Secrecy Analysis in Protocol Composition Logic

Arnab ROY [a], Anupam DATTA [b], Ante DEREK [c], John C. MITCHELL [a], and Jean-Pierre SEIFERT [d]

[a] *Stanford University, USA*
[b] *Carnegie Mellon University, USA*
[c] *Google Corporation, USA*
[d] *University of Innsbruck, Austria*

Abstract. We present formal proof rules for inductive reasoning about the way that data transmitted on the network remains secret from a malicious attacker. 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 make 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.

Keywords. Security protocol analysis, Logic, Secrecy

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 arbitrarily many agents and 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) [14,8,9,10,11], 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. Among other challenges, the inductive secrecy rule presented here is carefully designed to be sound for reasoning about arbitrarily many simultaneous protocols 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" [20] and work using the strand space execution model [21], 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. From a broader point of view, we hope that our formal logic will help clearly identify the vocabulary, concepts, and forms of reasoning that are most effective for proving security properties of large-scale practical protocols.

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 V5 [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. While lengthy machine-checked proofs of Kerberos were previously given [3], and non-formal mathematical proofs have been developed for other abstractions of Kerberos [5], this is the first concise formal logic proof of secrecy and authentication for Kerberos and PKINIT.

Compositional secrecy proofs are made possible by the composition theorems developed in this paper. While these theorems resemble composition theorems for the simpler proof system presented in earlier work [10,15], adapting that approach for reasoning about secrecy requires new insights. For example, while proving that a protocol step does not violate secrecy, it is sometimes necessary to use information from earlier steps. This history information, which was not necessary in our earlier proofs of authentication properties, appears as preconditions in the secrecy induction of the sequential and staged composition theorems.

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. Related work is summarized in section 7 with conclusions in section 8.

2. Background

Protocol Composition Logic (PCL) is developed in [14,8,9,10], with [11] providing a relatively succinct overview 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). A principal can execute one or more copies of each role, concurrently. We use the word *thread* to refer to a principal executing a particular instance of a role. A *thread* X is identified with 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 concurrent execution of one or more instances of the protocol. Table 1 describes the syntax of the fragment of the logic that we will need 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. Many protocol properties are naturally expressible in this form.

The formulas of the logic are interpreted over protocol runs containing actions of honest parties executing roles of the protocol and a *Dolev-Yao* attacker (whose possible actions are define by a set of symbolic computation rules). We say that protocol $\mathcal{Q}$ satisfies formula ϕ, denoted $\mathcal{Q} \vDash \phi$, if in all runs R of $\mathcal{Q}$ the formula ϕ holds, *i.e.*, $\mathcal{Q}, R \vDash \phi$. For each run, satisfaction of a formula is defined inductively. 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 $\hat{X}$ acts honestly, *i.e.*, the actions of every thread of $\hat{X}$ precisely follow 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 possesses term t. This is "possess" in the symbolic sense of computing the term t using 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 establishes 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.

Action formulas

$\mathsf{a} ::= \mathsf{Start}(X) \mid \mathsf{Send}(X,t) \mid \mathsf{Receive}(X,t) \mid \mathsf{New}(X,t) \mid \mathsf{SymEnc}(X,t,k) \mid \mathsf{PkEnc}(X,t,k) \mid$
$\mathsf{SymDec}(X,t,k) \mid \mathsf{PkDec}(X,t,k) \mid \mathsf{Sign}(X,t,k) \mid \mathsf{Verify}(X,t,k) \mid \mathsf{Hash}(X,t,k)$

Formulas

$\phi ::= \mathsf{a} \mid \mathsf{Has}(X,t) \mid \mathsf{Honest}(\hat{X}) \mid \phi \wedge \phi \mid \neg\phi \mid \exists V.\ \phi$

Modal form

$\Psi ::= \phi\ [Actions]_X\ \phi$

Table 1. Syntax of the logic

Kerberos has four roles, one for each kind of participant - **Client**, **KAS**, **TGS** and **Server**. The long-term shared symmetric keys are written here in the form $k^{type}_{X,Y}$ 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 below is based on the A level formalization of Kerberos V5 in [5].

In the first stage, 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 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 message portion encrypted with $k^{t\rightarrow k}_{T,K}$ is called the *ticket granting ticket* (tgt). The client extracts $AKey$ by decrypting the component encrypted with $k^{c\rightarrow k}_{C,K}$ and using a `match` actions to separate $AKey$ from the nonce and $\hat{T}$.

In the second stage, the client generates another nonce, encrypts its identity with the session key established in stage one and sends it to the TGS along with the ticket granting ticket and the nonce. The TGS decrypts tgt with the key it shares with KAS and extracts the session key. It then uses the session key to decrypt the client's encryption and matches this with the identity of the client. The TGS then generates a new nonce to be used as a session key between the client and the application server. It then sends this key along with some other fields to the client encrypted under two different keys - the session key derived in the first stage and one it shares with the aplication server. The encryption with later key is called the service ticket (st). The client extracts this new session key by decrypting the component encrypted with the previous session key.

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 if an error occurs.

$\mathbf{Client} = (C, \hat{K}, \hat{T}, \hat{S}, t)$ [	$\mathbf{KAS} = (K)$ [
new n_1;	receive $\hat{C}.\hat{T}.n_1$;
send $\hat{C}.\hat{T}.n_1$;	new $AKey$;
	$tgt :=$ symenc $AKey.\hat{C}, k^{t\to k}_{T,K}$;
receive $\hat{C}.tgt.enc_{kc}$;	$enc_{kc} :=$ symenc $AKey.n_1.\hat{T}, k^{c\to k}_{C,K}$;
$text_{kc} :=$ symdec $enc_{kc}, k^{c\to k}_{C,K}$;	send $\hat{C}.tgt.enc_{kc}$;
match $text_{kc}$ as $AKey.n_1.\hat{T}$;	$]_K$
	$\mathbf{TGS} = (T, \hat{K})$ [
$\cdots stage\ boundary \cdots$	receive $tgt.enc_{ct}.\hat{C}.\hat{S}.n_2$;
	$text_{tgt} :=$ symdec $tgt, k^{t\to k}_{T,K}$;
new n_2;	match $text_{tgt}$ as $AKey.\hat{C}$;
$enc_{ct} :=$ symenc $\hat{C}, AKey$;	$text_{ct} :=$ symdec $enc_{ct}, AKey$;
send $tgt.enc_{ct}.\hat{C}.\hat{S}, n_2$;	match $text_{ct}$ as $\hat{C}$;
	new $SKey$;
receive $\hat{C}.st.enc_{tc}$;	$st :=$ symenc $SKey.\hat{C}, k^{s\to t}_{S,T}$;
$text_{tc} :=$ symdec $enc_{tc}, AKey$;	$enc_{tc} :=$ symenc $SKey.n_2.\hat{S}, AKey$;
match $text_{tc}$ as $SKey.n_2.\hat{S}$;	send $\hat{C}.st.enc_{tc}$;
	$]_T$
$\cdots stage\ boundary \cdots$	$\mathbf{Server} = (S, \hat{T})$ [
	receive $st.enc_{cs}$;
$enc_{cs} :=$ symenc $\hat{C}.t, SKey$;	$text_{st} :=$ symdec $st, k^{s\to t}_{S,T}$;
send $st.enc_{cs}$;	match $text_{st}$ as $SKey.\hat{C}$;
	$text_{cs} :=$ symdec $enc_{cs}, SKey$;
receive enc_{sc};	match $text_{cs}$ as $\hat{C}.t$;
$text_{sc} :=$ symdec $enc_{sc}, SKey$;	$enc_{sc} :=$ symenc $t, SKey$;
match $text_{sc}$ as t;	send enc_{sc};
$]_C$	$]_S$

Table 2. Formal description of Kerberos V5, with $\cdots stage\ boundary \cdots$ comments.

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, 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 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 by induction on i as follows:*

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

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.

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

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

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 (explained in Section 3). 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 [11].

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 $\mathsf{SendsSafeMsg}(X,s,\mathcal{K})\ [\mathsf{a}]_X\ \mathsf{SendsSafeMsg}(X,s,\mathcal{K})$, where 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.

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

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

The collection of new axioms and rules are summarized in Appendix B. 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 (see Appendix A) 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 $\mathcal{Q}$.

In the following theorem and proof, the closure $\tilde{\mathcal{M}}$ of a set $\mathcal{M}$ of messages is the least 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 the closure $\tilde{\mathcal{M}}$ contains only safe messages.*

Proof. Since $\tilde{\mathcal{M}}$ is the minimal set satisfying the given conditions, 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 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 all the messages it initially knows and has received are safe messages. Since it does not have a key in $\mathcal{K}$, the list of operations in theorem 1 enumerates a superset of all the operations it can do on this initial safe set (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 cases are sketched below, after the following definition.

A *prefix closed* formula Φ is a formula such that if a run R of a protocol $\mathcal{Q}$ satisfies Φ then any prefix of R also satisfies Φ. For example, the formula $\neg\mathsf{Send}(X, t)$ is prefix closed. This is because if in any run R, thread X has not sent the term t, it cannot have sent t in any prefix of R. In general, the negation of any action formula is prefix closed. Another example is $\forall X.\ \mathsf{New}(X, s) \supset \hat{X} = \hat{A}$ because this can be re-written as $\forall X.\ \neg\mathsf{New}(X, s) \vee \hat{X} = \hat{A}$ which is a disjunction of the negation of an action formula and an equality constraint.

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.

