Inductive Proofs of Computational Secrecy *

Arnab Roy[1], Anupam Datta[2], Ante Derek[1], John C. Mitchell[1]

[1] Stanford University, Stanford, CA
{arnab, aderek, mitchell}@cs.stanford.edu
[2] Carnegie Mellon University, Pittsburgh, PA
danupam@cmu.edu

Abstract. Secrecy properties of network protocols assert that no probabilistic polynomial-time distinguisher can win a suitable game presented by a challenger. Because such properties are not determined by trace-by-trace behavior of the protocol, we establish a trace-based protocol condition, suitable for inductive proofs, that guarantees a generic reduction from protocol attacks to attacks on underlying primitives. We use this condition to present a compositional inductive proof system for secrecy, and illustrate the system by giving a modular, formal proof of computational authentication and secrecy properties of Kerberos V5.

1 Introduction

Present-day Internet users and networked enterprises rely on key management and related protocols that use cryptographic primitives. In spite of the staggering financial value of, say, the total number of credit card numbers transmitted by SSL/TLS in a day, we do not have correctness proofs that respect cryptographic notions of security for many of these relatively simple distributed programs. In light of this challenge, there have been many efforts to develop and use methods for proving security properties of network protocols. Historically, most efforts used an abstract *symbolic model,* also referred to as the *Dolev-Yao* model [26, 20]. More recently, in part to draw stronger conclusions from existing methods and proofs, several groups of researchers have taken steps to connect the symbolic model to probabilistic polynomial-time *computational models* accepted in cryptographic studies, *e.g.* [2, 6, 8, 13, 27, 17, 18, 32].

A fundamental problem in reasoning about secrecy, such as computational indistinguishability of a key from a randomly chosen value, is that such secrecy properties are not trace properties – indistinguishability over a set of possible runs is *not* defined by summing the probability of indistinguishability on each run. As a result, it does not appear feasible to prove computational secrecy properties by induction on the steps of a protocol. A central contribution of this paper is a form of trace-based property, called *secretive,* suitable for inductive

* This work was partially supported by the NSF Science and Technology Center TRUST and U.S. Army Research Office contract on Perpetually Available and Secure Information Systems (DAAD19-02-1-0389) to CMU's CyLab. The first author was additionally supported by a Siebel Fellowship.

and compositional proofs, together with a form of standard cryptographic reduction argument which shows that any attack on a secretive protocol yields an attack on cryptographic primitives used in the protocol. We give the cryptographic reduction in a precise form, by inductively defining the operational behavior of a simulator that simulates the protocol to the protocol adversary. An essential problem in defining the simulator, which interacts with a game providing access to the cryptographic primitives, is that the simulator has two candidate secrets, but must present a view of the protocol that is consistent with either candidate being the actual protocol secret.

After proving that secretive protocols yield black-box reductions, we present one inductive method for showing that a protocol is secretive, based on Computational Protocol Composition Logic (CPCL) [17, 18]. In the process, we generalize a previous induction rule, so that only one core induction principle is needed in the logic. In contrast to proof systems for symbolic secrecy [28, 31], the induction is over actions of honest parties and not the structure of terms. We also extend previous composition theorems [16, 21] to the present setting, and illustrate the power of the resulting system by giving modular formal proofs of authentication and secrecy properties of Kerberos V5 and Kerberos V5 with PKINIT. We are also able to prove properties of a variant of the Needham-Schroeder-Lowe protocol that are beyond the standard rank function method [22, 19].

Our approach may be compared with equivalence-based methods [6, 27, 14, 5], used in [3] to derive some computational properties of Kerberos V5 from a symbolic proof. In equivalence-based methods, the behavior of a symbolic abstraction under symbolic attack must have the same observable behavior as a computational execution under computational (probabilistic polynomial-time) attack. In contrast, our approach only requires an implication between symbolic reasoning and computational execution. While we believe that both approaches have merit, the two are distinguished by (i) the need to additionally prove the absence of a "commitment problem" in [3], which appears to be a fundamental issue in equivalence-based security [15], and (ii) the open problem expressed in [3] of developing compositional methods in that framework. Symbolic abstractions for primitives like Diffie-Hellman key exchange are also problematic for equivalence-based approaches [4, 7], but amenable to treatment in PCL. In contrast to other symbolic or computationally sound methods, PCL reasoning proceeds only over action sequences of the protocol program, yet the conclusions are sound for protocol execution in the presence of attack. This formalizes and justifies a direct reasoning method that is commonly used informally among researchers, yet is otherwise not rigorously connected to reduction arguments.

Section 2 describes the protocol process calculus and computational execution model. A trace-based definition of "secretive protocols" and relevant computational notions are explained in section 3. The proof system, and soundness and composition theorems are presented in section 4, and applied in the proofs for Kerberos in section 5. Conclusions appear in section 6. Many of the proofs and technical details in this paper are omitted due to space constraints—interested readers can find them in the full version of this paper [29].

2 Syntax and Semantics

We begin by reviewing a protocol notation, protocol logic, and a security model for key exchange developed in earlier work [16–18].

Modeling Protocols. Protocols are expressed in a process calculus by defining a set of *roles*, such as "Client", or "Server", each given by a sequence of actions such as sending or receiving a message, generating a new nonce, or decrypting or encrypting a message (see [16]). In a run of a protocol, a principal may execute one or more instances of each role, each execution constituting a *thread* identified by a pair $(\hat{X}, \eta)$, where $\hat{X}$ is a principal and η is a unique session identifier.

We illustrate the protocol process calculus using Kerberos V5 [25], which will be the running example in this paper. Our formulation is based on the A level formalization of Kerberos V5 in [12]. Kerberos provides mutual authentication and establishes keys between clients and application servers, using a sequence of two-message interactions with trusted parties called the Kerberos Authentication Server (KAS) and the Ticket Granting Server (TGS).

