An Encapsulated Authentication Logic for Reasoning About Key Distribution Protocols*

Iliano Cervesato
Tulane University
iliano@math.tulane.edu

Catherine Meadows
Naval Research Laboratory
meadows@itd.nrl.navy.mil

Dusko Pavlovic
Kestrel Institute
dusko@kestrel.edu

Abstract

Authentication and secrecy properties are proved by very different methods: the former by local reasoning, leading to matching knowledge of all principals about the order of their actions, the latter by global reasoning towards the impossibility of knowledge of some data. Hence, proofs conceptually decompose in two parts, each encapsulating the other as an assumption. From this observation, we develop a simple logic of authentication that encapsulates secrecy requirements as assumptions. We apply it within the derivational framework to derive a large class of key distribution protocols based on the authentication properties of their components.

1 Introduction

Secrecy and authentication are the two main properties guaranteed by cryptographic protocols. Since they are highly dependent upon each other, they are commonly intertwined in protocol analysis. Indeed, early attempts to develop logics that reason only about authentication, such as BAN logic [2], were believed to be doomed to failure because the need to draw unverifiable conclusions about secrecy from authentication guarantees led to confusion and misunderstanding. It caused such problems as Lowe's attack [9] on the Needham-Schroeder public key protocol [12], which BAN had proved secure.

On the other hand, restricting one's reasoning to authentication alone has many benefits, if only it can be achieved. Authenticity properties specify conditions about the order of actions [7, 10]: if a particular action occurs, then some other set of actions must have occurred before it, and in a particular (partial) order. Thus, it is conceivable that a useful logic for protocol analysis could be developed that allows using a minimal set of primitives describing principal actions (e.g., sending and receiving messages and generating nonce, etc.) and reasoning about their order.

In this paper we present a logic that does just that. Unlike BAN logic, we do not avoid reasoning about secrecy entirely. Rather, we encapsulate the secrecy properties needed as proof obligations that can be proved by other means (we are developing a companion logic of secrecy to do just that). Our authentication logic reasons about the partial order of actions, Lamport-style. When a consequence of secrecy is needed, we include an assumption that defines a proof obligation which can be thought of as a system call to another verification method.

The logic has much in common with the compositional logic described in [5, 8]. Like that logic, it supports deriving complex protocols from simple ones using composition, refinement, and transformation. However, unlike that logic, in which predicates are defined as they are needed, the new logic relies on just three built-in predicates describing the ordering of events. It allows inferring the actions of other principals and their order based on one's local observations only: a fairly simple deductive process, even in our setting. It does not establish secrecy results, which use an entirely different flavor of logical inference: a form of inductive reasoning over the global knowledge of all principals. The clean separation of authentication proofs and secrecy proofs results in simple and elegant analyses.

We have applied an earlier version of this logic to the Group Domain of Interpretation (GDOI) Protocol [11]. Its simplicity turned out to be quite helpful in allowing us to identify a security flaw that careful reviews and another formal analysis had previously missed. The simplicity of the proof system allowed us to identify a key assumption being made that was not consistent with the protocol design. When the assumption was removed, the flaw was revealed.

We apply this logic to study the general mechanism of key distribution from an authentication perspective. Key distribution protocols feature complex interaction of secrecy and authentication requirements, and are therefore a prime target for our methodology. We start from an

*Supported by ONR and NSF.

abstract two-party key distribution protocol and, through refinements and transformations, we systematically derive the skeleton of such well-known protocols as shared-key Needham-Schroeder (NSSK) [12, 13], Denning-Sacco [6], and Kerberos 4 and 5 [14, 15]. An overview of the resulting taxonomy is displayed in Figure 1.

