

numbers(Table IX). Further, note that the this invariant would not hold if the Group key protocol were allowed to restart; clearly the reuse of sequence numbers breaks the key ordering property of the Group Key protocol.

Furthermore, our proof implies the correctness of other deployment modes in 802.11i, which are different from that in section 2.1. First, 802.11i may be run without the TLS stage, using Pre-Shared Keys (PSK) instead. When a PSK is shared by an authenticator-supplicant pair, the proof of correctness of the 4Way Handshake is easier than the case where TLS establishes the shared secret. Second, supplicants may also run the 4-Way Handshake using a PMK cached from an eariler execution of the composite protocol. This corresponds to restarting exection from the 4-Way Handshake stage. Since our proof holds for an arbitrary set of such backward arcs, using a cached PMK is safe. Note that most of our effort was on proving individual components correct; establishing invariants is a fairly mechanical task.

7. CONCLUSIONS

We present a formal correctness proof for the IEEE 802.11i and TLS protocols using Protocol Composition Logic (PCL). The proof of IEEE 802.11i and TLS supports many design decisions of the 802.11i committee and reinforces conclusions of our previous studies [He and Mitchell 2004; 2005]. For example, the PCL proof demonstrates the need for separate keys for supplicant and authenticator to prevent a reflection attack, supports our previous intuition that various sequence numbers were unnecessary, shows that the protocol remains secure when a nonce-reuse mechanism is adopted in the 4-Way Handshake to reduce a vulnerability of Denial of Service [He and Mitchell 2004; 2005], and supports the adoption of optimized error-recovery strategies. Developing a single correctness proof for a class of error-recovery strategies requires a new composition principle for PCL, which is formulated and proved semantically sound in this paper.

The compositional nature of our protocol logic distinguishes our effort from other methods with similar foundations, such as Paulson's Inductive method [Paulson 1999] and Meadows' NRL protocol analyzer [Meadows 1994], which have been applied to protocols of similar scale and structure. In fact, Meadows previously identified composability of cryptographic protocols as a significant concern in establishing correctness [Meadows 2001]. A compositional approach is useful both for managing scale when working with a large protocol and in understanding how a single protocol interacts with its environment. In order to achieve compositionality, each individual proof involves a set of protocol invariants that must be satisfied in any environment where the protocol runs. These conditions must not only be satisfied by other subprotocols of 802.11i, but also by other protocols using the same certificates or other critical data which are run in parallel with 802.11i. Our proof therefore provides useful deployment information beyond the correctness of the protocols.

Our high-level logic with provable soundness over arbitrary symbolic runs is intended to combine the relative readability and ease of use of BAN-style logics [Burrows et al. 1990] while being based on the semantic protocol execution model of Paulson's method [Paulson 1999]. Although we constructed all proofs manually,

the proof system is completely rigorous and amenable to future automation. For more details about the PCL proof system and the actual proofs described in this paper, we refer the reader to our Protocol Composition Logic web site. Finally, the axioms and rules used in the current proof have been proved sound for a symbolic model of protocol execution and attack. We hope that in the future a computational semantics of PCL, such as suggested in [Datta et al. 2005], can be developed for the entire proof system used here, providing a correctness proof of 802.11i under standard cryptographic assumptions.

REFERENCES

1999. IEEE Standard 802.11-1999. Local and metropolitan area networks - specific requirements - part 11: Wireless LAN Medium Access Control and Physical Layer specifications.

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

2004. Verified By Visa. https://usa.visa.com/personal/security/vbv/.

A. Roy, A. Datta, A. D. J. C. M. J. P. S. Secrecy analysis in protocol composition logic. In *Formal Logical Methods for System Security and Correctness*.

A. Roy, A. Datta, A. D. J. C. M. J. P. S. 2006. Secrecy analysis in protocol composition logic. In *Proceedings of 11th Annual Asian Computing Science Conference (ASIAN)*.

Abadi, M. and Gordn, A. D. 1997. A calculus for cryptographic protocols The Spi Calculus. In *Proceedings of Fourth ACM Conference on Computer and Communications Security*.

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

Borisov, N., Goldberg, I., and Wagner, D. 2001. Intercepting mobile communications: the insecurity of 802.11. In *Proceedings of the 7th Annual International Conference on Mobile Computing and Networking*.

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

Cheung, H. 2005. FBI Teaches Lesson in how to break into Wi-Fi networks. Information Week Network Pipeline.

