Secrecy Analysis in Protocol Composition Logic*

Arnab Roy[1], Anupam Datta[1], Ante Derek[1], John C. Mitchell[1], Jean-Pierre Seifert[2]

[1] Department of Computer Science, Stanford University, Stanford, CA 94305, USA
[2] Institute for Computer Science, University of Innsbruck, 6020 Innsbruck, Austria

Abstract. Extending a compositional protocol logic with an induction rule for secrecy, we prove soundness for a conventional symbolic protocol execution model, adapt and extend previous composition theorems, and illustrate the logic by proving properties of two key agreement protocols. The first example is a variant of the Needham-Schroeder protocol that illustrates the ability to reason about temporary secrets. The second example is Kerberos V5. The modular nature of the secrecy and authentication proofs for Kerberos makes it possible to reuse proofs about the basic version of the protocol for the PKINIT version that uses public-key infrastructure instead of shared secret keys in the initial steps.

1 Introduction

Two important security properties for key exchange and related protocols are authentication and secrecy. Intuitively, authentication holds between two parties if each is assured that the other has participated in the same session of the same protocol. A secrecy property asserts that some data that is used in the protocol is not revealed to others. If a protocol generates a fresh value, called a *nonce*, and sends it in an encrypted message, then under ordinary circumstances the nonce remains secret in the sense that only agents that have the decryption key can obtain the nonce. However, many protocols have steps that receive a message encrypted with one key, and send some of its parts out encrypted with a different key. Since network protocols are executed asynchronously by independent agents, some potentially malicious, it is non-trivial to prove that even after arbitrarily many steps of independent protocol sessions, secrets remain inaccessible to an attacker.

Our general approach involves showing that every protocol agent that receives data protected by one of a chosen set of encryption keys only sends sensitive data out under encryption by another key in the set. This reduces a potentially complicated proof about arbitrary runs involving a malicious attacker to a case-by-case analysis of how each protocol step might save and send data. We formalize this form of inductive reasoning about secrecy in a set of new axioms and inference rules that are added to Protocol Composition Logic (PCL) [14, 9, 8, 10, 11], prove soundness of the system over a conventional symbolic protocol execution model, and illustrate its use with two protocol examples. The extended logic may be used to prove authentication or secrecy, independently and in situations where one property may depend upon the other.

* This work was partially supported by the NSF TRUST Science and Technology Center. The first author was partially supported by a Siebel Fellowship. Part of the work was done when the first and fifth authors were at Intel Corporation.

Among other challenges, the inductive secrecy rule presented here is carefully designed to be sound for reasoning about arbitrarily many simultaneous protocol sessions, and powerful enough to prove meaningful properties about complex protocols used in practice. While the underlying principles are similar to the "rank function method" [18] and work using the strand space execution model [19], our system provides precise formal proof rules that are amenable to automation. In addition, casting secrecy induction in the framework of Protocol Composition Logic avoids limitations of some forms of rank function arguments and eliminates the need to reason explicitly about possible actions of a malicious attacker. Compositional secrecy proofs are made possible by theorems developed in this paper, extending previous composition theorems for a simpler proof system [11, 15].

Our first protocol example is a variant of the Needham-Schroeder protocol, used in [16] to illustrate a limitation of the original rank function method and motivate an extension for reasoning about temporary secrets. The straightforward formal proof in section 4 therefore shows that our method does not suffer from the limitations identified in [16]. Intuitively, the advantage of our setting lies in the way that modal formulas of PCL state properties about specific points in protocol execution, rather than only properties that must be true at all points in all runs.

Our second protocol example is Kerberos [17], which is widely used for authenticated client-server interaction in local area networks. The basic protocol has three sections, each involving an exchange between the client and a different service. We develop a formal proof that is modular, with the proof for each section assuming a precondition and establishing a postcondition that implies the precondition of the following section. One advantage of this modular structure is illustrated by our proof for the PKINIT [7] version that uses public-key infrastructure instead of shared secret keys in the initial steps. Since only the first section of PKINIT is different, the proofs for the second and third sections of the protocol remain unchanged. In previous work, Bella and Paulson use theorem proving techniques to reason about properties of Kerberos that hold in all traces containing actions of honest parties and a malicious attacker [3]. Our high-level axiomatic proofs are significantly more concise since we do not require explicit reasoning about attacker actions. Another line of work uses a multiset rewriting model to develop proofs in the symbolic and computational model [4, 2]. However, proofs in these papers use unformalized (though rigorous) mathematical arguments and are not modular.

The rest of the paper is organized as follows. Some background on PCL is given in section 2, followed by the secrecy-related axioms and proof rules in section 3. The first protocol example is presented in section 4. Composition theorems are developed in section 5, and applied in the proofs for Kerberos in section 6. Finally, we conclude in section 7.

2 Background

Protocol Composition Logic (PCL) is developed in [14, 9, 8, 10], with [11] providing a relatively succinct presentation of the most current form. A simple protocol programming language is used to represent a *protocol* by a set of *roles*, such as "Initiator", "Responder" or "Server", each specifying a sequence of actions to be executed by an honest participant. Protocol actions include nonce generation, encryption, decryption and communication steps (sending and receiving). Every principal can be executing one or more copies of each role at the same time. We use the word *thread* to refer to

Action formulas
$\mathsf{a} ::= \mathsf{Send}(X,t) \mid \mathsf{Receive}(X,t) \mid \mathsf{New}(X,t) \mid \mathsf{Encrypt}(X,t) \mid \mathsf{Start}(X)$
Formulas
$\varphi ::= \mathsf{a} \mid \mathsf{Has}(X,t) \mid \mathsf{Honest}(\hat{X}) \mid \varphi \wedge \varphi \mid \neg\varphi \mid \exists V.\ \varphi$
Modal form
$\Psi ::= \varphi\ [Actions]_X\ \varphi$

Table 1. Syntax of the logic

a principal executing one particular instance of a role. Each *thread* X is a pair $(\hat{X}, \eta)$ where $\hat{X}$ is a principal and η is a unique session id. A *run* is a record of all actions executed by honest principals and the attacker during protocol execution. The set of runs of a protocol is determined by the operational semantics of the protocol programming language.