Kerberos has four roles, **Client**, **KAS**, **TGS** and **Server**. The pre-shared long-term keys between the client and KAS, the KAS and TGS, and the TGS and application server, will be written as $k^{type}_{X,Y}$ where X and Y are the principals sharing the key. The *type* appearing in the superscript indicates the relationship between X and Y: $c \rightarrow k$ indicates that X is acting as a client and Y is acting as a KAS, $t \rightarrow k$ for TGS and KAS and $s \rightarrow t$ for application server and TGS.

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

In the first stage, the client (C) generates a nonce (represented by `new` n_1) and sends it to the KAS (K) along with the identities of the TGS (T) and itself. The KAS generates a new nonce ($AKey$ - Authentication Key) to be used as a session key between the client and the TGS. It then sends this key along with some other fields to the client encrypted (represented by the `symenc` actions) under two different keys - one it shares with the client ($k^{c\rightarrow k}_{C,K}$) and one it shares with the TGS ($k^{t\rightarrow k}_{T,K}$). The encryption with $k^{t\rightarrow k}_{T,K}$ is called the ticket granting ticket (tgt). The client extracts $AKey$ by decrypting the component encrypted with $k^{c\rightarrow k}_{C,K}$ and recovering its parts using the `match` action which deconstructs $text_{kc}$ and associates the parts of this plaintext with $AKey$, n_1, and $\hat{T}$. The ellipses (...) indicates further client steps for interacting with KAS, TGS, and the application server that are omitted due to space constraints (see [31] for a full description).

Action Predicates:
$\mathsf{a} ::= \mathsf{Send}(X,t) \mid \mathsf{Receive}(X,t) \mid \mathsf{SymEnc}(X,t,k) \mid \mathsf{SymDec}(X,t,k) \mid \mathsf{New}(X,n)$
Formulas:
$\varphi ::= \mathsf{a} \mid t = t \mid \mathsf{Start}(X) \mid \mathsf{Honest}(\hat{X}) \mid \mathsf{Possess}(X,\ t) \mid \mathsf{Indist}(X,\ t) \mid$
$\mathsf{GoodKeyAgainst}(X,\ t) \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \exists V.\, \varphi \mid \forall V.\, \varphi \mid \neg\varphi \mid \varphi \supset \varphi \mid \varphi \Rightarrow \varphi$
Modal formulas:
$\Psi ::= \varphi\ [Actions]_X\ \varphi$

Table 1. Syntax of the logic

In the second stage, the client gets a new session key ($SKey$ - Service Key) and a service ticket (st) to converse with the application server S which takes place in the third stage. The control flow of Kerberos exhibits a staged architecture where once one stage has been completed successfully, the subsequent stages can be performed multiple times or aborted and started over for handling errors.

Execution Model. We use a standard two-phase protocol execution model as in [11]. In the initialization phase, we assign roles to each principal, identify a subset that is honest, and provide encryption keys and random coins as needed. In the execution phase, the adversary executes the protocol by interacting with honest principals. We assume the adversary has complete control over the network, as in [11]. The length of keys and the running time of the protocol are polynomially bounded in the security parameter.

A *trace* is a record of all actions executed by honest principals and the attacker during protocol execution. Since honest principals execute roles defined by a symbolic process calculus, our traces contain symbolic descriptions of the actions of honest parties and a mapping of symbolic variables to bitstrings values. The attacker may produce and send arbitrary bitstrings, but the trace only records the send-receive actions of the attacker, and not its internal actions. Our traces also include the random bits (used by the honest parties, the adversary and the distinguisher), as well as a few other elements used in defining semantics of formulas over traces [17]. In section 3, which presents semantic arguments independent of the protocol logic, we omit these additional fields and refer to a trace as $\langle e, \lambda \rangle$, where e is a symbolic description of the trace and λ maps terms in e to bitstrings.

For technical reasons, we assume that honest parties conform to certain type conventions. These restrictions may be imposed by prefixing the values of each type (nonces, ids, constant strings, pairs, encryptions with key k, etc.) with a tag such as 'constant' or 'encrypted with key k' that are respected by honest parties executing protocol roles. The adversary may freely modify or spoof tags or produce arbitrary untagged bitrings.

Syntax of Computational PCL. The formulas of the logic are given in Table 1. 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. Most formulas have the same intuitive meaning as in the symbolic model [16].

For every protocol action, there is a corresponding action predicate which asserts that the action has occurred in the run. For example, $\mathsf{Send}(X, t)$ holds in a run where the thread X has sent the term t. $\mathsf{Honest}(\hat{X})$ means that the principal $\hat{X}$ is acting honestly, *i.e.,* the actions of every thread of $\hat{X}$ precisely follows some role of the protocol. $\mathsf{Start}(X)$ means that the thread X did not execute any actions in the past. $\mathsf{Indist}(X, t)$ means that agent X cannot tell the bitstring representation of the term t from another bitstring chosen at random from the same distribution. The logical connectives have standard interpretations, except that conditional implication ($\Rightarrow$), related to a form of conditional probability, appears essential for reasoning about cryptographic reductions (see [17] for further discussion).

Semantics of Computational PCL. Intuitively, a protocol Q satisfies a formula φ, written $Q \models \varphi$ if for all adversaries and sufficiently large security parameters, the *probability* that φ "holds" is asymptotically close to 1 (in the security parameter). Intuitively, the meaning of a formula φ on a set T of computational traces is usually a subset $T' \subseteq T$ that respects φ in some specific way. For example, an action predicate such as Send selects a set of traces in which a send occurs. More precisely, the semantics $[\![\varphi]\!]\,(T, D, \epsilon)$ of a formula φ is inductively defined on the set T of traces, with distinguisher D and tolerance ϵ. The distinguisher and tolerance are only used in the semantics of Indist and $\mathsf{GoodKeyAgainst}$, where they determine whether the distinguisher has more than a negligible chance of distinguishing the given value from random or winning an IND-CCA game, respectively. The precise inductive semantics for formulas is given in [17].