Datta, A., Derek, A., Mitchell, J., Shmatikov, V., and Turuani, M. 2005. Probabilistic polynomial-time semantics for a protocol security logic. In *Int'l Colloquium on Automata, Languages, and Programming (ICALP)*. To appear.

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

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

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

Datta, A., Derek, A., Mitchell, J. C., and Pavlovic, D. 2004c. Secure protocol composition. In *Proceedings of 19th Annual Conference on Mathematical Foundations of Programming Semantics*. Electronic Notes in Theoretical Computer Science, vol. 83.

Datta, A., Derek, A., Mitchell, J. C., and Roy, A. Protocol composition logic.

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

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

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

F. J. Thayer, J. C. H. and Guttman., J. D. 1999. Mixed strand spaces. In *IEEE Computer Security Foundations Workshop*.

He, C. and Mitchell, J. C. 2004. Analysis of 802.11i 4-way handshake. In *Proceedings of the Third ACM International Workshop on Wireless Security (WiSe'04)*.

He, C. and Mitchell, J. C. 2005. Security analysis and improvements for IEEE 802.11i. In *Proceedings of the 12th Annual Network and Distributed System Security Symposium (NDSS'05)*.

J. Benaloh, B. Lampson, D. S. T. S. and Yee., B. 1995. The private communication technology protocol. *draft-benaloh-pct-00.txt*.

Manna, Z. and Pnueli, A. 1995. *Temporal verification of reactive systems: safety*. Springer-Verlag New York, Inc., New York, NY, USA.

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

Meadows, C. 2001. Open issues in formal methods for cryptographic protocol analysis. *Lecture Notes in Computer Science 2052*, 21–36.

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

Millen, J. K. and Shmatikov, V. 2001. Constraint solving for bounded-process cryptographic protocol analysis. In *Proceedings of ACM Conference on Computer and Communications Security*. 166–175.

Mitchell, J. C. 1998. Finite-state analysis of security protocols. In *Computer Aided Verification*. 71–76.

Mitchell, J. C., Mitchell, M., and U.Stern. 1997. Automated analysis of cryptographic protocols using murphi. In *IEEE Symp. Security and Privacy*. 141–153.

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

Paulson, L. C. 1998. The inductive approach to verifying cryptographic protocols. *Journal of Computer Security 6*, 85–128.

Paulson, L. C. 1999. Inductive analysis of the Internet protocol TLS. *ACM Transactions on Computer and System Security 2,* 3, 332–351.

Paulson, L. C. 2001. Verifying the SET protocol. In *Proceedings of International Joint Conference on Automated Reasoning*.

Ryan, P., Schneider, S., Goldsmith, M., Lowe, G., and Roscoe, A. 2000. *Modelling and Analysis of Security Protocols*. Addison-Wesley Publishing Co.

SETCo. 1997. SET Secure Electronic Transaction specification: Business description.

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

A. PCL PROOF SYSTEM

The details of the programming language, protocol logic and proof system in PCL have been described in previous work [Datta et al. 2004b; Durgin et al. 2003]. The axioms and rules that we use are proven sound in these papers. In our proofs, we use additional axioms regarding hashes.

The proof system combines a complete axiom system for first-order logic (not listed since any axiomatization will do), together with axioms and proof rules for protocol actions, temporal reasoning, and a specialized form of invariance rule.

A.1 Axioms for Protocol Actions

Axioms for protocol actions state properties that hold in the state as a result of executing certain actions (or not executing certain actions). We use a in the axioms to denote any one of the actions and a to denote the corresponding predicate in the

logic. $\top$ denotes the boolean value *true*. Axiom **AA1** states that if a principal has executed an action in some role, then the corresponding predicate asserting that the action had occurred in the past is true while **AA2** states that at the start of a thread any action predicate applied to the thread is false. Axiom **AA3** states that the predicate asserting thread X has not sent the term t remains false after any action that does not send a term that unifies with t, if it is false before the action. **AA4** states that after thread X does actions $a, \cdots, b$ in sequence, the predicates corresponding to the actions a and b are temporally ordered in the same sequence.

AA1	$\top[a]_X\ \mathsf{a}$
AA2	$\mathsf{Start}(X)[\,]_X\ \neg\mathsf{a}(\mathsf{X})$
AA3	$\neg\mathsf{Send}(X,t)[b]_X\neg\mathsf{Send}(X,t)$
	if $\sigma\mathsf{Send}(X,t) \neq \sigma\mathsf{b}$ for all substitutions σ
AA4	$\top[a;\cdots;b]_X\mathsf{a}\langle\mathsf{b}$

The following axioms deal with properties of freshly generated nonces. Axiom **AN1** states that a particular nonce is generated by a unique thread. If thread X generates a new value n and does no further actions, then axiom **AN2** says that no one else knows n, and axiom **AN3** says that n is fresh, and axiom **AN4** says that X is the originating thread of nonce n.