More formally, let us have:

1. $\mathcal{Q}, R \vDash \mathsf{KOHonest}(s, \mathcal{K}) \wedge \Phi$
2. $\mathcal{Q}, R \nvDash \mathsf{SafeNet}(s, \mathcal{K})$

Condition 2 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:

- $\mathcal{Q}, R_0 \vDash \mathsf{SafeNet}(s, \mathcal{K})$
- $R_1 = \langle([\texttt{receive}\ \ x; S']_Y \mid [\texttt{send}\ \ \tilde{m}; T']_X \longrightarrow [S'(\tilde{m}/x)]_Y \mid [T']_X)\rangle$

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 $\mathcal{Q}$. That is, $R = R'_0.R'_1.R'_2$ such that R'_0 is a prefix of R_0, P matches $R'_1|_X$ with substituition σ and R'_2 is the rest of R. So we have:

- P matches $R'_1|_X$ with substituition σ
- $\mathcal{Q}, R'_0 \vDash \mathsf{SafeNet}(s, \mathcal{K})$
- $\mathcal{Q}, R'_0.R'_1 \vDash \mathsf{Honest}(\hat{X}) \wedge \Phi$, since Φ is prefix closed.
- $\mathcal{Q}, R'_0.R'_1 \nvDash \mathsf{SendsSafeMsg}(X, s, \mathcal{K})$

Hence, we have: $\mathcal{Q}, R \nvDash \mathsf{SafeNet}(s, \mathcal{K})[P]_X \mathsf{Honest}(\hat{X}) \wedge \Phi \supset \mathsf{SendsSafeMsg}(X, s, \mathcal{K})$, thus violating the premise.

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**.

Formally, assume that the following formula is valid:

$$\mathcal{P} : \psi \wedge \mathsf{SafeNet}(s, \mathcal{K})\,[S]_X\,\mathsf{SendsSafeMsg}(X, s, \mathcal{K}) \wedge \mathsf{Has}(Y, M) \wedge \neg\mathsf{SafeMsg}(M, s, \mathcal{K})$$

Consider R, an arbitrary run of the protocol $\mathcal{Q}$ such that $R = R_0.R_1.R_2$ and the following conditions hold:

1. S matches $R_1|_X$ with substituition σ.
2. $\mathcal{Q}, R_0 \models \sigma(\psi \wedge \mathsf{SafeNet}(s, \mathcal{K}))$

Therefore, from the validity of $\mathcal{P}$ we have:

$$\mathcal{Q}, R_0.R_1 \models \sigma(\mathsf{SendsSafeMsg}(X, s, \mathcal{K}) \wedge \mathsf{Has}(Y, M) \wedge \neg\mathsf{SafeMsg}(M, s, \mathcal{K}))$$

Now, we construct a run $R'_1 \cong \sigma S$, that is, R'_1 has only actions of the thread X (any send/receive by X is with a buffer chord). Since the conditions 1 and 2 still hold for the run $R' = R_0.R'_1.R_2$, we have:

$$\mathcal{Q}, R_0.R_1' \models \sigma(\mathsf{SendsSafeMsg}(X, s, \mathcal{K}) \wedge \mathsf{Has}(Y, M) \wedge \neg\mathsf{SafeMsg}(M, s, \mathcal{K}))$$

We have two cases here: $Y = X$ or, $Y \neq X$. In the first case, since $\mathcal{Q}, R_0 \models \sigma\mathsf{SafeNet}(s, \mathcal{K})$ and $[S]_X$ can receive at most once - just after R_0, therefore, if thread X possesses an unsafe message then $\sigma(\exists k \in \mathcal{K}.\ \mathsf{Has}(X, k) \vee \mathsf{New}(X, s))$ - and this fact cannot be altered by further actions of X.

In the second case, we observe that R_1' does not contain the action of any thread other than X, excepting receipt of the messages sent by X, which are safe anyway. Therefore, $\mathcal{Q}, R_0 \models \sigma(\mathsf{Has}(Y, M) \wedge \neg\mathsf{SafeMsg}(M, s, \mathcal{K}))$. From this, condition 2 and **POS** we have: $\mathcal{Q}, R_0 \models \sigma(\exists k \in \mathcal{K}.\ \mathsf{Has}(Y, k) \vee \mathsf{New}(Y, s))$. Again, further actions by any thread after R_0 cannot alter this fact. Therefore, $\mathcal{Q}, R_0.R_1 \models \sigma(\exists k \in \mathcal{K}.\mathsf{Has}(Y, k) \vee \mathsf{New}(Y, s))$.

Hence, for all runs R the following formula holds:

$$\mathcal{Q}, R \models \psi \wedge \mathsf{SafeNet}(s, \mathcal{K})\ [S]_X\ \exists k \in \mathcal{K}.\ \mathsf{Has}(Y, k) \vee \mathsf{New}(Y, s)$$

□

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 in Figure 3.

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:

$\mathbf{Init} = (A, \hat{B}, m)$ [	$\mathbf{Resp} = (B)$ [
new n_a;	receive enc_{r1};
$enc_{r1} :=$ pkenc $\hat{A}.n_a, \hat{B}$;	$text_{r1} :=$ pkdec $enc_{r1}, \hat{B}$;
send enc_{r1};	match $text_{r1}$ as $\hat{A}.n_a$;
	new k;
receive enc_i;	$enc_i :=$ pkenc $n_a.\hat{B}.k, \hat{A}$;
$text_i :=$ pkdec $enc_i, \hat{A}$;	send enc_i;
match $text_i$ as $n_a.\hat{B}.k$;	
$enc_{r2} :=$ symenc m, k;	
	receive $enc_{r2}.n_a$;
send $enc_{r2}.n_a$;	$m :=$ symdec enc_{r2}, k;
$]_A$	$]_B$

Table 3. Formal description of *NSLVAR*

$$\Phi_1 : \forall X, \hat{Y}.\ \mathsf{New}(X, n_a) \wedge \mathsf{PkEnc}(X, \hat{X}.n_a, \hat{Y}) \supset \hat{Y} = \hat{B}$$

$$\Phi_2 : \forall X, \hat{Y}, n.\ \mathsf{New}(X, n_a) \supset \neg\mathsf{PkEnc}(X, n.\hat{X}.n_a, \hat{Y})$$

$$\Phi_3 : \forall X, e.\ \mathsf{New}(X, n_a) \supset \neg\mathsf{Send}(X, e.n_a)$$

$$\Phi_4 : \mathsf{Honest}(\hat{X}) \wedge \mathsf{Send}(X, e.n) \supset \mathsf{New}(X, n)$$

$$\Phi_5 : \mathsf{Honest}(\hat{X}) \wedge \mathsf{PkEnc}(X, \hat{X}'.n, \hat{Y}) \supset \hat{X}' = \hat{X}$$

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 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:

$$\mathsf{SafeNet}(n_a, \mathcal{K})[\mathbf{P}]_{A'}\ \mathsf{Honest}(\hat{A}') \wedge \Phi \supset \mathsf{SendsSafeMsg}(A', n_a, \mathcal{K})$$

	$[\tilde{\mathbf{Init}}]_A\ \mathsf{New}(A, n_a)$	(1)
$(-1), \mathbf{N1}$	$[\tilde{\mathbf{Init}}]_A\ \mathsf{New}(X, n_a) \supset X = A$	(2)
	$\mathsf{Start}(A)[]_A\ \neg\mathsf{PkEnc}(A, \hat{A}.n_a, \hat{Y}) \vee \hat{Y} = \hat{B}$	(3)
	$\neg\mathsf{PkEnc}(A, \hat{A}.n_a, \hat{Y}) \vee \hat{Y} = \hat{B}\ [\mathtt{new}\ \ n_a;]_A\ \neg\mathsf{PkEnc}(A, \hat{A}.n_a, \hat{Y}) \vee \hat{Y} = \hat{B}$	(4)
	$\top\ [enc_{r1} := \mathtt{pkenc}\ \ \hat{A}.n_a, \hat{B};]_A\ \mathsf{PkEnc}(A, \hat{A}.n_a, \hat{B})$	(5)
	$\neg\mathsf{PkEnc}(A, \hat{A}.n_a, \hat{Y}) \vee \hat{Y} = \hat{B}\ [\mathtt{send}\ \ enc_{r1};$ $\quad \mathtt{receive}\ \ enc_i;$ $\quad text_i := \mathtt{pkdec}\ \ enc_i, \hat{A};$ $\quad \mathtt{match}\ \ text_i\ \mathtt{as}\ \ n_a.\hat{B}.k;$ $\quad enc_{r2} := \mathtt{symenc}\ \ m, k;]_A\ \neg\mathsf{PkEnc}(A, \hat{A}.n_a, \hat{Y}) \vee \hat{Y} = \hat{B}$	(6)
	$[\tilde{\mathbf{Init}}]_A\ \mathsf{PkEnc}(A, \hat{A}.n_a, \hat{Y}) \supset \hat{Y} = \hat{B}$	(7)
(-1)	$[\tilde{\mathbf{Init}}]_A\ \Phi_1$	(8)

Table 4. Formal proof of $[\tilde{\mathbf{Init}}]_A\ \Phi_1$

The formal proof is done in Appendix C. The variables used in the basic sequence we are inducting over are consistently primed so that we do not capture variables in Φ, n_a or $\mathcal{K}$. Finally, we use the **NET** rule and **POS** axiom to show that n_a is a shared secret between A and B at a state where A has just finished executing $\tilde{\mathbf{Init}}$. □

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 [10,15]. However, adapting that approach for reasoning about secrecy requires new insights. 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 [10]), 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 $\mathcal{Q}_1 \mid \mathcal{Q}_2$ 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 $\mathcal{Q}_1 \mid \mathcal{Q}_2 \vdash \Psi$, 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. [10]).