The semantics of the predicate $\mathsf{GoodKeyAgainst}(X, k)$ is defined using a standard cryptographic-style game condition. It captures the intuition that a key output by a secure key exchange protocol should be suitable for use in some application protocol of interest (e.g. as a key for an IND-CCA secure encryption scheme) [18]. Formally, $[\![\mathsf{GoodKeyAgainst}(X, k)]\!](T, D, \epsilon)$ is the complete set of traces T if the distinguisher D, who is given X's view of the run has an advantage less than ϵ in winning the IND-CCA game [9] against a challenger using the bitstring corresponding to term k as the key, and $\emptyset$ otherwise. Here the probability is taken by choosing a uniformly random trace $t \in T$ (which includes the randomness of all parties, the attacker and the distinguisher). The same approach can be used to define other game conditions.

A *trace property* is a formula φ such that for any set of protocol traces T, $[\![\varphi]\!]\,(T) = \bigcup_{t \in T} [\![\varphi]\!]\,(\{t\})$. The distinguisher and tolerance are omitted since they are not used in defining semantics for such predicates. Thus all action formulas, such as $\mathsf{Send}(X, m)$, are trace properties whereas aggregrate properties such as $\mathsf{Indist}(X, k)$ and $\mathsf{GoodKeyAgainst}\ (X, k)$ are not.

3 Secretive Protocols

In this section, we define a trace property of protocols and show that this property implies computational secrecy and integrity. The computational secrecy properties include key indistinguishability and key usability for IND-CCA secure encryption. These results are established first for the simple case when secrets are protected by pre-shared "level-0" keys (Theorem 1), then generalized (Theorems 2-3) under the condition that each key is protected by predecessor keys in an acyclic graph[3]. The proofs use standard cryptographic reductions.

Let s and $\mathcal{K}$ be the symbolic representations of a nonce and a set of keys associated with a specific thread in a trace $\langle e, \lambda \rangle$. Define $\Lambda(\mathcal{K})$ to be the set of bitstrings corresponding to the keys in $\mathcal{K}$, *i.e.*, $\{\lambda(k) \mid k \in \mathcal{K}\}$.

Definition 1 (Secretive Trace). *A trace $\langle e, \lambda \rangle$ is a* secretive trace *with respect to s and $\mathcal{K}$ if the following properties hold for every thread belonging to honest principals:*

- *a thread which generates a new nonce r in e, with $\lambda(r) = \lambda(s)$, ensures that r is encrypted with a key k with bitstring representation $\lambda(k) \in \Lambda(\mathcal{K})$ in any message sent out.*
- *whenever a thread decrypts a message with a key k with $\lambda(k) \in \Lambda(\mathcal{K})$, which was produced by encryption with key k by an honest party, and parses the decryption, it ensures that the results are encrypted with some key k' with $\lambda(k') \in \Lambda(\mathcal{K})$ in any message sent out.*

To lift this definition of secretive traces to secretive protocols we need a way to identify the symbol s and the set of symbols $\mathcal{K}$ in each protocol execution trace. We do this by assuming functions $\bar{s}$ and $\bar{\mathcal{K}}$ that map a trace to symbols in the trace corresponding to s and the set of keys in $\mathcal{K}$ respectively. In applications, these mappings will be given by the semantics of logical formulas.

Definition 2 (Secretive Protocol). *Given the mappings $\bar{s}$ and $\bar{\mathcal{K}}$, a protocol $\mathcal{Q}$ is a* secretive protocol *with respect to s and $\mathcal{K}$ if for all probabilistic poly-time adversaries $\mathcal{A}$ and for all sufficiently large security parameters η, the probability that a trace t, generated by the interaction of $\mathcal{A}$ with principals following roles of $\mathcal{Q}$, is a secretive trace with respect to $\bar{s}(t)$ and $\bar{\mathcal{K}}(t)$ is overwhelmingly close to 1, the probability being taken over all adversary and protocol randomness.*

In proving properties of secretive protocols, we focus on the subset of protocol traces that are secretive. Adversary advantages retain the same asymptotic behavior over this set because non-secretive traces are a negligible fraction of all traces.

The general structure of the proofs of the secrecy theorems is by reduction of the appropriate protocol secrecy game to a multi-party IND-CCA game: given protocol adversary $\mathcal{A}$, we construct an adversary $\mathcal{A}'$ against a multi-party IND-CCA challenger which provides $|\mathcal{K}|$-party Left-or-Right encryption oracles

[3] Some of the results here were presented in the informal WITS'07 [30] workshop.

$\mathcal{E}_{k_i}(LoR\,(\cdot,\cdot,b))$ parameterized by a challenge bit b and decryption oracles $\mathcal{D}_{k_i}(\cdot)$ for all $k_i \in \mathcal{K}$ (Following [9], $LoR(m_0, m_1, b)$ is a function which returns m_b). We use multi-party security definitions due to Bellare, Boldyreva and Micali [9] applied to symmetric encryption schemes. [4]

The strategy of $\mathcal{A}'$ is to provide a simulation of the *secretive protocol* to $\mathcal{A}$ by using these oracles such that the capability of $\mathcal{A}$ to break the indistinguishability or key usability of the nonce can be leveraged in some way to guess the challenge bit b of the multi-party IND-CCA challenger. To this end, $\mathcal{A}'$ employs a *bilateral simulator* $\mathcal{S}$ which randomly chooses two bit-strings s_0, s_1 as alternate representations of the putative secret s and then simulates execution of the protocol to the protocol adversary $\mathcal{A}$ for both the representations.