Table 1 summarizes the syntax of the logic used in this paper. Protocol proofs usually use modal formulas of the form $\psi[P]_X\varphi$. The informal reading of the modal formula is that if X starts from a state in which ψ holds, and executes the program P, then in the resulting state the security property φ is guaranteed to hold irrespective of the actions of an attacker and other honest principals. The formulas of the logic are interpreted over protocol runs. We say that protocol $\mathcal{Q}$ satisfies formula φ, denoted $\mathcal{Q} \vDash \varphi$, if in all runs R of $\mathcal{Q}$ the formula φ holds, *i.e.,* $\mathcal{Q}, R \vDash \varphi$. For example, $\mathsf{Send}(X,t)$ holds in a run where the thread X has sent the term t. For every protocol action, there is a corresponding action predicate which asserts that the action has occurred in the run. Action predicates are useful for capturing authentication properties of protocols since they can be used to assert which principals sent and received certain messages. $\mathsf{Encrypt}(X,t)$ means that X computes the encrypted term t, while $\mathsf{New}(X,n)$ means X generates fresh nonce n. $\mathsf{Honest}(\hat{X})$ means that $\hat{X}$ is acting honestly, *i.e.,* the actions of every thread of $\hat{X}$ precisely follows some role of the protocol. $\mathsf{Start}(X)$ means that the thread X did not execute any actions in the past. $\mathsf{Has}(X,t)$ means X can compute the term t using symbolic Dolev-Yao rules, *e.g.* receiving it in the clear or receiving it under encryption where the decryption key is known.

To illustrate the terminology used in this section we describe the formalization of Kerberos V5, which is a protocol used to establish mutual authentication and a shared session key between a client and an application server [17]. It involves trusted principals known as the Kerberos Authentication Server (KAS) and the Ticket Granting Server (TGS). There are pre-shared long term keys between the client and the KAS, the KAS and the TGS, and the TGS and the application server. Typically, the KAS shares long-term keys with a number of clients and the TGS with a number of application servers. However, there is no pre-shared long term secret between a given client and an application server. Kerberos achieves establishment of mutual authentication and a shared session key between the client and the application server using the chain of trust leading from the client to the KAS and the TGS to the application server.

Kerberos has four roles, one for each kind of participant - **Client**, **KAS**, **TGS** and **Server**. The long-term shared keys are written here in the form $k_{X,Y}^{type}$ where X and Y are the principals sharing the key. The *type* appearing in the superscript indicates the relationship between X and Y in the transactions involving the use of the key. There

are three *type*s required in Kerberos: $c \rightarrow k$ indicates that X is acting as a client and Y is acting as a KAS, $t \rightarrow k$ is for TGS and KAS and $s \rightarrow t$ is for application server and TGS. Kerberos runs in three stages with the client role participating in all three. The description of the roles is based on the A level formalization of Kerberos V5 in [5]. We describe the formalization of the first stage in some detail so that the rest is easy to follow.

In the first stage, shown below, the client thread (C) generates a nonce (n_1) and sends it to the KAS ($\hat{K}$) along with the identities of the TGS ($\hat{T}$) and itself. The KAS generates a new nonce ($AKey$ - Authentication Key) to be used as a session key between the client and the TGS. It then sends this key along with some other fields to the client encrypted (represented by the `match` actions) under two different keys - one it shares with the client ($k^{c \rightarrow k}_{C,K}$) and one it shares with the TGS ($k^{t \rightarrow k}_{T,K}$). The encryption with $k^{t \rightarrow k}_{T,K}$ is called the *ticket granting ticket* (tgt). The client extracts $AKey$ by decrypting the component encrypted with $k^{c \rightarrow k}_{C,K}$.

Client $= (C, \hat{K}, \hat{T}, \hat{S}, t)$ [	**KAS** $= (K)$ [
`new` n_1;	`receive` $\hat{C}.\hat{T}.n_1$;
`send` $\hat{C}.\hat{T}.n_1$;	`new` $AKey$;
	`match` $E_{sym}[k^{t \rightarrow k}_{T,K}](AKey.\hat{C})$ `as` tgt;
`receive` $\hat{C}.tgt.enc_{kc}$;	`match` $E_{sym}[k^{c \rightarrow k}_{C,K}](AKey.n_1.\hat{T})$
`match` enc_{kc}	`as` enc_{kc};
`as` $E_{sym}[k^{c \rightarrow k}_{C,K}](AKey.n_1.\hat{T})$;	`send` $\hat{C}.tgt.enc_{kc}$;
.....	$]_K$

In the second stage (not shown), the client uses the session key $AKey$ and the *ticket granting ticket* to interact with the TGS and gets a new session key $SKey$ and a *service ticket* (st). In the third stage, the client encrypts its identity and a timestamp with $SKey$ and sends it to the application server along with the service ticket. The server decrypts st and extracts the $SKey$. It then uses the session key to decrypt the client's encryption, matches the first component of the decryption with the identity of the client and extracts the timestamp. It then encrypts the timestamp with the session key and sends it back to the client. The client decrypts the message and matches it against the timestamp it used. The control flow of Kerberos exhibits a staged architecture where once one stage has been completed successfully, the subsequent stages can be performed multiple times or aborted and started over for handling errors.

3 Proof System for Secrecy Analysis

In this section, we extend PCL with new axioms and rules for establishing secrecy. Secrecy properties are formalized using the $\mathsf{Has}(X, s)$ predicate and requiring that $\hat{X}$ refer only to honest principals who share 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 and reasoning 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, for each $n > 0$, there is an $(n+2)$-ary predicate $\mathsf{SafeMsg}^n(M, s, \mathcal{K})$, with n corresponding to the size of set $\mathcal{K}$. 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 1 (SafeMsg). *Given a run R of a protocol $\mathcal{Q}$, we say $\mathcal{Q}, R \vDash \mathsf{SafeMsg}(M, s, \mathcal{K})$ if there exists an i such that $\mathsf{SafeMsg}_i(M, s, \mathcal{K})$ where $\mathsf{SafeMsg}_i$ is defined as follows:*

$\mathsf{SafeMsg}_0(M, s, \mathcal{K})$	*if M is an atomic term different from s*
$\mathsf{SafeMsg}_0(HASH(M), s, \mathcal{K})$	*for any 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 $k \in \mathcal{K}$*
$\mathsf{SafeMsg}_{i+1}(E_{pk}[k](M), s, \mathcal{K})$	*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 $\mathsf{SafeMsg}(HASH(M), s, \mathcal{K})$

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 formulas may be written 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 of basic sequences BS for all roles of the protocol. A set of basic sequences of a role is any partition of the sequence of actions in the role such that if any element sequence has a `receive` action then it is only at the beginning.