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)*
   - $\mathcal{Q}_1 \mid \mathcal{Q}_2 \vdash \mathsf{Start}(X) \supset \theta_{P_1}$ *and* $\mathcal{Q}_1 \mid \mathcal{Q}_2 \vdash \theta_{P_1} [P_1]_X\ \theta_{P_2}$
   - $\forall i. \forall S \in BS(P_i).\ \theta_{P_i} [S]_X\ \theta_{P_i}$.

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.

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)*
   - $\mathcal{Q}_1 \mid \mathcal{Q}_2 \cdots \mid \mathcal{Q}_n \vdash \mathsf{Start}(X) \supset \theta_{P_1}$ *and* $\mathcal{Q}_1 \mid \mathcal{Q}_2 \cdots \mid \mathcal{Q}_n \vdash \forall i.\ \theta_{P_i}[P_i]_X\ \theta_{P_{i+1}}$
   - $\forall i. \forall S \in \bigcup_{j \geq i} BS(P_j).\ \theta_{P_i}[S]_X\ \theta_{P_i}$.

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.

6. Analysis of Kerberos V5

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 5. 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.

$$SEC_{akey} : \mathsf{Hon}(\hat{C}, \hat{K}, \hat{T}) \supset (\mathsf{Has}(X, AKey) \supset \hat{X} \in \{\hat{C}, \hat{K}, \hat{T}\})$$

$$SEC_{skey} : \mathsf{Hon}(\hat{C}, \hat{K}, \hat{T}, \hat{S}) \supset (\mathsf{Has}(X, SKey) \supset \hat{X} \in \{\hat{C}, \hat{K}, \hat{T}, \hat{S}\})$$

$$AUTH_{kas} : \exists \eta.\ \mathsf{Send}((\hat{K}, \eta), \hat{C}.E_{sym}[k^{t \to k}_{T,K}](AKey.\hat{C}).E_{sym}[k^{c \to k}_{C,K}](AKey.n_1.\hat{T}))$$

$$AUTH_{tgs} : \exists \eta.\ \mathsf{Send}((\hat{T}, \eta), \hat{C}.E_{sym}[k^{s \to t}_{S,T}](SKey.\hat{C}).E_{sym}[AKey](SKey.n_2.\hat{S}))$$

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

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

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

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

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

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

Table 5. Kerberos Security Properties

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* $\vdash AUTH^{client}_{kas}, AUTH^{tgs}_{kas}$

Proof Sketch. In the course of executing the **Client** role, principal $\hat{C}$ receives a message containing the encrypted term $E_{sym}[k^{c \to k}_{C,K}](AKey.n_1.\hat{T})$. Using axiom **ENC4**, we derive that this message was encrypted by one of the owners of $k^{c \to k}_{C,K}$, which is either $\hat{C}$ or $\hat{K}$. Then, by using the rule **HON** we establish that no thread of $\hat{C}$ does this unless $\hat{C} = \hat{K}$, and so this must be some thread of $\hat{K}$. Once again we use the **HON** rule to reason that if an honest thread encrypts a message of this form then it also sends out a message of the form described in $AUTH_{kas}$. The proof of $AUTH^{tgs}_{kas}$ is along identical lines. In Appendix D.2, we first give a *template* proof for the underlying reasoning and then instantiate it for both $AUTH^{client}_{kas}$ and $AUTH^{tgs}_{kas}$. $\square$

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. In Appendix D.3 we formally prove the secrecy of the session key $AKey$ with respect to the key-set $\mathcal{K} = \{k^{c \to k}_{C,K}, k^{t \to k}_{T,K}\}$. The proof is modular and broadly, there are two stages to the proof:

1. In the first stage we assume certain conditions, denoted Φ, and the honesty of principals $\hat{C}, \hat{K}$ and $\hat{T}$ and prove that this implies $\mathsf{SafeNet}(AKey, \mathcal{K})$. The proof of this part uses the Staged Composition Theorem. The components of this proof are:

   - secrecy induction - we will describe this shortly.

- precondition induction - in case of *KERBEROS* most basic sequences do not need any precondition to facilitate the secrecy induction. For two of the basic sequences in the Client program, the preconditions are simply of the form that a certain message was received. Since receiving a message is a monotonic property, that is - once it is true it is always true thereafter - the precondition induction goes through simply.

2. In the second stage we prove that execution of the $\mathbf{Client}, \mathbf{KAS}$ or the $\mathbf{TGS}$ roles discharge the assumptions Φ. These proofs are derived from the authentication properties $AUTH_{kas}^{client}, AUTH_{kas}^{tgs}$. Now we combine the two derivations, use the $\mathbf{POS}$ axiom and conclude $SEC_{akey}^{client}, SEC_{akey}^{kas}$ and SEC_{akey}^{tgs}.

As the form of the secrecy induction suggests, we do an induction over all the basic sequences of *KERBEROS*. Broadly, the induction uses a combination of the following types of reasoning:

- The secrecy axioms enumerated in the proof system section. The structure of Kerberos suggests that in many of the basic sequences the messages being sent out are functions of messages received. A key strategy here is to use $\mathbf{NET1}$ and the safe network hypothesis to derive that the message received is safe and then proceed to prove that the messages being sent out are also safe. Consider as an example the sequence of actions by an application server thread $[\mathbf{Server}]_{S'}$: S' receives a message $E_{sym}[SKey'](\hat{C'}.t')$ and sends out a message $E_{sym}[SKey'](t')$. It is provable, just by using the $\mathbf{SAF}$ axioms that the later message is safe if the former message is safe.

- Derivations from Φ: The structure of Φ is dictated by the structure of the basic sequences we are inducing over. 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. In case of *KERBEROS*, these parts are formulae that state that the generating thread of the putative secret $AKey$ did not perform certain types of action on $AKey$ or did it in a restricted form. 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 $\Phi_1 : \forall X, M.\ \mathsf{New}(X, AKey) \supset \neg(\mathsf{Send}(X, M) \wedge \mathsf{ContainsOpen}(M, AKey))$ - this tells us that the generator of $AKey$ did not send it out unprotected in the open.

- Derivations from the θ's, that is, the preconditions. These are conditions which are true at the beginning of the basic sequence we are inducing over with respect to the staged control flow that *KERBEROS* exhibits. As before, a practical proof strategy is to find out what precondition we need for the secrecy induction and do the precondition induction part afterwards. Consider for example the end of the first stage of the client thread $[\mathbf{Client}]_{C'}$. We know that at the beginning of the second stage the following formula always holds - $\theta : \mathsf{Receive}(\hat{C'}, tgt'.E_{sym}[k_{C',K'}^{c \rightarrow k}](AKey'.n_1'.\hat{T'}))$. The reason this information is necessary is that the second stage sends out tgt' in the open - in order to reason that this send is safe, given the safe network hypothesis at the beginning of the second stage, we use the precondition and the theorem $\mathbf{SREC}$ to derive that tgt' was safe to begin with. $\square$

Theorem 9 (TGS Authentication) *On execution of the* $\mathbf{Client}$ *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* $\vdash AUTH_{tgs}^{client}, AUTH_{tgs}^{server}$

Proof Sketch. The proof of $AUTH_{tgs}^{server}$ can be instantiated from the *template* proof for theorem 7 and is formally done in Appendix D.2. The proof of $AUTH_{tgs}^{client}$ uses the secrecy property SEC_{akey}^{client} established in theorem 8 and is formally done in Appendix D.4. 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* $\vdash SEC_{skey}^{client}, SEC_{skey}^{tgs}$

Proof Sketch. The idea here is that the Service Key $SKey$ is protected by the key-set $\{k_{S,T}^{s \rightarrow t}, AKey\}$. The proof of this theorem being very similar to the proof of theorem 8 is omitted from this paper. □

Kerberos with PKINIT

We prove theorems for Kerberos with PKINIT [22] that are analogous to theorems 7-10 and are listed in Table 6. The proofs are omitted due to space constraints. 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. Related Work

Some secrecy proofs using the CSP [20] or strand space [21] 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* [20,16], with an inductive proof used to show that rank is preserved by the attacker actions and all honest parties.

$$SEC_k : \mathsf{Hon}(\hat{C}, \hat{K}) \supset (\mathsf{GoodKeyAgainst}(X, k) \vee \hat{X} \in \{\hat{C}, \hat{K}\})$$

$$SEC_{akey} : \mathsf{Hon}(\hat{C}, \hat{K}, \hat{T}) \supset (\mathsf{GoodKeyAgainst}(X, AKey) \vee \hat{X} \in \{\hat{C}, \hat{K}, \hat{T}\})$$

$$SEC_{skey} : \mathsf{Hon}(\hat{C}, \hat{K}, \hat{T}, \hat{S}) \supset (\mathsf{GoodKeyAgainst}(X, SKey) \vee \hat{X} \in \{\hat{C}, \hat{K}, \hat{T}, \hat{S}\})$$

$$AUTH_{kas} : \exists \eta.\, \mathsf{Send}((\hat{K}, \eta), E_{pk}[pk_C](Cert_K.SIG[sk_K](k.ck)).$$
$$\hat{C}.E_{sym}[k^{t \rightarrow k}_{T,K}](AKey.\hat{C}).E_{sym}[k](AKey.n_1.t_K.\hat{T}))$$

$$AUTH_{tgs} : \exists \eta.\, \mathsf{Send}((\hat{T}, \eta), \hat{C}.E_{sym}[k^{s \rightarrow t}_{S,T}](SKey.\hat{C}).E_{sym}[AKey](SKey.n_2.\hat{S}))$$

$$SEC^{client}_k : [\mathbf{Client}]_C\ SEC_k \qquad SEC^{kas}_k : [\mathbf{KAS}]_K\ SEC_k$$

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

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

$$SEC^{tgs}_{akey} : [\mathbf{TGS}]_T\ SEC_{akey} \qquad \supset \exists n_1, k, ck, t_K.\ AUTH_{kas}$$

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

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

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

Table 6. PKINIT Security Properties

In comparison, arguments in our formal logic use a conjunction involving the 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 functions approach is that the induction is performed by "global" reasoning – trying to capture all possible properties of the system at once. This makes the method less applicable since it cannot handle protocols which deal with temporary secrets or use authentication to ensure secrecy properties. Although some of these issues can be resolved by extensions of the rank function method [13,12], we expect that the tools available in PCL are more general and may be better suited for application to real-world protocols.