The main contribution of this work is twofold: (1) We formally separate the reasoning about authenticity from the reasoning about secrecy, and develop a simple logic of partial orders that supports the former. The secrecy inferences present in every proof of authentication are encapsulated as assumptions. (2) We use this logic within the derivational framework [5, 8] to perform a systematic analysis of an important branch of the family of key distribution protocols. This yields a novel classification of these protocols based on the mechanisms used to construct them and the properties they support. This promotes a deeper understanding of these fundamental protocols than individual analyses, and is readily extensible as new members are proposed.

This work is organized as follows: in Section 2 we explain our authentication logic. We use it to express the basic key distribution mechanism in Section 3. We extend it in the direction of NSSK in Section 4 with nonce-based recency and key confirmation. We extend it in a different direction with timestamp-based recency in section 5 obtaining the Denning-Sacco protocol as well as Kerberos 4 and 5. Section 6 concludes with statements of future work.

2 A Logic of Pure Authentication

As a principal A executes a protocol P, the events she observes locally (receiving a messages, comparing a component with an expected value, etc) allow her to make deductions about the actions of the principals she is interacting with. This implicitly identifies a class $\mathcal{Q}_A$ of possible runs, each of which intersperses her own actions with compatible actions by the other participants. As an authentication property Prop also identifies a class $\mathcal{Q}_{\mathsf{Prop}}$ of legal runs for P, the verification task traditionally reduces to showing that $\mathcal{Q}_A$ is contained in $\mathcal{Q}_{\mathsf{Prop}}$, and similarly for the other parties in the protocol. Every run in $\mathcal{Q}_A$ but not in $\mathcal{Q}_{\mathsf{Prop}}$ is an attack on A with respect to Prop.

We take a different approach: rather than comparing $\mathcal{Q}_A$ with the legal runs of a given authentication property, we synthesize a logical expression Φ describing $\mathcal{Q}_A$. This explicit representation is carefully engineered to be compositional: we dissect A's observations into elementary components and give a logical representation of the property they each realize (their *raison d'être* in a protocol). We similarly give a logical justification of the various mechanisms that allow combining components into bigger protocol fragments, and in particular of what properties emerge from the properties of the parts. By iterating this process all the way to A's original observations, we derive a formula, Φ, that in a strong sense describes *the* authentication properties of $\mathcal{Q}_A$. Indeed, this constructions provides us with a clear view of the properties contributed by each component and whether they propagate to Φ. We often restrict our attention to interesting scenarios by assuming, for example, that other principals behave honestly, or that a certain key has not been compromised. These assumptions are elective.

Rather than checking that a protocol satisfies a given property Prop, our approach enumerates the properties supported by a protocol based on its construction. Whenever an expected property is not manifested, we can rapidly point to a missing component or a composition mechanism failing to propagate it, and produce a counterexample, as done in [11]. We can also scrutinize the formula Φ summarizing the possible runs of each principal A in the light of a well-known authentication property, such as matching histories [7] or agreement [10].

2.1 Specifying Protocols

We begin by presenting a syntax to describe security protocols and enough of its execution semantics to define the notion of observation, a central concept in this work. The interested reader will find further details, as well as a semantics of protocol execution, in [4].

We use the letters A, B, and S to denote the *principals* participating in a given protocol. X, Y, ... will be variables ranging over principals. Principals exchange *messages*. This is modeled as an abstract term algebra $\mathcal{T}$ over a set of variables, constants and operators. Principals are a subclass of terms, and so are standard classes in crypto-protocols such as nonces, keys and timestamps. We write m for generic message, but use k, n, and t, for keys, nonces and timestamps. The letters $x, y, z, \ldots$ denote term variables. In this work, we will make use of two operators: (m, m') for the concatenation of m and m', and $(k\ m)$ for the encryption of m with shared key k. $\mathcal{T}$ may contain additional constructors, but we will not need them here.