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

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

$(*)$: $[P]_A$ does not capture free variables in Φ and $\mathcal{K}$ and the variable s. Φ should be prefix closed .

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

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

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

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.

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

POSL $$\frac{\psi \wedge \mathsf{SafeNet}(s, \mathcal{K})\ [S]_X\ \mathsf{SendsSafeMsg}(X, s, \mathcal{K}) \wedge \mathsf{Has}(Y, M) \wedge \neg\mathsf{SafeMsg}(M, s, \mathcal{K})}{\psi \wedge \mathsf{SafeNet}(s, \mathcal{K})\ [S]_X\ \exists k \in \mathcal{K}.\ \mathsf{Has}(Y, k) \vee \mathsf{New}(Y, s)},$$
where S is any basic sequence of actions.

Following are useful theorems which follow easily from the axioms.

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

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

We write $\Gamma \vdash \gamma$ if γ is provable from the formulas in Γ and any axiom or inference rule of the proof system except the honesty rule **HON** from previous formulations of PCL and the secrecy rule **NET**. We write $\mathcal{Q} \vdash \gamma$ if γ is provable from the axioms and inference rules of the proof system including the rules **HON** and **NET** for protocol $\mathcal{Q}$.

Given a set of messages $\mathcal{M}$, let us denote by $\tilde{\mathcal{M}}$ to be the minimal set containing $\mathcal{M}$ and closed under pairing, unpairing, encryption with any public key or symmetric key, decryption with a private key or a symmetric key not in $\mathcal{K}$ and hashing.

Theorem 1. *If $\mathcal{M}$ is a set of messages, all safe with respect to secret s and key-set $\mathcal{K}$, then $\tilde{\mathcal{M}}$ also contains only safe messages.*

Proof. Any element $m \in \tilde{\mathcal{M}}$ can be constructed from elements in $\mathcal{M}$ using a finite sequence of the operations enumerated. From the semantics of $\mathsf{SafeMsg}$ it is easily seen that all the operations preserve safeness. Hence by induction, all the elements of $\tilde{\mathcal{M}}$ will be safe. □

Lemma 1. *If a thread X, at any point in any protocol, possesses an unsafe message with respect to secret s and key-set $\mathcal{K}$ then either X received an unsafe message earlier, or X generated s, or X possesses a key in $\mathcal{K}$.*

Proof. Suppose thread X does not satisfy any of the conditions enumerated. Then the set of messages $\mathcal{M}$ it initially knows and has received are safe messages. Since it does not have a key in $\mathcal{K}$, $\tilde{\mathcal{M}}$ is a superset of all the messages it can construct from $\mathcal{M}$ (in the Dolev-Yao model). Hence, by theorem 1, X cannot compute any unsafe message. So it cannot possess an unsafe message – a contradiction. □

Theorem 2 (Soundness). *If $\mathcal{Q} \vdash \gamma$, then $\mathcal{Q} \vDash \gamma$. Furthermore, if $\Gamma \vdash \gamma$, then $\Gamma \vDash \gamma$.*

Proof. Soundness for this proof system is proved, by induction on the length of proofs of the axioms and rules, the most interesting of which are sketched below.

NET : Consider a run R of protocol $\mathcal{Q}$ such that the consequent of **NET** is false. We will show that the antecedent is false too. We have $\mathcal{Q}, R \vDash \mathsf{KOHonest}(s, \mathcal{K}) \wedge \Phi$, but $\mathcal{Q}, R \nvDash \mathsf{SafeNet}(s, \mathcal{K})$. This implies that $\mathcal{Q}, R \vDash \exists m, X. \mathsf{Send}(X, m) \wedge \neg \mathsf{SafeMsg}(m, s, \mathcal{K})$. Note that there must be a first instance when an unsafe message is sent out - let $\tilde{m}$ be the first such message. Hence, we can split R into $R_0.R_1.R_2$ such that $\mathcal{Q}, R_0 \vDash \mathsf{SafeNet}(s, \mathcal{K})$ and $R_1 = \langle X \texttt{ sends } \tilde{m}; Y \texttt{ receives } \tilde{m} \rangle$, for some Y.

Since this is the first send of an unsafe message, therefore X could not have received an unsafe message earlier. Therefore, by the lemma, either X generated s or, X has a key in $\mathcal{K}$. In both cases, $\mathsf{KOHonest}(s, \mathcal{K})$ implies $\mathsf{Honest}(\hat{X})$. Therefore the fragment $[\texttt{send } \tilde{m}]_X$ must be part of a sequence of actions $[P]_X$ such that P is a basic sequence of one of the roles in $\mathcal{Q}$ - but, this violates the premise of **NET**. Hence the theorem. The need for Φ to be prefix-closed comes from a more detailed version of this proof .

POS : $\mathsf{SafeNet}(s, \mathcal{K})$ implies no thread sent out an unsafe message in the run. Hence no thread received an unsafe message. Therefore, by lemma 1, any thread X possessing an unsafe message must have either generated s or possesses a key in $\mathcal{K}$.

POSL : The premise of the rule informally states that starting from a "safe" network and additional constraints ψ thread X concludes that some thread Y possesses an unsafe message M in all possible runs of any protocol. Specifically this should be true for a run where thread X executes the basic sequence $[S]_X$ uninterspersed with the actions of any other thread except the receipt of messages sent by X. Now the premise implies that X only sends safe messages - also since S is a basic sequence, the only message that X can receive in $[S]_X$ will be only at its beginning, which, due to the starting "safe" network precondition will be a safe message. Hence we can conclude that thread Y possessed an unsafe message before X started executing $[S]_X$ *i.e.*, when $\mathsf{SafeNet}(s, \mathcal{K})$ was true. Therefore using axiom **POS** we derive that thread Y either generated s or possesses a key in $\mathcal{K}$, which establises the conclusion of **POSL**. □

4 Analysis of a variant of NSL

In this section we use the proof system developed in section 3 to prove a secrecy property of a simple variant $NSLVAR$ of the Needham-Schroeder-Lowe protocol, proposed in [16], in which parties A and B use an authenticated temporary secret n_a to establish a secret key k that is in turn used to protect the actual message m. The main difference from the original NSL protocol is that the initiator's nonce is leaked in the final message. Reasoning from A's point of view, nonce n_a should be secret between A and B at the point of the run in the protocol where A is just about to send the last

message. This protocol was originally used to demonstrate a limitation of the original rank function method in reasoning about temporary secrets. Modal formulas in PCL allow us to naturally express and prove properties that hold at intermediate points of a protocol execution.