Our composition theorems allow us to use a divide-and-conquer approach for complex protocols with different parts serving different purposes. By varying the preconditions of the secrecy induction in the staged composition theorem, we are essentially modifying the rank function as we shift our attention from one protocol stage to the other.

Because of its widespread deployment and relative complexity, Kerberos has been the subject of several logical studies. Bella and Paulson use automated theorem proving techniques to reason explicitly about properties of Kerberos that hold in all traces con-

taining 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 [4,2] to develop proofs in the symbolic and computational model. However, proofs in these papers use unformalized (though rigorous) mathematical arguments and are not modular.

8. 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 presented in this paper by proving properties of two protocols, a variant of the Needham-Schroeder protocol that illustrates the ability to reason about temporary secrets, and Kerberos. 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 section 5 of this paper.

We have also developed a proof system for secrecy analysis that is sound over a “computational" protocol execution model which involves probabilistic polynomial-time computation [19]. The proofs of Kerberos security properties in the computationally sound logic turn out to be syntactically analogous to the symbolic version described in this paper. However, the proofs for NSL and variants are not entirely analogous to the symbolic versions. Specifically, these proofs involve axioms capturing some subtle ways in which cryptographic reduction proofs work which do not seem to have a direct correspondence with the symbolic way of interpreting the cryptographic primitives.

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.

[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. A derivation system for security protocols and its logical formalization. In *Proceedings of 16th IEEE Computer Security Foundations Workshop*, pages 109–125. IEEE, 2003.

[9] 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.

[10] 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.

[11] A. Datta, A. Derek, J. C. Mitchell, and A. Roy. Protocol Composition Logic (PCL). *Electronic Notes in Theoretical Computer Science*, 172:311–358, 2007.

[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.

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

[19] A. Roy, A. Datta, A. Derek, and J. Mitchell. Inductive proofs of computational secrecy. In *Proc. 12th European Symposium On Research In Computer Security*, 2007.

[20] S. Schneider. Verifying authentication protocols with csp. *IEEE Transactions on Software Engineering*, pages 741–58, 1998.

[21] F. J. Thayer, J. C. Herzog, and J. D. Guttman. Strand spaces: Proving security protocols correct. *Journal of Computer Security*, 7(1), 1999.

[22] L. Zhu and B. Tung. Public key cryptography for initial authentication in kerberos, 2006. Internet Draft.

A. Protocol Logic

A.1. Axioms and Inference Rules

A representative fragment of the axioms and inference rules in the proof system are collected in Table 7. For expositional convenience, we divide the axioms into four groups.

The axioms about protocol actions state properties that hold in the state reached by executing one of the actions in a state in which formula ϕ holds. Note that the a in axiom **AA1** is any one of the actions and a is the corresponding predicate in the logic. Axiom **N1** states that two different threads cannot generate the same nonce while axiom **AN2** states that if a thread generates a nonce and does nothing else, only that thread possesses the nonce.

The possession axioms reflect a fragment of Dolev-Yao rules for constructing or decomposing messages while the encryption axioms symbolically model encryption. The generic rules are used for manipulating modal formulas.

A.2. The Honesty Rule

The honesty rule is essentially an invariance rule for proving properties of all roles of a protocol. It is similar to the basic invariance rule of LTL [18]. The honesty rule is used to combine facts about one role with inferred actions of other roles.

For example, suppose Alice receives a response from a message sent to Bob. Alice may wish to use properties of Bob's role to reason about how Bob generated his reply. In order to do so, Alice may assume that Bob is honest and derive consequences from this assumption. Since honesty, by definition in this framework, means "following one or more roles of the protocol," honest principals must satisfy every property that is a provable invariant of the protocol roles. Using the notation just introduced, the honesty rule may be written as follows.

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

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

Axioms for protocol actions

AA1 $\phi[a]_X$ 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

AN2 $\phi[\mathsf{new}\ \ x]_X\ \mathsf{Has}(Y,x) \supset (Y = X)$

ARP $\mathsf{Receive}(X,p(x))[\mathtt{match}\ \ q(x)\ \mathsf{as}\ \ q(t)]_X\ \mathsf{Receive}(X,p(t))$

P1 $\mathsf{Persist}(X,t)[a]_X\ \mathsf{Persist}(X,t)$, for $\mathsf{Persist} \in \{\mathsf{Has}, \mathsf{Send}, \mathsf{Receive}\}$

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

Possession Axioms

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

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

Encryption Axioms

Let $\mathsf{Enc} \in \{\mathsf{SymEnc}, \mathsf{PkEnc}\}$, $\mathsf{Dec} \in \{\mathsf{SymDec}, \mathsf{PkDec}\}$ in the following:

ENC0 $[m' := \mathsf{enc}\ \ m,k;]_X\ \mathsf{Enc}(X,m,k)$

ENC1 $\mathsf{Start}(X)\ [\,]_X\ \neg\mathsf{Enc}(X,m,k)$

ENC2 $\pi(X,m,k)\ [\mathsf{a}]_X\ \pi(X,m,k),$ for $\pi \in \{\mathsf{Enc}, \neg\mathsf{Enc}\}$

where, either $\mathsf{a} \neq \mathsf{enc}\ \ \cdots$ or, $\mathsf{a} = (p := \mathsf{enc}\ \ k', q)$, such that $(q,k') \neq (m,k)$

ENC3 $\mathsf{Enc}(X,m,k) \supset \mathsf{Has}(X,k) \wedge \mathsf{Has}(X,m)$

ENC4 $\mathsf{SymDec}(X, E[k](m), k) \supset \exists Y.\ \mathsf{SymEnc}(Y,m,k)$

PENC4 $\mathsf{PkDec}(X, E[k](m), \bar{k}) \supset \exists Y.\ \mathsf{PkEnc}(Y,m,k)$

Generic Rules

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

Table 7. Fragment of the Proof System

B. New Definitions, Axioms and Rules for Secrecy

$$\mathsf{SendsSafeMsg}(X,s,\mathcal{K}) \equiv \forall M.\ (\mathsf{Send}(X,M) \supset \mathsf{SafeMsg}(M,s,\mathcal{K}))$$

$$\mathsf{SafeNet}(s,\mathcal{K}) \equiv \forall X.\ \mathsf{SendsSafeMsg}(X,s,\mathcal{K})$$

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

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

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

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

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, $\mathcal{K}$, s, and Φ is prefix closed.

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

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

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.

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

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

C. PCL Proof of NSL Variant Secrecy

As in the theorem, $\tilde{\mathbf{Init}}$ is the initial segment of the $\mathbf{Init}$ role excluding the last send action. 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:

$$\Phi_1 : \forall X, \hat{Y}.\ \mathsf{New}(X, n_a) \wedge \mathsf{Send}(X, E_{pk}[k_Y](\hat{X}.n_a)) \supset \hat{Y} = \hat{B}$$

$$\Phi_2 : \forall X, \hat{Y}, n.\ \mathsf{New}(X, n_a) \supset \neg\mathsf{Send}(X, E_{pk}[k_Y](n.\hat{X}.n_a))$$

$$\Phi_3 : \forall X, e.\ \mathsf{New}(X, n_a) \supset \neg\mathsf{Send}(X, e.n_a)$$

$$\Phi_4 : \mathsf{Honest}(\hat{X}) \wedge \mathsf{Send}(X, E_{sym}[k_0](m_0).n) \supset \mathsf{New}(X, n)$$

$$\Phi_5 : \mathsf{Honest}(\hat{X}) \wedge \mathsf{PkEnc}(X, \hat{X}'.n, \hat{Y}) \supset \hat{X}' = \hat{X}$$

In the next step we prove the antecedents of the $\mathbf{NET}$ 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:

$$\mathsf{SafeNet}(n_a, \mathcal{K})[\mathbf{P}]_{A'}\ \mathsf{Honest}(\hat{A}') \wedge \Phi \supset \mathsf{SendsSafeMsg}(A', n_a, \mathcal{K})$$

The variables used in the basic sequence we are inducting over are consistently primed so that we do not capture variables in Φ, n_a or $\mathcal{K}$. Finally, we use the $\mathbf{NET}$ rule and $\mathbf{POS}$ axiom to show that n_a is a shared secret between A and B at a state where A has just finished executing $\tilde{\mathbf{Init}}$.