AN1	$\mathsf{New}(X,x) \wedge \mathsf{New}(Y,x) \supset X = Y$
AN2	$\top[\texttt{new}\quad x]_X\ \mathsf{Has}(Y,x) \supset (Y = X)$
AN3	$\top[\texttt{new}\quad x]_X\ \mathsf{Fresh}(X,x)$
AN4	$\mathsf{Fresh}(X,x) \supset \mathsf{Gen}(X,x)$

A.2 Possession Axioms

The possession axioms characterize the terms that a principal can derive if it possesses certain other terms. **ORIG** and **REC** state respectively that a principal possesses a term if she freshly generated it (a nonce) or if she received it in some message. **TUP** and **ENC** enable construction of tuples and encrypted terms if the parts are known. **PROJ** and **DEC** allow decomposition of a tuple into its components and decryption of an encrypted term if the key is known.

ORIG	$\mathsf{New}(X,x) \supset \mathsf{Has}(X,x)$
REC	$\mathsf{Receive}(X,x) \supset \mathsf{Has}(X,x)$
TUP	$\mathsf{Has}(X,x) \wedge \mathsf{Has}(X,y) \supset \mathsf{Has}(X,(x,y))$
ENC	$\mathsf{Has}(X,x) \wedge \mathsf{Has}(X,K) \supset \mathsf{Has}(X,ENC[K](x))$
PROJ	$\mathsf{Has}(X,(x,y)) \supset \mathsf{Has}(X,x) \wedge \mathsf{Has}(X,y)$
DEC	$\mathsf{Has}(X,ENC[K](x)) \wedge \mathsf{Has}(X,K) \supset \mathsf{Has}(X,x)$

Axioms **AR1**, **AR2** and **AR2** are used to model obtaining information about structure of terms as they are being parsed. They us plug in appropriate substitutions obtained by a matching, signature verification and decryption actions to

terms in an action predicate a.

$$\begin{array}{ll}
\textbf{AR1} & \mathsf{a}(x)[\texttt{match} \quad q(x)/q(t)]_X \, \mathsf{a}(t) \\
\textbf{AR2} & \mathsf{a}(x)[\texttt{verify} \quad x,t,K]_X \, \mathsf{a}(SIG[K](t)) \\
\textbf{AR3} & \mathsf{a}(x)[y := \texttt{dec} \quad x,K]_X \, \mathsf{a}(ENC[K](y))
\end{array}$$

A.3 Encryption and Signature

The next two axioms are aimed at capturing the black-box model of encryption and signature. Axiom **VER** refers to the unforgeability of signatures while axiom **SEC** stipulates the need to possess the private key in order to decrypt a message encrypted with the corresponding public key.

$$\begin{array}{ll}
\textbf{SEC} & \mathsf{Honest}(\hat{X}) \wedge \mathsf{Decrypt}(Y, ENC[\hat{X}](x)) \supset (\hat{Y} = \hat{X}) \\
\textbf{VER} & \mathsf{Honest}(\hat{X}) \wedge \mathsf{Verify}(Y, SIG[\hat{X}](x)) \wedge \hat{X} \neq \hat{Y} \supset \\
& \quad \exists X.\mathsf{Send}(X, m) \wedge \mathsf{Contains}(m, SIG[\hat{X}](x))
\end{array}$$

A.4 Generic Rules

These are generic Floyd-Hoare style rules for reasoning about program pre-conditions and post-conditions. For example, the generalization rule **G4** says that if ϕ is a valid formula (it holds in all runs of all protocols) then it can be used in a postcondition of any modal form.

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

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

A.5 Sequencing rule

Sequencing rule gives us a way of sequentially composing two cords P and P' when post-condition of P, matches the pre-condition of P'.