Formally, $NSLVAR$ is a protocol defined by roles $\{\mathbf{Init}, \mathbf{Resp}\}$, with the roles, written using the protocol program notation, given below.

$\mathbf{Init} = (A, \hat{B}, m)$ [	$\mathbf{Resp} = (B)$ [
`new` n_a;	`receive` enc_{r1};
`match` $E_{pk}[k_B](\hat{A}.n_a)$ `as` enc_{r1};	`match` enc_{r1} `as` $E_{pk}[k_B](\hat{A}.n_a)$;
`send` enc_{r1};	`new` k;
	`match` $E_{pk}[k_A](n_a.\hat{B}.k)$ `as` enc_i;
`receive` enc_i;	`send` enc_i;
`match` enc_i `as` $E_{pk}[k_A](n_a.\hat{B}.k)$;	
`match` $E_{sym}[k](m)$ `as` enc_{r2};	`receive` $enc_{r2}.n_a$;
`send` $enc_{r2}.n_a$;	`match` enc_{r2} `as` $E_{sym}[k](m)$;
$]_A$	$]_B$

Theorem 3. *Let* $\tilde{\mathbf{Init}}$ *denote the initial segment of the initiator's role ending just before the last send action. The nonce* n_a *is a shared secret between* A *and* B *in every state of the protocol where* A *has executed* $\tilde{\mathbf{Init}}$ *and no further actions, as long as both* $\hat{A}$ *and* $\hat{B}$ *are honest. Formally,*

$$NSLVAR \vdash [\tilde{\mathbf{Init}}]_A\ \mathsf{Honest}(\hat{A}) \wedge \mathsf{Honest}(\hat{B}) \supset (\mathsf{Has}(X, n_a) \supset \hat{X} = \hat{A} \vee \hat{X} = \hat{B})$$

Proof Sketch. To prove the secrecy property, we start off by proving an authentication property $[\tilde{\mathbf{Init}}]_A \mathsf{Honest}(\hat{A}) \wedge \mathsf{Honest}(\hat{B}) \supset \Phi$, where Φ is the conjunction of the following formulas:

$$\Phi_1 : \forall X, \hat{Y}.\ \mathsf{New}(X, n_a) \wedge \mathsf{Send}(X, E_{pk}[k_Y](\hat{X}.n_a)) \supset \hat{Y} = \hat{B}$$
$$\Phi_2 : \forall X, \hat{Y}, n.\ \mathsf{New}(X, n_a) \supset \neg\mathsf{Send}(X, E_{pk}[k_Y](n.\hat{X}.n_a))$$
$$\Phi_3 : \forall X, e.\ \mathsf{New}(X, n_a) \supset \neg\mathsf{Send}(X, e.n_a)$$
$$\Phi_4 : \mathsf{Honest}(\hat{X}) \wedge \mathsf{Send}(X, E_{sym}[k_0](m_0).n) \supset \mathsf{New}(X, n)$$
$$\Phi_5 : \mathsf{Honest}(\hat{X}) \wedge \mathsf{EncSend}(X, E_{pk}[k_Y](\hat{X}'.n)) \supset \hat{X}' = \hat{X}$$

Informally, Φ_1 and Φ_2 hold because from the thread A's point of view it is known that it itself generated the nonce n_a and did not send it out encrypted with any other principal's public key except $\hat{B}$'s and that too in a specific format described by the protocol. Φ_3 holds because we are considering a state in the protocol execution where A has not yet sent the last message - sending of the last message will make $\mathsf{Send}(A, e.n_a)$ true with $e = E_{sym}[k](m)$. These intuitive explanations can be formalized using a previously developed fragment of PCL but we will omit those steps in this paper. Φ_4 and Φ_5 follow from a straightforward use of the honesty rule.

In the next step we prove the antecedents of the **NET** rule. We take $\mathcal{K} = \{\bar{k_A}, \bar{k_B}\}$ where the bar indicates private key which makes $\mathsf{KeyHon}(\mathcal{K}) \equiv \mathsf{Honest}(\hat{A}) \wedge \mathsf{Honest}(\hat{B})$.

In addition, since thread A generates n_a, therefore $\mathsf{KOHonest}(n_a, \mathcal{K}) \equiv \mathsf{Honest}(\hat{A}) \wedge \mathsf{Honest}(\hat{B})$. We show that all basic sequence of the protocol send "safe" messages, assuming that formula Φ holds and that the predicate $\mathsf{SafeNet}$ holds at the beginning of that basic sequence. Formally, for every basic sequence $\mathbf{P} \in \{\mathbf{Init}_1, \mathbf{Init}_2, \mathbf{Resp}_1, \mathbf{Resp}_2\}$ we prove that: $\mathsf{SafeNet}(n_a, \mathcal{K})[\mathbf{P}]_{A'}\ \mathsf{Honest}(\hat{A}') \wedge \Phi \supset \mathsf{SendsSafeMsg}(A', n_a, \mathcal{K})$ □

Some secrecy proofs using the CSP [18] or strand space [19] protocol execution model use inductive arguments that are similar to the form of inductive reasoning codified in our formal system. For example, within CSP, properties of messages that may appear on the network have been identified by defining a *rank function* [18, 16], with an inductive proof used to show that rank is preserved by the attacker actions and all honest parties. In comparison, arguments in our formal logic use a conjunction involving the $\mathsf{SafeNet}$ predicate and protocol specific properties Φ in our inductive hypotheses. These two formulas together characterize the set of possible messages appearing on the network and can be viewed as a symbolic definition of a rank function. We believe that our method is as powerful as the rank function method for any property expressible in our logic. However, it is difficult to prove a precise connection without first casting the rank function method in a formal setting that relies on a specific class of message predicates.