Let, $[\mathbf{Init}_1]_{A'}$: [new n'_a;
$enc'_{r1} :=$ pkenc $\hat{A}'.n'_a, \hat{B}'$;
send enc'_{r1};$]_{A'}$

Case 1 : $n'_a \neq n_a$ (1)

(1) $[\mathbf{Init}_1]_{A'}\ \mathsf{SafeMsg}(E_{pk}[k_{B'}](\hat{A}'.n'_a), n_a, \mathcal{K})$ (2)

(2), $\mathbf{NET}*$ $\mathsf{SafeNet}(n_a, \mathcal{K})[\mathbf{Init}_1]_{A'}\ \mathsf{SendsSafeMsg}(A', n_a, \mathcal{K})$ (3)

Case 2 : $n'_a = n_a$ (4)

$[\mathbf{Init}_1]_{A'}\ \mathsf{New}(A', n_a) \wedge \mathsf{Send}(A', E_{pk}[k_{B'}](\hat{A}'.n_a))$ (5)

Φ_1 $[\mathbf{Init}_1]_{A'}\ \hat{B}' = \hat{B}$ (6)

(6) $[\mathbf{Init}_1]_{A'}\ \mathsf{SafeMsg}(E_{pk}[k_{B'}](\hat{A}'.n'_a), n_a, \mathcal{K})$ (7)

(7), $\mathbf{NET}*$ $\mathsf{SafeNet}(n_a, \mathcal{K})[\mathbf{Init}_1]_{A'}\ \mathsf{SendsSafeMsg}(A', n_a, \mathcal{K})$ (8)

Let, $[\mathbf{Init}_2]_{A'}$: $[\texttt{receive}\ \ enc'_i;$

$text'_i := \texttt{pkdec}\ \ enc'_i, \hat{A}';$

$\texttt{match}\ \ text'_i\ \texttt{as}\ \ n'_a.\hat{B}'.k';$

$enc'_{r2} := \texttt{symenc}\ \ m', k';$

$\texttt{send}\ \ enc'_{r2}.n'_a;]_{A'}$

$$[\mathbf{Init}_2]_{A'}\ \mathsf{Send}(A', E_{sym}[k'](m').n'_a) \tag{9}$$

$$\Phi_4 \quad \mathsf{Honest}(\hat{X}) \wedge \mathsf{Send}(X, E_{sym}[k_0](m_0).n) \supset \mathsf{New}(X, n) \tag{10}$$

$$(9), (10) \quad [\mathbf{Init}_2]_{A'}\ \mathsf{New}(A', n'_a) \wedge \mathsf{Send}(A', E_{sym}[k'](m').n'_a) \tag{11}$$

$$\Phi_3, (11) \quad [\mathbf{Init}_2]_{A'}\ n'_a \neq n_a \tag{12}$$

$$\mathbf{SAF0}, (12) \quad [\mathbf{Init}_2]_{A'}\ \mathsf{SafeMsg}(n'_a, n_a, \mathcal{K}) \tag{13}$$

$$\mathbf{SAF0} \quad [\mathbf{Init}_2]_{A'}\ \mathsf{SafeMsg}(m', n_a, \mathcal{K}) \tag{14}$$

$$\mathbf{SAF}*, (13), (14) \quad [\mathbf{Init}_2]_{A'}\ \mathsf{SafeMsg}(E_{sym}[k'](m').n'_a, n_a, \mathcal{K}) \tag{15}$$

$$(15) \quad \mathsf{SafeNet}(n_a, \mathcal{K})\ [\mathbf{Init}_2]_{A'}\ \mathsf{SendsSafeMsg}(A', n_a, \mathcal{K}) \tag{16}$$

Let, $[\mathbf{Resp}_1]_{B'}$: $[\texttt{receive}\ \ enc'_{r1};$

$text'_{r1} := \texttt{pkdec}\ \ enc'_{r1}, \hat{B}';$

$\texttt{match}\ \ text'_{r1}\ \texttt{as}\ \ \hat{A}'.n'_a;$

$\texttt{new}\ \ k';$

$enc'_i := \texttt{pkenc}\ \ n'_a.\hat{B}'.k', \hat{A}';$

$\texttt{send}\ \ enc'_i;]_{B'}$

$$[\mathbf{Resp}_1]_{B'}\ \mathsf{New}(B', k') \wedge \mathsf{Send}(B', E_{pk}[k_{A'}](n'_a.\hat{B}'.k')) \tag{17}$$

$$\Phi_2, (17) \quad [\mathbf{Resp}_1]_{B'}\ k' \neq n_a \tag{18}$$

$$\text{Case 1} : \mathsf{SafeMsg}(n'_a, n_a, \mathcal{K}) \tag{19}$$

$$\mathbf{SAF}*, (18) \quad \mathsf{SafeMsg}(E_{pk}[k_{A'}](n'_a.\hat{B}'.k'), n_a, \mathcal{K}) \tag{20}$$

$$\mathbf{NET}*, (20) \quad \mathsf{SafeNet}(n_a, \mathcal{K})\ [\mathbf{Resp}_1]_{B'}\ \mathsf{SendsSafeMsg}(B', n_a, \mathcal{K}) \tag{21}$$

Case 2 : $\neg \mathsf{SafeMsg}(n'_a, n_a, \mathcal{K})$ (22)

ENC4 $[\texttt{receive}\ enc'_{r1}; \texttt{match}\ enc'_{r1}\ \texttt{as}\ E_{pk}[k_{B'}](\hat{A}'.n'_a);]_{B'}$

$\exists X.\ \mathsf{PkEnc}(X, \hat{A}'.n'_a, \hat{B}')$ (23)

Inst $X \mapsto X_0$ $[\texttt{receive}\ enc'_{r1}; \texttt{match}\ enc'_{r1}\ \texttt{as}\ E_{pk}[k_{B'}](\hat{A}'.n'_a);]_{B'}$

$\mathsf{PkEnc}(X_0, \hat{A}'.n'_a, \hat{B}')$ (24)

(24) $[\texttt{receive}\ enc'_{r1}; \texttt{match}\ enc'_{r1}\ \texttt{as}\ E_{pk}[k_{B'}](\hat{A}'.n'_a);]_{B'}$

$\mathsf{Has}(X_0, n'_a)$ (25)

NET$*$, (25) $\mathsf{SafeNet}(n_a, \mathcal{K})$

$[\texttt{receive}\ enc'_{r1}; \texttt{match}\ enc'_{r1}\ \texttt{as}\ E_{pk}[k_{B'}](\hat{A}'.n'_a);]_{B'}$

$\mathsf{SendsSafeMsg}(B', n_a, \mathcal{K}) \wedge \mathsf{Has}(X_0, n'_a) \wedge \neg \mathsf{SafeMsg}(n'_a, n_a, \mathcal{K})$ (26)

POSL, (26) $\mathsf{SafeNet}(n_a, \mathcal{K})$

$[\texttt{receive}\ enc'_{r1}; \texttt{match}\ enc'_{r1}\ \texttt{as}\ E_{pk}[k_{B'}](\hat{A}'.n'_a);]_{B'}$

$\exists k \in \mathcal{K}.\ \mathsf{Has}(X_0, k) \vee \mathsf{New}(X_0, n_a)$ (27)

(27) $\hat{X_0} = \hat{A} \vee \hat{X_0} = \hat{B}$ (28)

(28) $\mathsf{Honest}(\hat{X_0})$ (29)

Φ_5 $\mathsf{Honest}(\hat{X}) \wedge \mathsf{PkEnc}(X, \hat{X}'.n, \hat{Y}) \supset \hat{X}' = \hat{X}$ (30)

(24), (28), $\mathsf{SafeNet}(n_a, \mathcal{K})$

(29), (30) $[\texttt{receive}\ enc'_{r1}; \texttt{match}\ enc'_{r1}\ \texttt{as}\ E_{pk}[k_{B'}](\hat{A}'.n'_a);]_{B'}$

$\hat{A}' = \hat{A} \vee \hat{A}' = \hat{B}$ (31)

SAF3, (31) $\mathsf{SafeNet}(n_a, \mathcal{K})\ [\mathbf{Resp}_1]_{B'}\ \mathsf{SafeMsg}(E_{pk}[k_{A'}](n'_a.\hat{B}'.k'), n_a, \mathcal{K})$ (32)

(32) $\mathsf{SafeNet}(n_a, \mathcal{K})[\mathbf{Resp}_1]_{B'}\mathsf{SendsSafeMsg}(B', n_a, \mathcal{K})$ (33)

Let, $[\mathbf{Resp}_2]_{B'}$: $[\texttt{receive}\ enc'_{r2}.n'_a;$

$m' := \texttt{symdec}\ enc'_{r2}, k';]_{B'}$

NET$*$ $\mathsf{SafeNet}(n_a, \mathcal{K})[\mathbf{Resp}_2]_{B'}\mathsf{SendsSafeMsg}(B', n_a, \mathcal{K})$ (34)

NET $[\tilde{\mathbf{Init}}]_A\ \mathsf{Honest}(\hat{A}) \wedge \mathsf{Honest}(\hat{B}) \supset \mathsf{SafeNet}(n_a, \mathcal{K})$ (35)

POS, (35) $[\tilde{\mathbf{Init}}]_A\ \mathsf{Honest}(\hat{A}) \wedge \mathsf{Honest}(\hat{B}) \supset (\mathsf{Has}(X, n_a) \supset \hat{X} = \hat{A} \vee \hat{X} = \hat{B})$ (36)

D. Proof of Kerberos Security Properties

D.1. Environmental Assumptions

Long term symmetric keys possessed by pairs of honest principals are possessed by only themselves.

$$\Gamma_0 : \forall X, Y, Z, type.\ \mathsf{Hon}(\hat{X}, \hat{Y}) \wedge \mathsf{Has}(Z, k^{type}_{X,Y}) \supset (\hat{Z} = \hat{X} \vee \hat{Z} = \hat{Y})$$

D.2. Proofs of $AUTH^{client}_{kas}$, $AUTH^{tgs}_{kas}$ and $AUTH^{server}_{tgs}$

Below we give a *template* proof of $[\mathbf{Role}]_X\ \mathsf{Hon}(\hat{X}, \hat{Y}) \supset \exists \eta.\ \mathsf{SymEnc}((\hat{Y}, \eta), M_1, k^{type}_{X,Y})$, where $\mathbf{Role}$ receives the message $M_0.E_{sym}[k^{type}_{X,Y}](M_1).M_2$.