Principals participate in a protocol by performing atomic *actions*. The actions we will rely on are summarized in the following table:

Action	*Form*	*Informal meaning*
send	$\langle m : A \to B \rangle$	The term m is sent, purportedly from A to B
receive	$(x : Y \to Z)$	Term, source and destination are received into x, Y and Z
match	$(m/p(\vec{x}))$	Term m is matched against term $p(\vec{x})$, binding $\vec{x}$
new	$(\nu\ x)$	A fresh value is created and stored in variable x
now	$(\tau\ x)$	The system time is read and stored in variable x

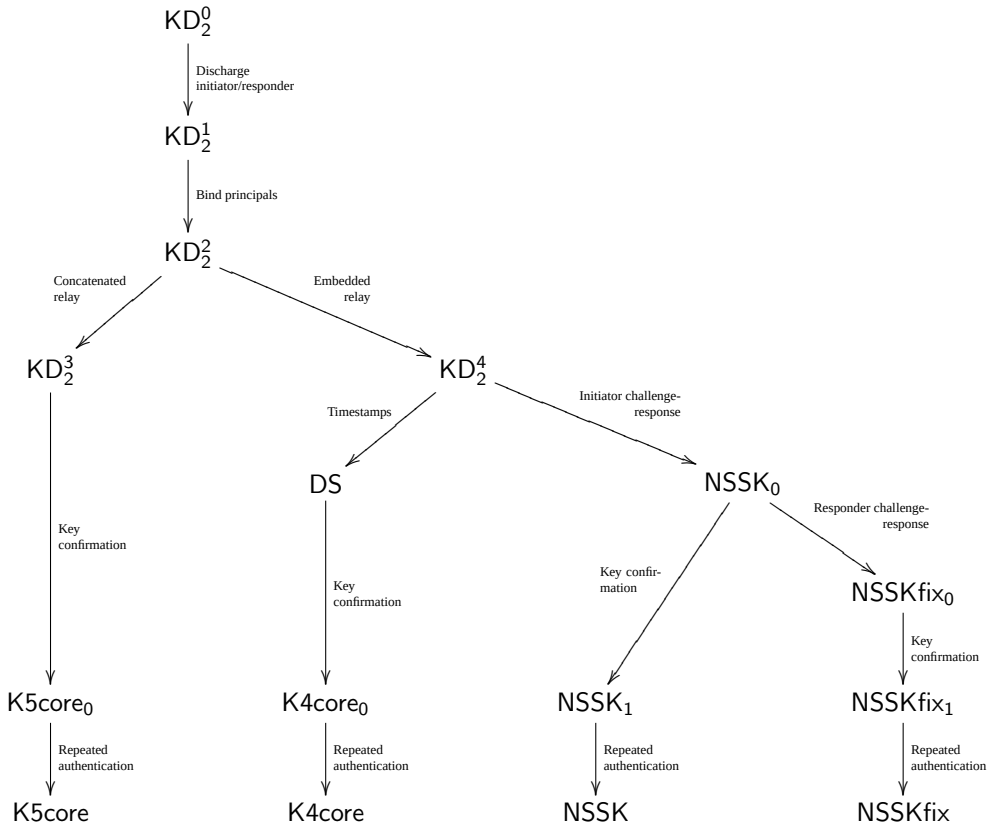


Figure 1. Overview of the Derivation of Key-Distribution Protocols

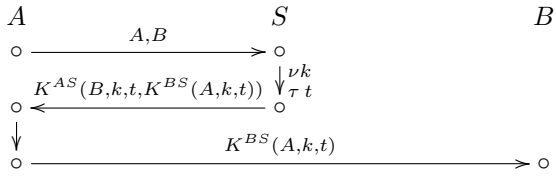
The variables x, Y, Z in **receive**, $\vec{x}$ in **match**, and x in **new** and **now** are binding occurrences: any subsequent mention in an expression involving actions are interpreted as bound by them. The semantics of actions is formalized in [4]. We will often use partial descriptions of actions, and elide e.g., the source and the destination, as in $\langle m \rangle$ or (y), and merge a reception and subsequent matches, writing (m).