One drawback of the rank function approach is that the induction is performed by "global" reasoning. While analyzing a protocol, all relevant properties of the system (such as authentication and secrecy, for example) are modelled using a single rank function and proved to hold simultaneously. This makes the method somewhat less applicable since it cannot handle protocols which deal with temporary secrets or use authentication to ensure secrecy properties. In contrast, PCL allows separation and incremental proofs of different properties. Although some of these issues can be resolved by extensions of the rank function method [13, 12], the PCL approach seems more general and may be better suited for some applications.

5 Compositional Reasoning for Secrecy

In this section, we present composition theorems that allow secrecy proofs of compound protocols to be built up from proofs of their parts. An application of this method to the Kerberos protocol is given in the next section. We consider three kinds of composition operations on protocols—*parallel*, *sequential*, and *staged*—as in our earlier work [11, 15]. However, adapting that approach for reasoning about secrecy requires some work. One central concept in our compositional proof methods is the notion of an *invariant*. An invariant for a protocol is a logical formula that characterizes the environment in which it retains its security properties. While in previous work we had one rule for establishing invariants (the **HON** rule [11]), reasoning about secrecy requires, in addition, the **NET** rule introduced in this paper. A second point of difference arises from the fact that reasoning about secrecy requires a certain degree of global knowledge. Specifically, while proving that a protocol step does not violate secrecy, it is sometimes necessary to use information from earlier steps. In the technical presentation, this history information shows up as preconditions in the secrecy induction of the sequential and staged composition theorems.

Definition 2 (Parallel Composition). *The parallel composition $\mathcal{Q}_1 \otimes \mathcal{Q}_2$ of protocols $\mathcal{Q}_1$ and $\mathcal{Q}_2$ is the union of the sets of roles of $\mathcal{Q}_1$ and $\mathcal{Q}_2$.*

The parallel composition operation allows modelling agents who simultaneously engage in sessions of multiple protocols. The parallel composition theorem provides a method for ensuring that security properties established independently for the constituent protocols are still preserved in such a situation.

Theorem 4 (Parallel Composition). *If $\mathcal{Q}_1 \vdash \Gamma$ and $\Gamma \vdash \Psi$ and $\mathcal{Q}_2 \vdash \Gamma$ then $\mathcal{Q}_1 \otimes \mathcal{Q}_2 \vdash \Psi$, where Γ denotes the set of invariants used in the proof of Ψ.*

One way to understand the parallel composition theorem is to visualize the proof tree for Ψ for protocol $\mathcal{Q}_1$ in red and green colors. The steps which use the invariant rules are colored red and correspond to the part $\mathcal{Q}_1 \vdash \Gamma$, while all other proof steps are colored green and correspond to the part $\Gamma \vdash \Psi$. While composing protocols, all green steps are obviously preserved since they involve proof rules which hold for all protocols. The red steps could possibly be violated because of $\mathcal{Q}_2$. For example, one invariant may state that honest principals only sign messages of a certain form, while $\mathcal{Q}_2$ may allow agents to sign other forms of messages. The condition $\mathcal{Q}_2 \vdash \Gamma$ ensures that this is not the case, i.e., the red steps still apply for the composed protocol.

Definition 3 (Sequential Composition). *A protocol $\mathcal{Q}$ is a sequential composition of two protocols $\mathcal{Q}_1$ and $\mathcal{Q}_2$, if each role of $\mathcal{Q}$ is obtained by the sequential composition of a role of $\mathcal{Q}_1$ with a role of $\mathcal{Q}_2$.*

In practice, key exchange is usually followed by a secure message transmission protocol which uses the resulting shared key to protect data. Sequential composition is used to model such compound protocols. Formally, the composed role $P_1; P_2$ is obtained by concatenating the actions of P_1 and P_2 with the output parameters of P_1 substituted for the input parameters of P_2 (cf. [11]).

Theorem 5 (Sequential Composition). *If $\mathcal{Q}$ is a sequential composition of protocols $\mathcal{Q}_1$ and $\mathcal{Q}_2$ then we can conclude $\mathcal{Q} \vdash \mathsf{KOHonest}(s, \mathcal{K}) \wedge \Phi \supset \mathsf{SafeNet}(s, \mathcal{K})$ if the following conditions hold for all $P_1; P_2$ in $\mathcal{Q}$, where $P_1 \in \mathcal{Q}_1$ and $P_2 \in \mathcal{Q}_2$:*

1. *(Secrecy induction)*
   - $\forall i. \forall S \in BS(P_i).\ \theta_{P_i} \wedge \mathsf{SafeNet}(s, \mathcal{K})[S]_X \mathsf{Honest}(\hat{X}) \wedge \Phi \supset \mathsf{SendsSafeMsg}(X, s, \mathcal{K})$
2. *(Precondition induction)*
   - $\mathcal{Q}_1 \otimes \mathcal{Q}_2 \vdash \mathsf{Start}(X) \supset \theta_{P_1}$ *and* $\mathcal{Q}_1 \otimes \mathcal{Q}_2 \vdash \theta_{P_1}[P_1]_X\ \theta_{P_2}$
   - $\forall i. \forall S \in BS(P_i).\ \theta_{P_i}[S]_X\ \theta_{P_i}$.

The final conclusion of the theorem is a statement that secrecy of s is preserved in the composed protocol. The secrecy induction is very similar to the **NET** rule. It states that all basic sequences of the two roles only send out safe messages. This step is compositional since the condition is proved independently for steps of the two protocols. One point of difference from the **NET** rule is the additional precondition θ_{P_i}. This formula usually carries some information about the history of the execution, which helps in deciding what messages are safe for A to send out. For example, if θ_{P_i} says that A received some message m, then it is easy to establish that m is a safe message for A to send out again. The precondition induction proves that the θ_{P_i}'s hold at each point where they are assumed in the secrecy induction. The first bullet states the base case of the induction: θ_{P_1} holds at the beginning of the execution and θ_{P_2} holds when P_1 completes. The second bullet states that the basic sequences of P_1 and P_2 preserve their respective preconditions. This theorem is existential in the preconditions, *i.e.*, the theorem holds if there exist any set of formulas θ_{P_i} satisfying the conditions.