As with the execution of the actual protocol, $\mathcal{S}$ receives messages and scheduling information from $\mathcal{A}$ and acts according to the roles of the given protocol. The difference from a normal protocol execution is that in computing bitstring representations of terms that involve s, $\mathcal{S}$ does so for both representations of s. We will show that for secretive protocols the representation of s that $\mathcal{A}$ sees is determined by the challenge bit b of the CCA challenger. The operational semantics of the bilateral simulator is formally described in the full version of [30]. We explain the form of the definition using an example.

$$\frac{\triangleright\ \mathsf{m}' \quad \triangleright\ \mathsf{m}'' \quad \mathsf{m} := \mathtt{pair}\ \mathsf{m}', \mathsf{m}'';}{\triangleright\ \mathsf{m},\ lv(\mathsf{m}) = pair(lv(\mathsf{m}'), lv(\mathsf{m}'')),\ rv(\mathsf{m}) = pair(rv(\mathsf{m}'), rv(\mathsf{m}''))}$$

The notation $\triangleright\ \mathsf{m}$ means that the symbol m has been computationally evaluated according to the semantics. The premise of the rule requires that the symbols m and m' have already been evaluated and we are considering the action $\mathsf{m} := \mathtt{pair}\ \mathsf{m}', \mathsf{m}''$ in some thread. The functions lv and rv map a symbol to its bitstring values corresponding to the representations s_0 and s_1 of s respectively. The function *pair* is the actual computational implementation of pairing. The conclusion of the rule states that $lv(\mathsf{m})$ is evaluated by pairing the bit-strings $lv(\mathsf{m}')$ and $lv(\mathsf{m}'')$ and similarly for $rv(\mathsf{m})$. In simulating the protocol to the protocol adversary, the simulator executes each action of the currently scheduled thread following this definition.

Suppose m is a term explicitly constructed from s. As $\mathcal{S}$ is simulating a *secretive protocol*, this term is to be encrypted with a key k in $\mathcal{K}$ to construct a message to be sent out to $\mathcal{A}$. So, $\mathcal{S}$ asks the encryption oracle of the $|\mathcal{K}|$-IND-CCA challenger to encrypt $(lv(m), rv(m))$ with k. In addition, this pair of bitstrings is recorded and the result of the query is logged in the set qdb_k. If a message construction involves decryption with a key in $\mathcal{K}$, $\mathcal{S}$ first checks whether the term to be decrypted was produced by an encryption oracle by accessing the log qdb_k—if not, then the decryption oracle is invoked; if yes, then $\mathcal{S}$ uses the corresponding encryption query as the decryption. In the second case the encryption query must have been of the form (m_0, m_1). Following the definition of *secretive protocol*, terms constructed from this decryption will be re-encrypted with a key in $\mathcal{K}$ before sending out. Thus we note here that all

[4] In [9], IND-CCA2 security and multi-party IND-CCA security are shown to be asymptotically equivalent.

such replies will be consistent to $\mathcal{A}$ with respect to any choice of b. The situation becomes trickier when encryption or decryption of a term is required with s as the key. In this case $\mathcal{S}$ encrypts or decrypts with s_0. We therefore always have $lv(m) = rv(m)$ for any message m being sent out.

One subtle issue arises when we consider term deconstructors such as unpairings, decryptions, and pattern matching actions: we need to ensure that the success of such actions are independent of the challenge bit b The type information carried by terms (mentioned in Section 2) ensures this consistency in an overwhelming number of traces. The proofs proceed by induction over the operational semantics of the simulator.

Theorem 1 (CCA security - level 1). *Assume that a probabilistic poly-time adversary interacts with a secretive protocol with respect to nonce s and a set of level-0 keys $\mathcal{K}$.*

- Key indistinguishability*: If s is not used as a key by the honest principals, the adversary has negligible advantage at distinguishing s from random after the interaction provided the encryption scheme is IND-CCA secure.*
- Key usability*: If the honest principals use s as a key, the adversary has negligible advantage at winning an IND-CCA game against a symmetric encryption challenger using the key s after the interaction provided the encryption scheme is IND-CCA secure.*

A *level-0* key for a protocol execution is an encryption key which is only used as a key but never as a payload. We now extend Theorem 1 to directed key hierarchies to reason about key distribution protocols such as Kerberos.

Let $\mathcal{K}$ be the symbolic representations of nonces and keys associated with a specific thread in a trace $\langle e, \lambda \rangle$. The *key graph* of $\mathcal{K}$ in a protocol is a directed graph with keys in $\mathcal{K}$ as vertices. There is an edge from key k_1 to k_2 if the protocol is secretive with respect to k_2 and a key set which includes k_1. Consider a directed acyclic key graph. Keys at the root are *level 0* keys. The *level* of any other key is one more than the maximum level among its immediate predecessors. For a set of keys $\mathcal{K}$ from a directed acyclic key graph, we define its *closure* $\mathcal{C}(\mathcal{K})$ to be the union of sets of keys at the root which are predecessors of each key in $\mathcal{K}$.

Theorem 2 (CCA security - Key DAGs). *Assume that a probabilistic poly-time adversary interacts with a secretive protocol with respect to nonce s and a set of keys $\mathcal{K}$ in a DAG (Directed Acyclic Graph) of finite and statically bounded level.*

- Key indistinguishability*: If s is not used as a key by the honest principals, the adversary has negligible advantage at distinguishing s from random after the interaction provided the encryption scheme is IND-CCA secure.*
- Key usability*: If the honest principals use s as a key, the adversary has negligible advantage at winning an IND-CCA game against a symmetric encryption challenger using the key s after the interaction provided the encryption scheme is IND-CCA secure.*

The following theorem establishes the integrity of encryptions done with nonces protected by key hierarchies. The security definition INT-CTXT for ciphertext integrity is due to [10] and also referred to as *existential unforgeability* of ciphertexts in [23].

Theorem 3 (CTXT integrity). *Assume that a probabilistic poly-time adversary interacts with a secretive protocol with respect to nonce s and a set of keys $\mathcal{K}$ in a DAG of finite, statically bounded levels. During the protocol run, if an honest principal decrypts a ciphertext with key s successfully, then with overwhelming probability the ciphertext was produced by an honest principal by encryption with s provided the encryption scheme is IND-CCA and INT-CTXT secure.*

4 Proof System

In this section, we present a general induction rule, axiomatize the informal definition of a *secretive protocol* given in Section 3 and formulate axioms stating that *secretive protocols* guarantee certain computational properties. The soundness proofs of these axioms are based on the theorems in Section 3.