Reference to equations by negative numbers is relative to the current equation - *e.g.*, (-1) refers to the last equation. Reference by positive number indicates the actual number of the equation.

$$[\mathbf{Role}]_X\ \mathsf{SymDec}(X, E_{sym}[k^{type}_{X,Y}](M_1), k^{type}_{X,Y}) \tag{1}$$

$$\mathsf{Hon}(\hat{X}, \hat{Y}), \Gamma_0 \quad [\mathbf{Role}]_X\ \exists \eta.\ \mathsf{SymEnc}((\hat{X}, \eta), M_1, k^{type}_{X,Y})$$

$$\mathbf{ENC4}, (-1) \qquad \vee\ \exists \eta.\ \mathsf{SymEnc}((\hat{Y}, \eta), M_1, k^{type}_{X,Y}) \tag{2}$$

$$\text{Case 1} : \hat{X} = \hat{Y} \tag{3}$$

$$(-2, -1) \quad [\mathbf{Role}]_X\ \exists \eta.\ \mathsf{SymEnc}((\hat{Y}, \eta), M_1, k^{type}_{X,Y}) \tag{4}$$

$$\text{Case 2} : \hat{X} \neq \hat{Y} \tag{5}$$

$$\mathbf{HON} \quad \mathsf{Honest}(\hat{X_0}) \wedge \hat{X_0} \neq \hat{Y_0} \supset \forall M.\ \neg \mathsf{SymEnc}(X_0, M, k^{type}_{X_0,Y_0}) \tag{6}$$

$$\mathsf{Hon}(\hat{X}), (-1) \quad [\mathbf{Role}]_X\ \neg \exists \eta.\ \mathsf{SymEnc}((\hat{X}, \eta), M_1, k^{type}_{X,Y}) \tag{7}$$

$$(-6, -1) \quad [\mathbf{Role}]_X\ \exists \eta.\ \mathsf{SymEnc}((\hat{Y}, \eta), M_1, k^{type}_{X,Y}) \tag{8}$$

Instantiating for $AUTH_{kas}^{client}$:

$$[\mathbf{Client}]_C \; \exists \eta.\, \mathsf{SymEnc}((\hat{C}, \eta), AKey.n_1.\hat{T}, k_{C,K}^{c \to k}) \tag{9}$$

$$\begin{aligned} \mathbf{HON} \quad & \mathsf{Honest}(\hat{X}) \wedge \mathsf{SymEnc}(X, Key.n.\hat{T_0}, k_{C_0,X}^{c \to k}) \\ & \supset \mathsf{Send}(X, \hat{C_0}.E_{sym}[k_{T_0,X}^{t \to k}](Key.\hat{C_0}).E_{sym}[k_{C_0,X}^{c \to k}](Key.n.\hat{T_0})) \end{aligned} \tag{10}$$

$$\begin{aligned} \mathsf{Hon}(\hat{K}), \quad & [\mathbf{Client}]_C \; \exists \eta.\, \mathsf{Send}((\hat{K}, \eta), \hat{C}.E_{sym}[k_{T,K}^{t \to k}](AKey.\hat{C}). \\ (-2,-1) \quad & E_{sym}[k_{C,K}^{c \to k}](AKey.n_1.\hat{T})) \end{aligned} \tag{11}$$

$$(-1) \quad AUTH_{kas}^{client} \tag{12}$$

Instantiating for $AUTH_{kas}^{tgs}$:

$$[\mathbf{TGS}]_T \; \exists \eta.\, \mathsf{SymEnc}((\hat{K}, \eta), AKey.\hat{C}, k_{T,K}^{t \to k}) \tag{13}$$

$$\begin{aligned} \mathbf{HON} \quad & \mathsf{Honest}(\hat{X}) \wedge \mathsf{SymEnc}(X, Key.\hat{C_0}, k_{Y,X}^{t \to k}) \\ & \supset \exists n.\, \mathsf{Send}(X, \hat{C_0}.E_{sym}[k_{Y,X}^{t \to k}](Key.\hat{C_0}).E_{sym}[k_{C_0,X}^{c \to k}](Key.n.\hat{Y})) \end{aligned} \tag{14}$$

$$\begin{aligned} \mathsf{Hon}(\hat{K}), \quad & [\mathbf{TGS}]_T \; \exists \eta, n.\, \mathsf{Send}((\hat{K}, \eta), \hat{C}.E_{sym}[k_{T,K}^{t \to k}](AKey.\hat{C}). \\ (-2,-1) \quad & E_{sym}[k_{C,K}^{c \to k}](AKey.n_1.\hat{T})) \end{aligned} \tag{15}$$

$$(-1) \quad AUTH_{kas}^{tgs} \tag{16}$$

Instantiating for $AUTH_{tgs}^{server}$:

$$[\mathbf{Server}]_S \; \exists \eta.\, \mathsf{SymEnc}((\hat{T}, \eta), E_{sym}[k_{S,T}^{s \to t}](SKey.\hat{C})) \tag{17}$$

$$\begin{aligned} \mathbf{HON} \quad & \mathsf{Honest}(\hat{X}) \wedge \mathsf{SymEnc}(X, Key.\hat{C_0}, k_{Y,X}^{s \to t}) \\ & \supset \exists n, Key'.\, \mathsf{Send}(X, \hat{C_0}.E_{sym}[k_{Y,X}^{s \to t}](Key.\hat{C_0}).E_{sym}[Key'](Key.n.\hat{Y})) \end{aligned} \tag{18}$$

$$\begin{aligned} \mathsf{Hon}(\hat{T}), \quad & [\mathbf{Server}]_S \; \exists \eta, n, Key'.\, \mathsf{Send}((\hat{T}, \eta), \hat{C}.E_{sym}[k_{S,T}^{s \to t}](SKey.\hat{C}). \\ (-2,-1) \quad & E_{sym}[Key'](SKey.n.\hat{S})) \end{aligned} \tag{19}$$

$$(-1) \quad AUTH_{tgs}^{server} \tag{20}$$

D.3. Proof of $SEC^{client}_{akey}, SEC^{kas}_{akey}, SEC^{tgs}_{akey}$

In this section we formally prove the secrecy of the session key $AKey$ with respect to the key-set $\mathcal{K} = \{k^{c\to k}_{C,K}, k^{t\to k}_{T,K}\}$.

The assumed condition Φ is the conjunction of the following formulas where the predicate $\mathsf{ContainsOpen}(m, a)$ asserts that a can be obtained from m by a series of unpairings only - no decryption required.

$$\Phi_1 : \forall X, M.\ \mathsf{New}(X, AKey) \supset \neg(\mathsf{Send}(X, M) \wedge \mathsf{ContainsOpen}(M, AKey))$$

$$\Phi_2 : \forall X, \hat{C_0}, \hat{K_0}, \hat{T_0}, n.\ \mathsf{New}(X, AKey) \wedge \mathsf{SymEnc}(X, AKey.n.\hat{T_0}, k^{c\to k}_{\hat{C_0},K_0})$$
$$\supset \hat{X} = \hat{K} \wedge \hat{C_0} = \hat{C} \wedge \hat{T_0} = \hat{T}$$

$$\Phi_3 : \forall X, \hat{S_0}, \hat{C_0}.\ \mathsf{New}(X, AKey) \supset \neg\mathsf{SymEnc}(X, AKey.\hat{C_0}, k^{s\to t}_{S_0,X})$$

Observe that Φ is prefix closed. The only principals having access to a key in $\mathcal{K}$ are $\hat{C}, \hat{K}$ and $\hat{T}$. In addition, Φ_2 assumes that some thread of $\mathcal{K}$ generated $AKey$. Therefore, we have $\mathsf{KOHonest}(AKey, \mathcal{K}) \equiv \mathsf{Hon}(\hat{C}, \hat{K}, \hat{T})$. As the form of the secrecy induction suggests, we do an induction over all the basic sequences of *KERBEROS*.