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

A.6 Preservation Axioms

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

$$\begin{array}{ll}
\textbf{P1} & \mathsf{Persist}(X,t)[a]_X\mathsf{Persist}(X,t) \\
& \quad \text{for } \mathsf{Persist} \in \{\mathsf{Has}, \mathsf{FirstSend}, \mathsf{a}, \mathsf{Gen}\}. \\
\textbf{P2} & \mathsf{Fresh}(X,t)[a]_X\mathsf{Fresh}(X,t) \\
& \quad \text{where } t \not\sqsubseteq a.
\end{array}$$

A.7 Axioms and Rules for Temporal Ordering

The next two axioms give us a way of deducing temporal ordering between actions of different threads. Informally, $\mathsf{FirstSend}(X, t, t')$ says that a thread X generated a fresh term t and sent it out first in message t'. This refers to the first such send event and is formally captured by axiom **FS1**. Axiom **FS2** lets us reason that if a thread Y does some action with a term t'' which contains a term t, first sent inside a term t' by a thread X, as a subterm then that send must have occurred before Y's action.

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

FS2 $\quad \mathsf{FirstSend}(X, t, t') \wedge \mathsf{a}(Y, t'') \supset \mathsf{Send}(X, t')\langle \mathsf{a}(Y, t'')$
where $X \neq Y$ and $t \subseteq t''$.

A.8 The Honesty Rule

The honesty rule is an invariance rule for proving properties about the actions of principals that execute roles of a protocol, similar in spirit to the basic invariance rule of LTL [Manna and Pnueli 1995] and invariance rules in other logics of programs. The honesty rule is often used to combine facts about one role with inferred actions of other roles. For example, suppose Alice receives a signed response from a message sent to Bob. Alice may use facts about Bob's role to infer that Bob must have performed certain actions before sending his reply. This form of reasoning may be sound if Bob is honest, since honest, by definition in our framework, means "follows one or more roles of the protocol." The assumption that Bob is honest is essential because the intruder may perform arbitrary actions with any key that has been compromised. Since we have added preconditions to the protocol logic presented in [Durgin et al. 2001; 2003], we reformulate the rule here is a more convenient form using preconditions and postconditions.

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

true. By analogy with database transactions, for example, we consider a property an invariant if it holds after every "transaction" is completed, allowing roles to temporarily violate invariants as long as they preserve them before pausing. A similar convention is normally associated with loop invariants: a property is a loop invariant if it holds every time the top of the loop is reached; it is not necessary that the invariant hold at every point in the body of the loop.

Recall that a protocol $\mathcal{Q}$ is a set of roles $\{\rho_1, \rho_2, \ldots, \rho_k\}$, each executed by zero or more honest principals in any run of $\mathcal{Q}$. A sequence P of actions is a *basic sequence* of role ρ, written $P \in BS(\rho)$, if P is a contiguous subsequence of ρ such that either (i) P starts at the beginning of ρ and ends with the last action before the first receive, or (ii) P starts with a receive action and continues up to the last action before the next receive, or (iii) P starts with the last receive action of the role and continues through the end of the role. In the syntactic presentation below, we use the notation $\forall \rho \in \mathcal{Q}. \forall P \in BS(\rho).\ \phi[P]_X\phi$ to denote a finite set of formulas of the form $\phi[P]_X\phi$ - one for each basic sequence P in the protocol. The quantifiers $\forall \rho \in \mathcal{Q}$ and $\forall P \in BS(\rho)$ are not part of the syntax of PCL, but are meta-notation used to state this rule schema.

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

B. PROOF SYSTEM FOR SECRECY ANALYSIS

In this section, we describe axioms and rules for establishing secrecy, based on [A. Roy 2006]. Secrecy properties are formalized using the $\mathsf{Has}(X, s)$ predicate, which is used to express that honest principal $\hat{X}$ has the information needed to compute the secret s. In a typical two party protocol, $\hat{X}$ is one of two honest agents and s is a nonce generated by one of them. As an intermediate step, we establish that all occurrences of the secret on the network are protected by keys. This property can be proved by induction over possible actions by honest principals, showing that no action leaks the secret if it was not compromised already.

We introduce the predicate $\mathsf{SafeMsg}(M, s, \mathcal{K})$ to assert that every occurrence of s in message M is protected by a key in the set $\mathcal{K}$. Technically speaking, there is an $(n+2)$-ary predicate $\mathsf{SafeMsg}^n(M, s, \mathcal{K})$ for each $n > 0$, allowing the elements of set $\mathcal{K}$ to be listed as arguments. However, we suppress this syntactic detail in this paper. The semantic interpretation of this predicate is defined by induction on the structure of messages. It is actually independent of the protocol and the run.