Definition 4 (Staged Composition). *A protocol $\mathcal{Q}$ is a staged composition of protocols $\mathcal{Q}_1, \mathcal{Q}_2, \ldots, \mathcal{Q}_n$ if each role of $\mathcal{Q}$ is of the form $RComp(\langle R_1, R_2, \ldots, R_n\rangle)$, where R_i is a role of protocol $\mathcal{Q}_i$.*

Consider the representation of sequential composition of n protocols as a directed graph with edges from $\mathcal{Q}_i$ to $\mathcal{Q}_{i+1}$. The staged composition operation extends sequential composition by allowing self loops and arbitrary backward arcs in this chain. This control flow structure is common in practice, e.g., Kerberos [17], IEEE 802.11i [1], and IKEv2 [6]. A role in this composition, denoted $RComp(\langle ...\rangle)$ corresponds to a possible execution path in the control flow graph by a single thread (cf. [15]). Note that the roles are built up from a finite number of basic sequences of the component protocol roles.

Theorem 6 (Staged Composition). *If $\mathcal{Q}$ is a staged composition of protocols $\mathcal{Q}_1$, $\mathcal{Q}_2$, $\cdots$, $\mathcal{Q}_n$ then we can conclude $\mathcal{Q} \vdash \mathsf{KOHonest}(s, \mathcal{K}) \wedge \Phi \supset \mathsf{SafeNet}(s, \mathcal{K})$ if for all $RComp(\langle P_1, P_2, \cdots, P_n\rangle) \in \mathcal{Q}$:*

1. *(Secrecy induction)*
   - $\forall i. \forall S \in BS(P_i).\ \theta_{P_i} \wedge \mathsf{SafeNet}(s, \mathcal{K})[S]_X \mathsf{Honest}(\hat{X}) \wedge \Phi \supset \mathsf{SendsSafeMsg}(X, s, \mathcal{K})$
2. *(Precondition induction)*
   - $\mathcal{Q}_1 \otimes \mathcal{Q}_2 \cdots \otimes \mathcal{Q}_n \vdash \mathsf{Start}(X) \supset \theta_{P_1}$ *and* $\mathcal{Q}_1 \otimes \mathcal{Q}_2 \cdots \otimes \mathcal{Q}_n \vdash \forall i.\, \theta_{P_i}[P_i]_X\, \theta_{P_{i+1}}$
   - $\forall i. \forall S \in \bigcup_{j \geq i} BS(P_j).\, \theta_{P_i}[S]_X\, \theta_{P_i}$.

The secrecy induction for staged composition is the same as for sequential composition. However, the precondition induction requires additional conditions to account for the control flows corresponding to backward arcs in the graph. The technical distinction surfaces in the second bullet of the precondition induction. It states that precondition θ_{P_i} should also be preserved by basic sequences of all higher numbered components, i.e., components from which there could be backward arcs to the beginning of P_i. Again, the theorem holds if there exist any set of formulas θ_{P_i} satisfying the conditions.

6 Analysis of Kerberos

In this section we analyze Kerberos V5, which was described in section 2. The security properties of Kerberos that we prove are listed in table 2. We abbreviate the honesty assumptions by defining $\mathsf{Hon}(\hat{X}_1, \cdots, \hat{X}_n) \equiv \mathsf{Honest}(\hat{X}_1) \wedge \cdots \mathsf{Honest}(\hat{X}_n)$. The security objectives are of two types: authentication and secrecy. The authentication objectives take the form that a message of a certain format was indeed sent by some thread of the expected principal. The secrecy objectives take the form that a putative secret is known only to certain principals. For example, $AUTH_{kas}^{client}$ states that when the thread C finishes executing the **Client** role, some thread of $\hat{K}$ (the KAS) indeed sent the expected message; SEC_{akey}^{client} states that the authorization key is secret after execution of the **Client** role by C; the other security properties are analogous.

Theorem 7 (KAS Authentication). *On execution of the* **Client** *role by a principal it is guaranteed that the intended KAS indeed sent expected response assuming that the both the client and the KAS are honest. Similar result holds for a principal executing the* **TGS** *role. Formally, $KERBEROS \vdash AUTH_{kas}^{client}, AUTH_{kas}^{tgs}$*

$$SEC_{akey} : \mathsf{Hon}(\hat{C}, \hat{K}, \hat{T}) \supset (\mathsf{Has}(X, AKey) \supset \hat{X} \in \{\hat{C}, \hat{K}, \hat{T}\})$$
$$SEC_{skey} : \mathsf{Hon}(\hat{C}, \hat{K}, \hat{T}, \hat{S}) \supset (\mathsf{Has}(X, SKey) \supset \hat{X} \in \{\hat{C}, \hat{K}, \hat{T}, \hat{S}\})$$
$$AUTH_{kas} : \exists \eta.\, \mathsf{Send}((\hat{K}, \eta), \hat{C}.E_{sym}[k^{t \to k}_{T,K}](AKey.\hat{C}).E_{sym}[k^{c \to k}_{C,K}](AKey.n_1.\hat{T}))$$
$$AUTH_{tgs} : \exists \eta.\, \mathsf{Send}((\hat{T}, \eta), \hat{C}.E_{sym}[k^{s \to t}_{S,T}](SKey.\hat{C}).E_{sym}[AKey](SKey.n_2.\hat{S}))$$

$SEC^{client}_{akey} : [\mathbf{Client}]_C\ SEC_{akey}$ $\quad$ $AUTH^{client}_{kas} : [\mathbf{Client}]_C\ \mathsf{Hon}(\hat{C}, \hat{K}) \supset AUTH_{kas}$

$SEC^{kas}_{akey} : [\mathbf{KAS}]_K\ SEC_{akey}$ $\quad$ $AUTH^{tgs}_{kas} : [\mathbf{TGS}]_T\ \mathsf{Hon}(\hat{T}, \hat{K}) \supset \exists n_1.\, AUTH_{kas}$

$SEC^{tgs}_{akey} : [\mathbf{TGS}]_T\ SEC_{akey}$

$AUTH^{client}_{tgs} : [\mathbf{Client}]_C\ \mathsf{Hon}(\hat{C}, \hat{K}, \hat{T}) \supset AUTH_{tgs}$

$SEC^{client}_{skey} : [\mathbf{Client}]_C\ SEC_{skey}$ $\quad$ $AUTH^{server}_{tgs} : [\mathbf{Server}]_S\ \mathsf{Hon}(\hat{S}, \hat{T})$