Let, $[\mathbf{Client}_1]_{C'} : [\mathtt{new}\ n'_1; \mathtt{send}\ \hat{C'}.\hat{T'}.n'_1;]_{C'}$

$$[\mathbf{Client}_1]_{C'}\ \mathsf{New}(C', n'_1) \wedge \mathsf{Send}(C', \hat{C'}.\hat{T'}.n'_1) \quad (1)$$

$\Phi_1, (1)$ $\quad [\mathbf{Client}_1]_{C'}\ n'_1 \neq AKey \quad (2)$

(2) $\quad [\mathbf{Client}_1]_{C'}\ \mathsf{SafeMsg}(\hat{C'}.\hat{T'}.n'_1, AKey, \mathcal{K}) \quad (3)$

$\mathbf{NET2}, (3)$ $\quad \mathsf{SafeNet}(AKey, \mathcal{K})\ [\mathbf{Client}_1]_{C'}\ \mathsf{SendsSafeMsg}(C', AKey, \mathcal{K}) \quad (4)$

Let, $[\mathbf{Client}_2]_{C'} : [\mathtt{receive}\ \hat{C'}.tgt'.enc'_{kc};$
$text'_{kc} := \mathtt{symdec}\ enc'_{kc}, k^{c\to k}_{C',K'};$
$\mathtt{match}\ text'_{kc}\ \mathtt{as}\ AKey'.n'_1.\hat{T'};]_{C'}$

$$\mathbf{NET}* \quad \mathsf{SafeNet}(AKey, \mathcal{K})\ [\mathbf{Client}_2]_{C'}\ \mathsf{SendsSafeMsg}(C', AKey, \mathcal{K}) \quad (5)$$

Precondition θ_3 : $\mathsf{Receive}(C', \hat{C}'.tgt'.E_{sym}[k^{c\rightarrow k}_{C',K'}](AKey'.n'_1.\hat{T}'))$

Let, $[\mathbf{Client}_3]_{C'}$: $[\mathtt{new}\ \ n'_2;$

$enc'_{ct} := \mathtt{symenc}\ \ \hat{C}', AKey';$

$\mathtt{send}\ \ tgt'.enc'_{ct}.\hat{C}'.\hat{S}', n'_2;]_{C'}$

SREC $\mathsf{SafeNet}(AKey, \mathcal{K}) \wedge \theta_3 \supset$

$\mathsf{SafeMsg}(\hat{C}'.tgt'.E_{sym}[k^{c\rightarrow k}_{C',K'}](AKey'.n'_1.\hat{T}'), AKey, \mathcal{K})$ (6)

SAF1 $\mathsf{SafeMsg}(\hat{C}'.tgt'.E_{sym}[k^{c\rightarrow k}_{C',K'}](AKey'.n'_1.\hat{T}'), AKey, \mathcal{K}) \supset$

$\mathsf{SafeMsg}(tgt', AKey, \mathcal{K})$ (7)

(7) $\theta_3 \wedge \mathsf{SafeNet}(AKey, \mathcal{K})\ [\mathbf{Client}_3]_{C'}\ \mathsf{SafeMsg}(tgt', AKey, \mathcal{K})$ (8)

$[\mathbf{Client}_3]_{C'}\ \mathsf{New}(C', n'_2) \wedge \mathsf{Send}(C', tgt'.E_{sym}[AKey'](\hat{C}').\hat{C}'.\hat{S}'.n'_2)$ (9)

Φ_1, (9) $[\mathbf{Client}_3]_{C'}\ n'_2 \neq AKey$ (10)

(8), (10) $\theta_3 \wedge \mathsf{SafeNet}(AKey, \mathcal{K})\ [\mathbf{Client}_3]_{C'}$

$\mathsf{SafeMsg}(tgt', AKey, \mathcal{K}) \wedge \mathsf{SafeMsg}(n'_2, AKey, \mathcal{K})$ (11)

(11) $\theta_3 \wedge \mathsf{SafeNet}(AKey, \mathcal{K})\ [\mathbf{Client}_3]_{C'}$

$\mathsf{SafeMsg}(tgt'.E_{sym}[AKey'](\hat{C}').\hat{C}'.\hat{S}'.n'_2, AKey, \mathcal{K})$ (12)

NET$*$, (12) $\theta_3 \wedge \mathsf{SafeNet}(AKey, \mathcal{K})\ [\mathbf{Client}_3]_{C'}\ \mathsf{SendsSafeMsg}(C', AKey, \mathcal{K})$ (13)

$\cdots$ proof for following BS similar to (5) $\cdots$

$\mathsf{SafeNet}(AKey, \mathcal{K})\ [\mathtt{receive}\ \ \hat{C}'.st'.enc'_{tc};$

$text'_{tc} := \mathtt{symdec}\ \ enc'_{tc}, AKey';$

$\mathtt{match}\ \ text'_{tc}\ \mathtt{as}\ \ SKey'.n'_2.\hat{S}';]_{C'}$

$\mathsf{SendsSafeMsg}(C', AKey, \mathcal{K})$ (14)

Precondition θ_5 : $\mathsf{Receive}(C', \hat{C}'.st'.E_{sym}[AKey'](SKey'.n'_2.\hat{S}'))$

$\cdots$ proof for following BS similar to (13) $\cdots$

$\theta_5 \wedge \mathsf{SafeNet}(AKey, \mathcal{K})\ [enc'_{cs} := \mathtt{symenc}\ \ \hat{C}'.t', SKey';$

$\mathtt{send}\ \ st'.enc'_{cs};]_{C'}$

$\mathsf{SendsSafeMsg}(C', AKey, \mathcal{K})$ (15)

$\cdots$ proof for following BS similar to (5) $\cdots$

$$\mathsf{SafeNet}(AKey, \mathcal{K})\ [\texttt{receive}\ enc'_{sc};$$
$$text'_{sc} := \texttt{symdec}\ enc'_{sc}, SKey';$$
$$\texttt{match}\ text'_{sc}\ \texttt{as}\ t';]_{C'}$$
$$\mathsf{SendsSafeMsg}(C', AKey, \mathcal{K}) \tag{16}$$

Let, $[\mathbf{KAS}]_{K'}$: $[\texttt{receive}\ \hat{C}'.\hat{T}'.n'_1;$
$$\texttt{new}\ AKey';$$
$$tgt' := \texttt{symenc}\ AKey'.\hat{C}', k^{t\rightarrow k}_{T',K'};$$
$$enc'_{kc} := \texttt{symenc}\ AKey'.n'_1.\hat{T}', k^{c\rightarrow k}_{C',K'};$$
$$\texttt{send}\ \hat{C}'.tgt'.enc'_{kc};]_{K'}$$

Case 1 : $AKey' = AKey$

$$[\mathbf{KAS}]_{K'}\mathsf{New}(K', AKey) \wedge \mathsf{SymEnc}(K', AKey.n'_1.\hat{T}', k^{c\rightarrow k}_{C',K'}) \tag{17}$$

$\Phi_2, (17)$ $$[\mathbf{KAS}]_{K'}\hat{C}' = \hat{C} \wedge \hat{K}' = \hat{K} \wedge \hat{T}' = \hat{T} \tag{18}$$

(18) $$[\mathbf{KAS}]_{K'}k^{c\rightarrow k}_{C',K'} \in \mathcal{K} \wedge k^{t\rightarrow k}_{T',K'} \in \mathcal{K} \tag{19}$$

$\mathbf{SAF}*, (19)$ $\mathsf{SafeNet}(AKey, \mathcal{K})\ [\mathbf{KAS}]_{K'}\ \mathsf{SafeMsg}($
$$\hat{C}'.E_{sym}[k^{t\rightarrow k}_{T',K'}](AKey'.\hat{C}').E_{sym}[k^{c\rightarrow k}_{C',K'}](AKey'.n'_1.\hat{T}'),$$
$$AKey, \mathcal{K}) \tag{20}$$

Case 2 : $AKey' \neq AKey$

$\mathbf{NET1}$ $\mathsf{SafeNet}(AKey, \mathcal{K})\ [\texttt{receive}\ \hat{C}'.\hat{T}'.n'_1;]_{K'}\ \mathsf{SafeMsg}(\hat{C}'.\hat{T}'.n'_1, AKey, \mathcal{K})$ (21)

(21) $\mathsf{SafeNet}(AKey, \mathcal{K})\ [\texttt{receive}\ \hat{C}'.\hat{T}'.n'_1;]_{K'}\ \mathsf{SafeMsg}(n'_1, AKey, \mathcal{K})$ (22)

$\mathbf{SAF}*, (22)$ $\mathsf{SafeNet}(AKey, \mathcal{K})\ [\mathbf{KAS}]_{K'}\ \mathsf{SafeMsg}($
$$\hat{C}'.E_{sym}[k^{t\rightarrow k}_{T',K'}](AKey'.\hat{C}').E_{sym}[k^{c\rightarrow k}_{C',K'}](AKey'.n'_1.\hat{T}')),$$
$$AKey, \mathcal{K}) \tag{23}$$

$(20), (23), \mathbf{NET}*$ $\mathsf{SafeNet}(AKey, \mathcal{K})\ [\mathbf{KAS}]_{K'}\ \mathsf{SendsSafeMsg}(K', AKey, \mathcal{K})$ (24)

Let, $[\mathbf{TGS}]_{T'}$: $[\texttt{receive}\ \ tgt'.enc'_{ct}.\hat{C}'.\hat{S}'.n'_2;$