DEFINITION B.1 SAFEMSG. *Given a run R of a protocol $\mathcal{Q}$, we say $\mathcal{Q}, R$ satisfies* $\mathsf{SafeMsg}(M, s, \mathcal{K})$ *if there exists an i such that* $\mathsf{SafeMsg}_i(M, s, \mathcal{K})$ *where* $\mathsf{SafeMsg}_i$ *is defined by induction on i as follows:*

$\mathsf{SafeMsg}_0(M, s, \mathcal{K})$	*if M is an atomic term different from s*
$\mathsf{SafeMsg}_0(SIG[k](M), s, \mathcal{K})$	*for any M*
$\mathsf{SafeMsg}_0(HASH(M), s, \mathcal{K})$	*for any M*
$\mathsf{SafeMsg}_0(HASH[k](M), s, \mathcal{K})$	*for any k, M*
$\mathsf{SafeMsg}_{i+1}(M_0.M_1, s, \mathcal{K})$	*if* $\mathsf{SafeMsg}_i(M_0, s, \mathcal{K})$ *and* $\mathsf{SafeMsg}_i(M_1, s, \mathcal{K})$
$\mathsf{SafeMsg}_{i+1}(E_{sym}[k](M), s, \mathcal{K})$	*if* $\mathsf{SafeMsg}_i(M, s, \mathcal{K})$ *or* $k \in \mathcal{K}$
$\mathsf{SafeMsg}_{i+i}(E_{pk}[k](M), s, \mathcal{K})$	*if* $\mathsf{SafeMsg}_i(M, s, \mathcal{K})$ *or* $\bar{k} \in \mathcal{K}$

The axioms **SAF0** to **SAF5** below parallel the semantic clauses and follow immediately from them. Equivalences follow as the term algebra is free.

SAF0 $\neg\mathsf{SafeMsg}(s, s, \mathcal{K}) \wedge \mathsf{SafeMsg}(x, s, \mathcal{K})$, where x is an atomic term different from s

SAF1 $\mathsf{SafeMsg}(M_0.M_1, s, \mathcal{K}) \equiv \mathsf{SafeMsg}(M_0, s, \mathcal{K}) \wedge \mathsf{SafeMsg}(M_1, s, \mathcal{K})$

SAF2 $\mathsf{SafeMsg}(E_{sym}[k](M), s, \mathcal{K}) \equiv \mathsf{SafeMsg}(M, s, \mathcal{K}) \vee k \in \mathcal{K}$

SAF3 $\mathsf{SafeMsg}(E_{pk}[k](M), s, \mathcal{K}) \equiv \mathsf{SafeMsg}(M, s, \mathcal{K}) \vee \bar{k} \in \mathcal{K}$

SAF4 $\mathsf{SafeMsg}(HASH(M), s, \mathcal{K})$

SAF5 $\mathsf{SafeMsg}(HASH[k](M), s, \mathcal{K})$

SAF6 $\mathsf{SafeMsg}(SIG[k](M), s, \mathcal{K})$

The formula $\mathsf{SendsSafeMsg}(X, s, \mathcal{K})$ states that all messages sent by thread X are "safe" while $\mathsf{SafeNet}(s, \mathcal{K})$ asserts the same property for all threads. These predicates are definable in the logic as $\mathsf{SendsSafeMsg}(X, s, \mathcal{K}) \equiv \forall M.\ (\mathsf{Send}(X, M) \supset \mathsf{SafeMsg}(M, s, \mathcal{K}))$ and $\mathsf{SafeNet}(s, \mathcal{K}) \equiv \forall X.\ \mathsf{SendsSafeMsg}(X, s, \mathcal{K})$.

In secrecy proofs, we will explicitly assume that the thread generating the secret and all threads with access to a relevant key belong to honest principals. This is semantically necessary since a dishonest principal may reveal its key, destroying secrecy of any data encrypted with it. These honesty assumptions are expressed by the formulas KeyHonest and OrigHonest respectively. KOHonest is the conjunction of the two.

—$\mathsf{KeyHonest}(\mathcal{K}) \equiv \forall X.\ \forall k \in \mathcal{K}.\ (\mathsf{Has}(X, k) \supset \mathsf{Honest}(\hat{X}))$

—$\mathsf{OrigHonest}(s) \equiv \forall X.\ (\mathsf{New}(X, s) \supset \mathsf{Honest}(\hat{X}))$.

—$\mathsf{KOHonest}(s, \mathcal{K}) \equiv \mathsf{KeyHonest}(\mathcal{K}) \wedge \mathsf{OrigHonest}(s)$

We now have the necessary technical machinery to state the induction rule. At a high-level, the **NET** rule states that if each "possible protocol step" P locally sends out safe messages, assuming all messages in the network were safe prior to that step, then all messages on the network are safe.