$SEC^{tgs}_{skey} : [\mathbf{TGS}]_T\ SEC_{skey}$ $\quad$ $\supset \exists n_2, AKey.\, AUTH_{tgs}$

Table 2. Kerberos Security Properties

Proof Sketch. At a high level, the authentication proofs start by reasoning that a ciphertext could have been produced only by one of the possessors of the corresponding key. As an example, observe that in the first stage of Kerberos (described in section 2), the client decrypts a ciphertext encrypted with a key shared only between itself and the KAS ($k^{c \to k}_{C,K}$). Hence we can infer that one of them did the encryption. However, it is still not obvious that the client itself did not produce the ciphertext! Some other thread of the client could have potentially created the ciphertext which could have been fed back to the thread under consideration as a reflection attack. We discount this case by observing that the client role of Kerberos never encrypts with a key of type $c \to k$. This property is an *invariant* of Kerberos proved by induction over all the protocol role programs. The **HON** rule enables us to perform this induction in the proof system. Thus, so far, we have reasoned that the encryption was done by the KAS. We again observe that any role of Kerberos which does an encryption of the specific form as in stage one also sends out a message of the intended form ($AUTH_{kas}$ in table 2). This is also an invariant of Kerberos. This concludes the proof of $AUTH^{client}_{kas}$. The proof of $AUTH^{tgs}_{kas}$ follows the same high level reasoning. □

Theorem 8 (Authentication Key Secrecy). *On execution of the* **Client** *role by a principal, secrecy of the Authentication Key is preserved assuming that the client, the KAS and the TGS are all honest. Similar results hold for principals executing the* **KAS** *and* **TGS** *roles. Formally,* $KERBEROS \vdash SEC^{client}_{akey}, SEC^{kas}_{akey}, SEC^{tgs}_{akey}$

Proof Sketch. This theorem states a secrecy property for the Authentication Key $AKey$. Observe that in the first stage, the KAS sends out $AKey$ encrypted under two different keys - $k^{c \to k}_{C,K}$ and $k^{t \to k}_{T,K}$, and the client uses $AKey$ as an encryption key. As a first approximation we conjecture that in the entire protocol execution, $AKey$ is either protected by encryption with either of the keys in $\mathcal{K} = \{k^{c \to k}_{C,K}, k^{t \to k}_{T,K}\}$ or else used as an encryption key in messages sent to the network by honest principals. This

seems like a claim to be established by induction. As a base case, we establish that the generator of $AKey$ (some thread of the KAS) satisfies the conjecture. The induction case is: whenever an honest principal decrypts a ciphertext with one of the keys in $\mathcal{K}$, it ensures that new terms generated from the decryption are re-encrypted with some key in $\mathcal{K}$ in any message sent out.

When we are reasoning from the point of view of the KAS (as in SEC^{kas}_{akey}), we already know the initial condition - that the KAS sent out $AKey$ encrypted under only these keys. However, when arguing from the point of view of the client and the TGS (as in SEC^{client}_{akey} and SEC^{tgs}_{akey}), we need to have some authentication conditions established first. These conditions are generally of the form that the KAS indeed behaved in the expected manner. Reasoning from this premise, it turns out that our initial conjecture is correct.

In the formal proof, we show that Kerberos is safe with respect to the nonce $AKey$ and the set of keys $\mathcal{K}$. The induction idea is captured, in its simplest form, by the proof rule **NET**. However, as Kerberos has a staged structure we use the staged composition theorem (theorem 6) which builds upon the rule **NET**. The core of the proof is the *secrecy induction* which is an induction over all the basic sequences of all the protocol roles. The authentication condition Φ is easily derived from the KAS Authentication theorem (theorem 7). The staged composition theorem allows us to facilitate the secrecy induction by obtaining inferences from the information flow induced by the staged structure of Kerberos in a simple and effective way. The secrecy induction is modular as the individual basic sequences are small in themselves. Secrecy of $AKey$ now follows from by the axiom **POS**. □

Theorem 9 (TGS Authentication). *On execution of the* **Client** *role by a principal it is guaranteed that the intended TGS indeed sent the expected response assuming that the client, the KAS and the TGS are all honest. Similar result holds for a principal executing the* **Server** *role. Formally,* $KERBEROS \vdash AUTH^{client}_{tgs}, AUTH^{server}_{tgs}$

Proof Sketch. The proof of $AUTH^{server}_{tgs}$ is very similar to the proof for theorem 7 . The proof of $AUTH^{client}_{tgs}$ uses the secrecy property SEC^{client}_{akey} established in theorem 8 . At a high level, the client reasons that since $AKey$ is known only to $\hat{C}, \hat{K}$ and $\hat{T}$, the term $E_{sym}[AKey](SKey.n_2.\hat{S})$ - which it receives during the protocol execution - could only have been computed by one of them. Some non-trivial technical effort is required to prove that this encryption was indeed done by a thread of $\hat{T}$ and not by any thread of $\hat{C}$ or $\hat{K}$, which could have been the case if *e.g.*, there existed a reflection attack. After showing that it was indeed a thread of $\hat{T}$ who encrypted the term, we use the honesty rule to show that it indeed sent the expected response to C's message. □

Theorem 10 (Service Key Secrecy). *On execution of the* **Client** *role by a principal, secrecy of the Service Key is preserved assuming that the client, the KAS, the TGS and the application server are all honest. Similar result holds for a principal executing the* **TGS** *role. Formally,* $KERBEROS \vdash SEC^{client}_{skey}, SEC^{tgs}_{skey}$

Proof Sketch. The idea here is that the Service Key $SKey$ is protected by the key-set $\{k^{s \rightarrow t}_{S,T}, AKey\}$. The proof of this theorem follows the same high level steps as the proof of theorem 8. □

Kerberos with PKINIT We prove theorems for Kerberos with PKINIT [20] that are analogous to theorems 7-10. In the first stage of Kerberos with PKINIT, the

KAS establishes the authorization key encrypted with a symmetric key which in turn is sent to the client encrypted with its public key. For client $\hat{C}$ and KAS $\hat{K}$ let us denote this symmetric key by $k_{C,K}^{pkinit}$. Since the structure of the rest of the protocol remains the same with respect to the level of formalization in this paper [7], we can take advantage of the PCL proofs for the symmetric key version. In particular, the proofs for the properties of Kerberos with PKINIT analogous to $AUTH_{kas}^{tgs}$, $AUTH_{tgs}^{client}$ and $AUTH_{tgs}^{server}$ are identical in structure to the symmetric key version. The proof of the property corresponding to $AUTH_{kas}^{client}$ is different because of the differing message formats in the first stage. There is an additional step of proving the secrecy of $k_{C,K}^{pkinit}$, after which the secrecy proofs of $AKey$ and $SKey$ are reused with only the induction over the first stage of the client and the KAS being redone.