$text'_{tgt} := \texttt{symdec}\ \ tgt', k^{t\rightarrow k}_{T,K};$

$\texttt{match}\ \ text'_{tgt}\ \texttt{as}\ \ AKey'.\hat{C}';$

$text'_{ct} := \texttt{symdec}\ \ enc'_{ct}, AKey';$

$\texttt{match}\ \ text'_{ct}\ \texttt{as}\ \ \hat{C}';$

$\texttt{new}\ \ SKey';$

$st' := \texttt{symenc}\ \ SKey'.\hat{C}', k^{s\rightarrow t}_{S,T};$

$enc'_{tc} := \texttt{symenc}\ \ SKey'.n'_2.\hat{S}', AKey';$

$\texttt{send}\ \ \hat{C}'.st'.enc'_{tc};]_{T'}$

NET1, SAF1 $\mathsf{SafeNet}(AKey, \mathcal{K})\ [\texttt{receive}\ \ enc'_{ct1}.enc'_{ct2}.\hat{C}'.\hat{S}'.n'_2;]_{K'}$

$\mathsf{SafeMsg}(n'_2, AKey, \mathcal{K})$ (25)

$[\mathbf{TGS}]_{T'}\ \mathsf{New}(T', SKey') \wedge \mathsf{SymEnc}(T', SKey'.\hat{C}', k^{s\rightarrow t}_{S',T'})$ (26)

$\Phi_3, (26)$ $[\mathbf{TGS}]_{T'}\ SKey' \neq AKey$ (27)

$(25), (27),$ $\mathsf{SafeNet}(AKey, \mathcal{K})\ [\mathbf{TGS}]_{T'}\ \mathsf{SafeMsg}($

SAF$*$ $\hat{C}'.E_{sym}[k^{s\rightarrow t}_{S',T'}](SKey'.\hat{C}').E_{sym}[AKey'](SKey'.n'_2.\hat{S}'), AKey, \mathcal{K})$ (28)

NET$*, (28)$ $\mathsf{SafeNet}(AKey, \mathcal{K})\ [\mathbf{TGS}]_{T'}\ \mathsf{SendsSafeMsg}(T', AKey, \mathcal{K})$ (29)

Let, $[\mathbf{Server}]_{S'}$: $[\texttt{receive}\ \ st'.enc'_{cs};$

$text'_{st} := \texttt{symdec}\ \ st', k^{s\rightarrow t}_{S,T};$

$\texttt{match}\ \ text'_{st}\ \texttt{as}\ \ SKey'.\hat{C}';$

$text'_{cs} := \texttt{symdec}\ \ enc'_{cs}, SKey';$

$\texttt{match}\ \ text'_{cs}\ \texttt{as}\ \ \hat{C}'.t';$

$enc'_{sc} := \texttt{symenc}\ \ t', SKey';$

$\texttt{send}\ \ enc'_{sc};]_{S'}$

NET1, SAF0 $\mathsf{SafeNet}(AKey, \mathcal{K})\ [\mathbf{Server}]_{S'}\ \mathsf{SafeMsg}(E_{sym}[SKey'](\hat{C}'.t'), AKey, \mathcal{K})$ (30)

SAF$*, (30)$ $\mathsf{SafeNet}(AKey, \mathcal{K})\ [\mathbf{Server}]_{S'}\ \mathsf{SafeMsg}(t', AKey, \mathcal{K}) \vee SKey' \in \mathcal{K}$ (31)

SAF1, (31) $\mathsf{SafeNet}(AKey, \mathcal{K})\ [\mathbf{Server}]_{S'}\ \mathsf{SafeMsg}(E_{sym}[SKey'](t'), AKey, \mathcal{K})$ (32)

NET2 $\mathsf{SafeNet}(AKey, \mathcal{K})\ [\mathbf{Server}]_{S'}\ \mathsf{SendsSafeMsg}(S', AKey, \mathcal{K})$ (33)

$$\text{Theorem 6} \quad \Phi \wedge \mathsf{Hon}(\hat{C}, \hat{K}, \hat{T}) \supset \mathsf{SafeNet}(AKey, \mathcal{K}) \tag{34}$$

$$\mathbf{POS}, (34) \quad \Phi \wedge \mathsf{Hon}(\hat{C}, \hat{K}, \hat{T}) \supset (\mathsf{Has}(X, AKey) \supset (\hat{X} = \hat{C} \vee \hat{X} = \hat{K} \vee \hat{X} = \hat{T})) \tag{35}$$

Based on $AUTH_{kas}^{client}$, the actions in $[\mathbf{KAS}]_K$, $AUTH_{tgs}^{client}$ and a few additional steps, we can infer that:

$$\textit{KERBEROS} \vdash [\mathbf{Client}]_C \, \mathsf{Hon}(\hat{C}, \hat{K}, \hat{T}) \supset \Phi$$

$$\textit{KERBEROS} \vdash [\mathbf{KAS}]_K \, \mathsf{Hon}(\hat{C}, \hat{K}, \hat{T}) \supset \Phi$$

$$\textit{KERBEROS} \vdash [\mathbf{TGS}]_T \, \mathsf{Hon}(\hat{C}, \hat{K}, \hat{T}) \supset \Phi$$

Combining these with the secrecy derivation (35) we have:

$$\textit{KERBEROS} \vdash SEC_{akey}^{client}, SEC_{akey}^{kas}, SEC_{akey}^{tgs}$$

D.4. Proof of $AUTH_{tgs}^{client}$

This proof uses the secrecy property SEC_{akey}^{client} which established the secrecy of $AKey$ among $\hat{C}, \hat{K}$ and $\hat{T}$ assuming their honesty. Again, reference to equations by negative numbers is relative to the current equation - *e.g.*, (-1) refers to the last equation. Reference by positive number indicates the actual number of the equation.

$$[\mathbf{Client}]_C\ \mathsf{SymDec}(C, E_{sym}[AKey](SKey.n_2.\hat{S}), AKey) \tag{1}$$

(-1) $$[\mathbf{Client}]_C\ \exists X.\ \mathsf{SymEnc}(X, SKey.n_2.\hat{S}, AKey) \tag{2}$$

Inst $X \mapsto X_0, (-1)$ $$[\mathbf{Client}]_C\ \mathsf{SymEnc}(X_0, SKey.n_2.\hat{S}, AKey) \tag{3}$$

$\mathbf{ENC3}, (-1)$ $$[\mathbf{Client}]_C\ \mathsf{Has}(X_0, AKey) \tag{4}$$

$SEC_{AKey}^{client}, (-1)$ $$\hat{X_0} = \hat{C} \wedge \hat{X_0} = \hat{K} \wedge \hat{X_0} = \hat{T} \tag{5}$$

$\mathbf{HON}$ $$\mathsf{Honest}(\hat{X}) \wedge \mathsf{SymEnc}(X, Key'.n.\hat{S_0}, Key) \wedge Key \neq k_{Z,X}^{c \rightarrow k} \supset \exists \hat{K_0}, \hat{C_0}.\ \mathsf{SymDec}(X, E_{sym}[k_{X,K_0}^{t \rightarrow k}](Key.\hat{C_0})) \wedge \mathsf{Send}(X, \hat{C_0}.E_{sym}[k_{S_0,X}^{s \rightarrow t}](Key'.\hat{C_0}).E_{sym}[Key](Key'.n.\hat{S_0})) \tag{6}$$

Inst, $(-4, -1)$ $$[\mathbf{Client}]_C\ \mathsf{SymDec}(X_0, E_{sym}[k_{X_0,K_0}^{t \rightarrow k}](AKey.\hat{C_0})) \wedge \mathsf{Send}(X_0, \hat{C_0}.E_{sym}[k_{S,X_0}^{s \rightarrow t}](SKey.\hat{C_0}).E_{sym}[AKey](SKey.n_2.\hat{S})) \tag{7}$$

(-1) $$[\mathbf{Client}]_C\ \exists Y.\ \mathsf{SymEnc}(Y, AKey.\hat{C_0}, k_{X_0,K_0}^{t \rightarrow k}) \tag{8}$$

Inst $Y \mapsto Y_0, (-1)$ $$[\mathbf{Client}]_C\ \mathsf{SymEnc}(Y_0, AKey.\hat{C_0}, k_{X_0,K_0}^{t \rightarrow k}) \tag{9}$$

$\mathbf{ENC3}, (-1)$ $$[\mathbf{Client}]_C\ \mathsf{Has}(Y_0, AKey) \tag{10}$$

$SEC_{AKey}^{client}, (-1)$ $$\mathsf{Honest}(\hat{Y_0}) \tag{11}$$

$\mathbf{HON}$ $$\mathsf{Honest}(\hat{X}) \wedge \mathsf{SymEnc}(Y, Key.\hat{W}, k_{X,Z}^{t \rightarrow k}) \supset \mathsf{New}(X, Key) \tag{12}$$

$(-4, -1)$ $$[\mathbf{Client}]_C\ \mathsf{New}(Y_0, AKey) \tag{13}$$

$AUTH_{kas}^{client}$ $$\mathsf{New}(X, AKey) \wedge \mathsf{SymEnc}(X, AKey.\hat{W}, k_{Y,Z}^{t \rightarrow k}) \supset \hat{Y} = \hat{T} \wedge \hat{Z} = \hat{K} \wedge \hat{W} = \hat{C} \tag{14}$$

$(9, -2, -1)$ $$\hat{X_0} = \hat{T} \wedge \hat{K_0} = \hat{K} \wedge \hat{C_0} = \hat{C} \tag{15}$$

$(7, -1)$ $$[\mathbf{Client}]_C\ \exists \eta.\ \mathsf{Send}((\hat{T}, \eta), \hat{C}.E_{sym}[k_{S,T}^{s \rightarrow t}](SKey.\hat{C}).\ E_{sym}[AKey](SKey.n_2.\hat{S})) \tag{16}$$

(-1) $$AUTH_{tgs}^{client} \tag{17}$$