A possible protocol step P is drawn from the set BS of all basic sequences of roles of the protocol. The basic sequences of a role arise from any partition of the actions in the role into subsequences, provided that if any subsequence contains a `receive` action, then this is the first action of the basic sequence.

NET $\forall \rho \in \mathcal{Q}.\forall P \in BS(\rho).$

$$\frac{\mathsf{SafeNet}(s, \mathcal{K})\ [P]_X\ \mathsf{Honest}(\hat{X}) \wedge \Phi \supset \mathsf{SendsSafeMsg}(X, s, \mathcal{K})}{\mathcal{Q} \vdash \mathsf{KOHonest}(s, \mathcal{K}) \wedge \Phi \supset \mathsf{SafeNet}(s, \mathcal{K})}\ (*)$$

The side condition $(*)$ is: $[P]_A$ does not capture free variables in Φ and $\mathcal{K}$ and the variable s. Φ should be prefix closed. The **NET** rule is written as a rule scheme, in a somewhat unusual form. When applied to a specific protocol $\mathcal{Q}$, there is one formula in the antecedent of the applicable rule instance for each role $\rho \in \mathcal{Q}$ and for each basic sequence $P \in BS(\rho)$; see [Datta et al.].

The axioms **NET0** to **NET3** below are used to establish the antecedent of the **NET** rule. Many practical security protocols consist of steps that each receive a message, perform some operations, and then send a resulting message. The proof strategy in such cases is to use **NET1** to reason that messages received from a safe network are safe and then use this information and the **SAF** axioms to prove that the output message is also safe.

NET0 $\mathsf{SafeNet}(s, \mathcal{K})\ [\,]_X\ \mathsf{SendsSafeMsg}(X, s, \mathcal{K})$

NET1 $\mathsf{SafeNet}(s, \mathcal{K})\ [\texttt{receive}\ M]_X\ \mathsf{SafeMsg}(M, s, \mathcal{K})$

NET2 $\mathsf{SendsSafeMsg}(X, s, \mathcal{K})\ [\mathtt{a}]_X\ \mathsf{SendsSafeMsg}(X, s, \mathcal{K})$, where $\mathtt{a}$ is not a send.

NET3 $\mathsf{SendsSafeMsg}(X, s, \mathcal{K})\ [\texttt{send}\ M]_X\ \mathsf{SafeMsg}(M, s, \mathcal{K}) \supset \mathsf{SendsSafeMsg}(X, s, \mathcal{K})$

Finally, **POS** and **POSL** are used to infer secrecy properties expressed using the Has predicate. The axiom **POS** states that if we have a safe network with respect to s and key-set $\mathcal{K}$ then the only principals who can possess an unsafe message are the generator of s or possessor of a key in $\mathcal{K}$. The **POSL** rule lets a thread use a similar reasoning locally.

$$\textbf{POS}\quad \mathsf{SafeNet}(s,\mathcal{K}) \wedge \mathsf{Has}(X,M) \wedge \neg\mathsf{SafeMsg}(M,s,\mathcal{K}) \supset \exists k \in \mathcal{K}.\ \mathsf{Has}(X,k) \vee \mathsf{New}(X,s)$$

$$\textbf{POSL}\quad \frac{\psi \wedge \mathsf{SafeNet}(s,\mathcal{K})\ [S]_X\ \mathsf{SendsSafeMsg}(X,s,\mathcal{K}) \wedge \mathsf{Has}(Y,M) \wedge \neg\mathsf{SafeMsg}(M,s,\mathcal{K})}{\psi \wedge \mathsf{SafeNet}(s,\mathcal{K})\ [S]_X\ \exists k \in \mathcal{K}.\ \mathsf{Has}(Y,k) \vee \mathsf{New}(Y,s)},$$

where S is any basic sequence of actions.

Following are useful theorems which follow easily from the axioms.

$$\textbf{SREC}\quad \mathsf{SafeNet}(s,\mathcal{K}) \wedge \mathsf{Receive}(X,M) \supset \mathsf{SafeMsg}(M,s,\mathcal{K})$$

$$\textbf{SSND}\quad \mathsf{SafeNet}(s,\mathcal{K}) \wedge \mathsf{Send}(X,M) \supset \mathsf{SafeMsg}(M,s,\mathcal{K})$$