4.1 Establishing Secretive Protocols

We introduce the predicate $\mathsf{Good}(X, m, s, \mathcal{K})$ to assert that the thread X constructed the term m in accordance with the rules allowing a *secretive protocol* with respect to nonce s and set of keys $\mathcal{K}$ to send out m. More formally, $[\![\mathsf{Good}(X, m, s, \mathcal{K})]\!](T, D, \epsilon)$ is the collection of all traces $t \in T$ where thread X constructs the term m in a 'good' way. Received messages, data of atomic type different from nonce or key, nonces different from s are all 'good' terms. Constructions that are 'good' consist of pairing or unpairing good terms, encrypting good terms, encrypting any term with a key in $\mathcal{K}$ and decrypting good terms with keys not in $\mathcal{K}$. The following axioms formalize reasoning about the Good predicate by induction on actions in protocol roles.

G0 $\mathsf{Good}(X, a, s, \mathcal{K})$, if a is of an atomic type different from nonce

G1 $\mathsf{New}(Y, n) \wedge n \neq s \supset \mathsf{Good}(X, n, s, \mathcal{K})$

G2 $[\mathtt{receive}\ m;]_X\ \mathsf{Good}(X, m, s, \mathcal{K})$

G3 $\mathsf{Good}(X, m, s, \mathcal{K})\ [\mathsf{a}]_X\ \mathsf{Good}(X, m, s, \mathcal{K})$, for all actions a

G4 $\mathsf{Good}(X, m, s, \mathcal{K})\ [\mathtt{match}\ m\ \mathtt{as}\ m';]_X\ \mathsf{Good}(X, m', s, \mathcal{K})$

G5 $\mathsf{Good}(X, m_0, s, \mathcal{K}) \wedge \mathsf{Good}(X, m_1, s, \mathcal{K})\ [m := \mathtt{pair}\ m_0, m_1;]_X\ \mathsf{Good}(X, m, s, \mathcal{K})$

G6 $\mathsf{Good}(X, m, s, \mathcal{K})\ [m' := \mathtt{symenc}\ m, k;]_X\ \mathsf{Good}(X, m', s, \mathcal{K})$

G7 $k \in \mathcal{K}\ [m' := \mathtt{symenc}\ m, k;]_X\ \mathsf{Good}(X, m', s, \mathcal{K})$

G8 $\mathsf{Good}(X, m, s, \mathcal{K}) \wedge k \notin \mathcal{K}\ [m' := \mathtt{symdec}\ m, k;]_X\ \mathsf{Good}(X, m', s, \mathcal{K})$

In the following lemma, the additional field σ in the trace definition refers to an environment that maps free variables in a formula to bitstrings. The proof is by induction on the construction of 'good' terms.

Lemma 1. *If* $\mathsf{Good}(X, m, s, \mathcal{K})$ *holds for a trace* $\langle e, \lambda, \cdots, \sigma \rangle$*, then any bilateral simulator with parameters* $s, \mathcal{K}$*, executing symbolic actions* e *produces identical bitstring representations for* m *on both sides of the simulation,* i.e.*, we will have* $\triangleright\, m$ *and* $lv(m) = rv(m)$*.*

The formula $\mathsf{SendGood}(X, s, \mathcal{K})$ asserts that all messages that thread X sends out are good and $\mathsf{Secretive}(s, \mathcal{K})$ asserts that all honest threads only send out good messages. Formally,

$$\mathsf{SendGood}(X, s, \mathcal{K}) \equiv \forall m.\, (\mathsf{Send}(X, m) \supset \mathsf{Good}(X, m, s, \mathcal{K}))$$
$$\mathsf{Secretive}(s, \mathcal{K}) \equiv \forall X.\, (\mathsf{Honest}(\hat{X}) \supset \mathsf{SendGood}(X, s, \mathcal{K}))$$

The axioms $\mathbf{SG0} - \mathbf{2}$ are based on the definition of $\mathsf{SendGood}$:

SG0 $\mathsf{Start}(X)\, [\,]_X\, \mathsf{SendGood}(X, s, \mathcal{K})$

SG1 $\mathsf{SendGood}(X, s, \mathcal{K})\, [\mathtt{a}]_X\, \mathsf{SendGood}(X, s, \mathcal{K})$, where $\mathtt{a}$ is not a send.

SG2 $\mathsf{SendGood}(X, s, \mathcal{K})\, [\mathtt{send}\ m;]_X\, \mathsf{Good}(X, m, s, \mathcal{K}) \supset \mathsf{SendGood}(X, s, \mathcal{K})$

SG1 is obviously valid for nonce generation, message receipt, encryption and pairing actions. Soundness for unpairing and decryption requires consistency of deconstructions in the bilateral simulation, e.g. unpairing should succeed on one side iff it succeeds on the other. Soundness of **SG2** follows from the operational semantics of the simulator on a send action and Lemma 1.

The $\mathbf{IND}_{GOOD}$ rule which follows states that if all honest threads executing some basic sequence in the protocol locally construct good messages to be sent out, given that they earlier also did so, then we can conclude $\mathsf{Secretive}(s, \mathcal{K})$.

$$\mathbf{IND}_{GOOD} \quad \forall \rho \in \mathcal{Q}. \forall P \in BS(\rho).\ \frac{\mathsf{SendGood}(X, s, \mathcal{K})\, [P]_X\, \Phi \supset \mathsf{SendGood}(X, s, \mathcal{K})}{\mathcal{Q} \vdash \Phi \supset \mathsf{Secretive}(s, \mathcal{K})}\ (*)$$

$(*)$: $[P]_X$ does not capture free variables in Φ, $\mathcal{K}$, s, and Φ is a prefix closed trace formula.