7 Conclusion

We present formal axioms and proof rules for inductive reasoning about secrecy and prove soundness of this system over a conventional symbolic model of protocol execution. The proof system uses a *safe message* predicate to express that any secret conveyed by the message is protected by a key from a chosen list. This predicate allows us to define two additional concepts: a principal *sends safe messages* if every message it sends is safe, and the *network is safe* if every message sent by every principal is safe. Our main inductive rule for secrecy, **NET**, states that if every honest principal preserves safety of the network, then the network is safe, assuming that only honest principals have access to keys in the chosen list. The remainder of the system makes it possible to discharge assumptions used in the proof, and prove (when appropriate) that only honest principals have the chosen keys. While it might initially seem that network safety depends on the actions of malicious agents, a fundamental advantage of Protocol Composition Logic is that proofs only involve induction over protocol steps executed by honest parties.

We illustrate the expressiveness of the logic by proving properties of two protocols: A variant of the Needham-Schroeder protocol that illustrates the ability to reason about temporary secrets, and Kerberos V5. The modular nature of the secrecy and authentication proofs for Kerberos makes it possible to reuse proofs about the basic version of the protocol for the PKINIT version that uses public-key infrastructure instead of shared secret keys in the initial steps. Compositional secrecy proofs are made possible by the composition theorems developed in this paper.

In an as-yet unpublished result, we have also developed a proof system for secrecy analysis that is sound over a computational cryptographic protocol execution model. While the Kerberos proofs are similar in that proof system, we have been unable to formulate computationally sound proofs of NSL and variants.

References

1. IEEE P802.11i/D10.0. Medium Access Control (MAC) security enhancements, amendment 6 to IEEE Standard for local and metropolitan area networks part 11: Wireless Medium Access Control (MAC) and Physical Layer (PHY) specifications., April 2004.
2. M. Backes, I. Cervesato, A. D. Jaggard, A. Scedrov, and J.-K. Tsay. Cryptographically sound security proofs for basic and public-key kerberos. In *Proceedings of 11th European Symposium on Research in Computer Security*, 2006. To appear.

3. G. Bella and L. C. Paulson. Kerberos version IV: Inductive analysis of the secrecy goals. In J.-J. Quisquater, editor, *Proceedings of the 5th European Symposium on Research in Computer Security*, pages 361–375, Louvain-la-Neuve, Belgium, Sept. 1998. Springer-Verlag LNCS 1485.
4. F. Butler, I. Cervesato, A. D. Jaggard, and A. Scedrov. A Formal Analysis of Some Properties of Kerberos 5 Using MSR. In *Fifteenth Computer Security Foundations Workshop — CSFW-15*, pages 175–190, Cape Breton, NS, Canada, 24–26 June 2002. IEEE Computer Society Press.
5. F. Butler, I. Cervesato, A. D. Jaggard, and A. Scedrov. Verifying confidentiality and authentication in kerberos 5. In *ISSS*, pages 1–24, 2003.
6. E. C. Kaufman. Internet Key Exchange (IKEv2) Protocol, 2005. RFC 4306.
7. I. Cervesato, A. Jaggard, A. Scedrov, J.-K. Tsay, and C. Walstad. Breaking and fixing public-key kerberos. Technical report.
8. A. Datta, A. Derek, J. C. Mitchell, and D. Pavlovic. Secure protocol composition. In *Proceedings of ACM Workshop on Formal Methods in Security Engineering*, October.
9. A. Datta, A. Derek, J. C. Mitchell, and D. Pavlovic. A derivation system for security protocols and its logical formalization. In *Proceedings of 16th IEEE Computer Security Foundations Workshop*, pages 109–125. IEEE, 2003.
10. A. Datta, A. Derek, J. C. Mitchell, and D. Pavlovic. Secure protocol composition. In *Proceedings of 19th Annual Conference on Mathematical Foundations of Programming Semantics*, volume 83. Electronic Notes in Theoretical Computer Science, 2004.
11. A. Datta, A. Derek, J. C. Mitchell, and D. Pavlovic. A derivation system and compositional logic for security protocols. *Journal of Computer Security*, 13:423–482, 2005.
12. R. Delicata and S. Schneider. Temporal rank functions for forward secrecy. In *18th IEEE Computer Security Foundations Workshop, (CSFW-18 2005)*, pages 126–139. IEEE Computer Society, 2005.
13. R. Delicata and S. A. Schneider. Towards the rank function verification of protocols that use temporary secrets. In *Proceedings of the Workshop on Issues in the Theory of Security: WITS '04*, 2004.
14. N. Durgin, J. C. Mitchell, and D. Pavlovic. A compositional logic for protocol correctness. In *Proceedings of 14th IEEE Computer Security Foundations Workshop*, pages 241–255. IEEE, 2001.
15. C. He, M. Sundararajan, A. Datta, A. Derek, and J. C. Mitchell. A modular correctness proof of ieee 802.11i and tls. In *ACM Conference on Computer and Communications Security*, pages 2–15, 2005.
16. J. Heather. Strand spaces and rank functions: More than distant cousins. In *Proceedings of the 15th IEEE Computer Security Foundations Workshop (CSFW'02)*, page 104, 2002.
17. J. Kohl and B. Neuman. The kerberos network authentication service, 1991. RFC 1510.
18. S. Schneider. Verifying authentication protocols with csp. *IEEE Transactions on Software Engineering*, pages 741–58, 1998.
19. F. J. Thayer, J. C. Herzog, and J. D. Guttman. Strand spaces: Proving security protocols correct. *Journal of Computer Security*, 7(1), 1999.
20. L. Zhu and B. Tung. Public key cryptography for initial authentication in kerberos, 2006. Internet Draft.