A set of basic sequences (BS) of a role is any partition of the sequence of actions in a role such that if any element sequence has a `receive` then it is only at its beginning. The formula Φ has to be *prefix closed* which means that it is a formula such that if it is true at some point in a trace, it is also true at all earlier points. This rule is an instance of a more general induction rule **IND** which is obtained by replacing $\mathsf{SendGood}(X, s, \mathcal{K})$ by a general trace formula $\Psi(X)$ and requiring that $\mathsf{Start}(X)\, [\,]_X\, \Phi \supset \Psi(X)$. The instance of the latter formula, the base case of the induction, is trivially satisfied when $\Psi(X)$ is $\mathsf{SendGood}(X, s, \mathcal{K})$ because of axiom **SG0**.

4.2 Relating Secretive Protocols to Good Keys

The remaining axioms relate the concept of a secretive protocol, which is trace-based, to complexity theoretic notions of security. As defined in section 3, a level-0 key is only used as a key. Note that this is a syntactic property and is

evident from inspection of the protocol roles. Typically, a long-term key shared by two principals is level-0. A nonce is established to be a level-1 key when the protocol is proved to be a *secretive protocol* with respect to the nonce and a set of level-0 keys. This concept is extended further to define level-2 keys.

The formula $\mathsf{InInitSet}(X, s, \mathcal{K})$ asserts X is either the generator of nonce s or a possessor of some key in the closure $\mathcal{C}(\mathcal{K})$. $\mathsf{GoodInit}(s, \mathcal{K})$ asserts that all such threads belong to honest principals. Formally,

$$\mathsf{InInitSet}(X, s, \mathcal{K}) \equiv \exists k \in \mathcal{C}(\mathcal{K}).\ \mathsf{Possess}(X, k) \vee \mathsf{New}(X, s)$$
$$\mathsf{GoodInit}(s, \mathcal{K}) \equiv \forall X.\ (\mathsf{InInitSet}(X, s, \mathcal{K}) \supset \mathsf{Honest}(\hat{X}))$$

Our objective is to state that secrets established by *secretive protocols*, where possibly the secrets are also used as keys, are good keys against everybody except the set of people who either generated the secret or are in possession of a key protecting the secret. The formula $\mathsf{GoodKeyFor}$ expresses this property. For level-0 keys that we want to claim are possessed only by honest principals we use the formula $\mathsf{GoodKey}$.

$$\mathsf{GoodKeyFor}(s, \mathcal{K}) \equiv \forall X.\ (\mathsf{GoodKeyAgainst}(X, s) \vee \mathsf{InInitSet}(X, s, \mathcal{K}))$$
$$\mathsf{GoodKey}(k) \equiv \forall X.\ (\mathsf{Possess}(X, k) \supset \mathsf{Honest}(\hat{X}))$$

For protocols employing an IND-CCA secure encryption scheme, the soundness of the following axiom follows from theorems 1 and 2:

$$\textbf{GK} \quad \mathsf{Secretive}(s, \mathcal{K}) \wedge \mathsf{GoodInit}(s, \mathcal{K}) \Rightarrow \mathsf{GoodKeyFor}(s, \mathcal{K})$$

If the encryption scheme is both IND-CCA and INT-CTXT secure then, the soundness of the following axioms follow from Theorem 3:

$$\textbf{CTX0} \quad \mathsf{GoodKey}(k) \wedge \mathsf{SymDec}(Z, E_{sym}[k](m), k) \Rightarrow \exists X.\ \mathsf{SymEnc}(X, m, k),$$
for level-0 key k.
$$\textbf{CTXL} \quad \mathsf{Secretive}(s, \mathcal{K}) \wedge \mathsf{GoodInit}(s, \mathcal{K}) \wedge \mathsf{SymDec}(Z, E_{sym}[s](m), s)$$
$$\Rightarrow \exists X.\ \mathsf{SymEnc}(X, m, s)$$

The **soundness theorem** is proved by showing that every axiom is a valid formula and that all proof rules preserve validity. Proofs for selected axioms are given in the full version of the paper [29].

Theorem 4 (Soundness). $\forall \mathcal{Q}, \varphi.$ *if* $\mathcal{Q} \vdash \varphi$ *then* $\mathcal{Q} \vDash \varphi$

Compositional Reasoning We develop composition theorems that allow *secretive*-ness proofs of compound protocols to be built up from proofs of their parts. We consider three kinds of composition operations on protocols—*parallel*, *sequential*, and *staged*—based on our previous work [16, 21]. However, adapting that approach for reasoning about secrecy requires new insights. One central concept in these 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 [16] the "honesty

rule" **HON** is used for establishing invariants, reasoning about *secretive*-ness requires a more general form of induction, captured in this paper by the **IND** rule. Also, in proving that a protocol step does not violate *secretive*-ness, we need to employ derivations from earlier steps executed by the principal. In the technical presentation, this history information shows up as preconditions in the secrecy induction of the sequential and staged composition theorems. The statement of the theorems is similar to the theorems proved for the symbolic model in earlier work [31], but the proofs use the computational semantics. In particular we need a staged composition operation that 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 [25], IEEE 802.11i [1], and IKEv2 [24], with backward arcs usually corresponding to error handling or rekeying.

5 Analysis of Kerberos

Table 2 lists the security properties of Kerberos that we prove. 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 a good key for certain principals. For example, $AUTH_{kas}^{client}$ states that when C finishes executing the **Client** role, some thread of $\hat{K}$ indeed sent the expected message; SEC_{akey}^{client} states that the authorization key is good after execution of the **Client** role by C; the other security properties are analogous. We abbreviate the honesty assumptions by defining $\mathsf{Hon}(\hat{X_1}, \hat{X_2}, \cdots, \hat{X_n}) \equiv \mathsf{Honest}(\hat{X_1}) \wedge \mathsf{Honest}(\hat{X_2}) \wedge \cdots \mathsf{Honest}(\hat{X_n})$. The formal proofs are omitted from this paper but present in the full version [29].

The overall proof structure demonstrates an interleaving of authentication and secrecy properties, reflecting the intuition behind the protocol design. We start with proving some authentication properties based on the presumed secrecy of long-term shared symmetric keys. As intended in the design, these authentication guarantees enable us to prove the secrecy of data protected by the long-term keys. This general theme recurs further down the protocol stages. Part of the data is used in subsequent stages as an encryption key. The secrecy of this transmitted encryption key lets us establish authentication in the second stage of the protocol. The transmitted key is also used to protect key exchange in this stage - the secrecy of which depends on the authentication established in the stage.

Theorem 5 (KAS Authentication). *On execution of the* **Client** *role by a principal, it is guaranteed with asymptotically overwhelming probability that the intended KAS indeed sent the expected response assuming that both the client and the KAS are honest. A similar result holds for a principal executing the* **TGS** *role. Formally,* $KERBEROS \vdash AUTH_{kas}^{client}, AUTH_{kas}^{tgs}$.

Authentication is achieved by the virtue of ciphertext integrity offered by the symmetric encryption scheme. At a high level, we reason that a ciphertext could have been produced only by one of the possessors of the corresponding key.

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

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

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

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

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

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

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

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

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

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

Table 2. Kerberos Security Properties

Theorem 6 (Authentication Key Secrecy). *On execution of the* **Client** *role by a principal, the Authentication Key is guaranteed to be good, in the sense of IND-CCA security, 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. Observe that in the first stage, the KAS sends out $AKey$ encrypted under two different keys - $k^{c \to k}_{C,K}$ and $k^{t \to k}_{T,K}$, and the client uses $AKey$ as an encryption key. As a first approximation we conjecture that in the entire protocol execution, $AKey$ is either protected by encryption with either of the keys in $\mathcal{K} = \{k^{c \to k}_{C,K}, k^{t \to k}_{T,K}\}$ or else used as an encryption key in messages sent to the network by honest principals. This seems like a claim to be established by induction. As a base case, we establish that the generator of $AKey$ (some thread of the KAS) satisfies the conjecture. The induction case is: whenever an honest principal decrypts a ciphertext with one of the keys in $\mathcal{K}$, it ensures that new terms generated from the decryption are re-encrypted with some key in $\mathcal{K}$ in any message sent out. The results (of the appropriate type) from such a decryption are however, allowed to be used as encryption keys, which as you can note is the case in the first stage of the client.

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

In the formal proof, we show that Kerberos is a *secretive protocol* with respect to the nonce $AKey$ and the set of keys $\mathcal{K}$. The induction idea is captured,

in its simplest form, by the proof rule IND_{GOOD}. However, as Kerberos has a staged structure we use the staged composition theorem which builds upon the rule IND_{GOOD}. The core of the proof is the *secrecy induction* which is an induction over all the basic sequences of all the protocol roles. The authentication condition Φ is easily derived from the KAS Authentication theorem (theorem 5). The staged composition theorem allows us to facilitate the secrecy induction by obtaining inferences from the information flow induced by the staged structure of Kerberos in a simple and effective way. The secrecy induction is modular as the individual basic sequences are small in themselves. Goodness of $AKey$ now follows from theorem 1 (CCA security - level 1), which is formally expressed by axiom **GK**.

Theorem 7 (TGS Authentication). *On execution of the* **Client** *role by a principal, it is guaranteed with asymptotically overwhelming probability that the intended TGS indeed sent the expected response assuming that the client, the KAS and the TGS are all honest. A similar result holds for a principal executing the* **Server** *role. Formally,* $KERBEROS \vdash AUTH_{tgs}^{client}, AUTH_{tgs}^{server}$.

Theorem 8 (Service Key Secrecy). *On execution of the* **Client** *role by a principal, the Service Key is guaranteed to be good, in the sense of IND-CCA security, assuming that the client, the KAS, the TGS and the application server are all honest. A similar result holds for a principal executing the* **TGS** *role. Formally,* $KERBEROS \vdash SEC_{skey}^{client}, SEC_{skey}^{tgs}$.

The proof of $AUTH_{tgs}^{server}$ is similar to the proof of theorem 5. The proof of $AUTH_{tgs}^{client}$ depends on the 'goodkey'-ness of $AKey$ established by theorem 6. For theorem 8, the idea is that the Service Key $SKey$ is protected by level-0 key $k_{S,T}^{s \to t}$ and level-1 key $AKey$. The proof of 'Secretive'-ness proceeds along the same line as for theorem 6 and uses derivations from theorem 7. Then we invoke axiom **GK** to establish $KERBEROS \vdash SEC_{skey}^{client}, SEC_{skey}^{tgs}$.

Kerberos with PKINIT. In the first stage of Kerberos with PKINIT [33], the KAS establishes the authorization key encrypted with a symmetric key which in turn is sent to the client encrypted with its public key. Since the protocol uses both public and symmetric keys at level 0, we formulate a definition of a joint public-symmetric key game. We then extend the proof system and prove all the syntactically analogous properties of the PKINIT version.

6 Conclusion

Computational secrecy properties, such as indistinguishability and suitability of a key ("GoodKey"), are not trace-based properties, making it awkward to reason inductively or compositionally about them. We therefore formulate the *secretive* trace-based property and prove that any secretive protocol can be used to construct a generic reduction from protocol attacks to attacks on underlying

primitives. This allows computational secrecy to be established by direct inductive reasoning about a relatively natural and intuitive trace-based property.

A second contribution of the paper is a proof system for secrecy, in a formal logic based on inductive reasoning about protocol actions carried out by honest parties (only). We illustrate the power of this system by giving a modular, formal proof of computational authentication and secrecy properties of the Kerberos V5 protocol, thus addressing an open problem posed in [3]. Other proofs have been carried out, such as for a protocol that poses a challenge for the rank function method [19], but are omitted due to space constraints.

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. Abadi and P. Rogaway. Reconciling two views of cryptography (the computational soundness of formal encryption). *Journal of Cryptology*, 15(2):103–127, 2002.
3. 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.
4. M. Backes and B. Pfitzmann. Limits of the cryptographic realization of XOR. In *Proc. of the 10th European Symposium on Research in Computer Security*. Springer-Verlag, 2005.
5. M. Backes and B. Pfitzmann. Relating symbolic and cryptographic secrecy. In *Proc. IEEE Symposium on Security and Privacy*, pages 171–182. IEEE, 2005.
6. M. Backes, B. Pfitzmann, and M. Waidner. A universally composable cryptographic library. Cryptology ePrint Archive, Report 2003/015, 2003.
7. M. Backes, B. Pfitzmann, and M. Waidner. Limits of the reactive simulatability/uc of dolev-yao models with hashes. In *Proc. of the 11th European Symposium on Research in Computer Security*. Springer-Verlag, 2006.
8. M. Baudet, V. Cortier, and S. Kremer. Computationally Sound Implementations of Equational Theories against Passive Adversaries. In *Proceedings of the 32nd ICALP*, 2005.
9. M. Bellare, A. Boldyreva, and S. Micali. Public-key encryption in a multi-user setting: Security proofs and improvements. In *Advances in Cryptology - EUROCRYPT 2000, Proceedings*, pages 259–274, 2000.
10. M. Bellare and C. Namprempre. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. In *ASIACRYPT*, pages 531–545, 2000.
11. M. Bellare and P. Rogaway. Entity authentication and key distribution. In *Proceedings of the 13th Annual International Cryptology Conference on Advances in Cryptology (CRYPTO '93)*, pages 232–249. Springer-Verlag, 1994.
12. F. Butler, I. Cervesato, A. D. Jaggard, and A. Scedrov. Verifying confidentiality and authentication in kerberos 5. In *ISSS*, pages 1–24, 2003.
13. R. Canetti and J. Herzog. Universally composable symbolic analysis of mutual authentication and key-exchange protocols. In *Proceedings of TCC 2006*, pages 380–403, 2006.

14. V. Cortier and B. Warinschi. Computationally sound, automated proofs for security protocols. In *Proceedings of 14th European Symposium on Programming (ESOP)*, pages 157–171, 2005.
15. A. Datta, A. Derek, J. Mitchell, A. Ramanathan, and A. Scedrov. Games and the impossibility of realizable ideal functionality. In *Theory of Cryptography Conference - Proceedings of TCC 2006*, 2006.
16. A. Datta, A. Derek, J. C. Mitchell, and D. Pavlovic. A derivation system and compositional logic for security protocols. *Journal of Computer Security*, 2005.
17. A. Datta, A. Derek, J. C. Mitchell, V. Shmatikov, and M. Turuani. Probabilistic polynomial-time semantics for a protocol security logic. In *Proceedings of ICALP '05*, pages 16–29, 2005.
18. A. Datta, A. Derek, J. C. Mitchell, and B. Warinschi. Computationally sound compositional logic for key exchange protocols. In *Proceedings of 19th IEEE Computer Security Foundations Workshop*, pages 321–334. IEEE, 2006.
19. R. Delicata and S. A. Schneider. Towards the rank function verification of protocols that use temporary secrets. In *WITS*, 2004.
20. F. J. T. Fábrega, J. C. Herzog, and J. D. Guttman. Strand spaces: Why is a security protocol correct? In *Proceedings of the 1998 IEEE Symposium on Security and Privacy*, pages 160–171, Oakland, CA, May 1998. IEEE Computer Society Press.
21. C. He, M. Sundararajan, A. Datta, A. Derek, and J. C. Mitchell. A modular correctness proof of IEEE 802.11i and TLS. In *CCS '05: Proceedings of the 12th ACM conference on Computer and communications security*, 2005.
22. J. Heather. Strand spaces and rank functions: More than distant cousins. In *Proceedings of CSFW*, page 104, 2002.
23. J. Katz and M. Yung. Unforgeable encryption and chosen ciphertext secure modes of operation. In *FSE*, pages 284–299, 2000.
24. C. Kauffman. Internet key exchange (IKEv2) protocol. IETF Internet draft, 1994.
25. J. Kohl and B. Neuman. The Kerberos network authentication service (version 5). IETF RFC 1510, September 1993.
26. C. Meadows. A model of computation for the NRL protocol analyzer. In *Proceedings of 7th IEEE Computer Security Foundations Workshop*, pages 84–89. IEEE, 1994.
27. D. Micciancio and B. Warinschi. Soundness of formal encryption in the presence of active adversaries. In *Proceedings of TCC*, 2004.
28. D. Pavlovic and C. Meadows. Deriving secrecy properties in key establishment protocols. In *Proceedings of ESORICS*, 2006.
29. A. Roy, A. Datta, A. Derek, and J. C. Mitchell. Inductive proofs of computational secrecy. http://www.stanford.edu/∼arnab/rddm-InductiveProofs.pdf, 2007.
30. A. Roy, A. Datta, A. Derek, and J. C. Mitchell. Inductive trace properties for computational security. *WITS*, 2007. Full version at http://www.stanford.edu/∼arnab/rddm-IndTraceProps.pdf.
31. A. Roy, A. Datta, A. Derek, J. C. Mitchell, and J.-P. Seifert. Secrecy analysis in protocol composition logic. In *Proceedings of 11th Annual Asian Computing Science Conference*, 2006. To appear.
32. B. Warinschi. A computational analysis of the Needham-Schroeder(-Lowe) protocol. In *Proceedings of 16th Computer Science Foundation Workshop*, pages 248–262. ACM Press, 2003.
33. L. Zhu and B. Tung. Public key cryptography for initial authentication in kerberos, 2006. Internet Draft.