A *role* is the complete code that a principal executes on her host to engage in a given protocol. We model a role as a collection of actions performed by a principal. We allow actions to be composed either sequentially (using ";" as a role constructor) or concurrently (using "$\otimes$"). The free variables of a role are its parameters, and should be instantiated prior to execution. The principal executing the role is a distinguished parameter. A *protocol* is a collection of roles. For example, the Denning-Sacco protocol [6] is specified as follows:

$$\begin{aligned}
\mathsf{DS_server}[S] &= (A, B : A \to S)\ ;\ (\nu\, k\ \otimes\ \tau\, t)\ ; \\
&\quad \langle K^{AS}(B, k, t, K^{BS}(A, k, t)) : S \to A \rangle \\
\mathsf{DS_init}[A; S, B] &= (K^{AS}(B, k, t, M) : S \to A)\ ; \\
&\quad \langle A, B : A \to S \rangle\ ;\ \langle M : A \to B \rangle \\
\mathsf{DS_resp}[B; S] &= (K^{BS}(A, k, t) : A \to B)
\end{aligned}$$

We will come back to this protocol in Section 5.

We will be primarily interested in the trace or *run* of a (possibly partial) execution, i.e., the set of actions executed by each principal and their relative ordering. For example the expected run of the Denning Sacco protocol is given as follows:



where we have liberally compacted our notation for succinctness. A run is a partial order over the set of *events* of the form a_A, where a is an action and A is the principal who has executed it. A *valid run* is subject to two conditions: (1) every receive event is preceded by a send in the partial order; (2) every match is successful. Given a valid run Q, the *local observation* of a principal A, denoted Q_A, is the restriction of Q to just the actions performed by A, and their relative order.

2.2 Reasoning about Authentication

We now set up a logic to draw inferences about valid runs. It will allow us to reconstruct the runs compatible with a principal's observations, often under assumptions.

We consider an instance of first-order logic with just three predicate forms:

a	*Event a has occurred*
$a < b$	*Event a has occurred before event b*
$a = b$	*a and b are the same event*

A formula combines these atomic predicates by means of the traditional connectives and quantifiers of first-order logic. Within an event, we omit the intended sender and recipient in a send or receive action when unimportant or easily reconstructible from the context. Additional abbreviations will make our discussion more clear and succinct:

This ...	...abbreviates ...
$(p)_A$	$(x)_A < (x/p)_A$
$(\!(p)\!)_A$	$(x)_A < (x/p')_A$ for p subterm of p'
$\langle\!\langle m \rangle\!\rangle_A$	$\langle m' \rangle_A$ for m subterm of m'
$\langle m \rangle_{A<}$	$\exists a = \langle m \rangle_A \wedge \forall b = \langle\!\langle m \rangle\!\rangle_B .\ a \leq b$
$\langle\!\langle m \rangle\!\rangle_{A<}$	$\exists a = \langle\!\langle m \rangle\!\rangle_A \wedge \forall b = \langle\!\langle m \rangle\!\rangle_B .\ a \leq b$
$a \prec b$	$b \Rightarrow a < b$

A valid run Q is immediately described by a formula Φ_Q consisting of a conjunction of the first two types of predicates above. An observation Q_A is similarly mapped to a formula Φ_A. Dually, an arbitrary formula Φ can be validated against a run Q, realizing a form of model checking [4].

In the sequel, we will define logical tools to complete the local observation Q_A of a principal A into a compatible run Q (which may or may not be the expected run). We will do so by building a tautology $\Phi_A \Rightarrow \Phi_Q$ which we abbreviate as $A : \Phi_Q$. Interesting completions will often require making assumptions Ψ, so that the general form of our statements will be $A : \Psi \Rightarrow \Phi_Q$.

2.3 Honesty and Secrecy Assumptions

We will often need a principal A deducing $A : \Phi$ to assume that another principal B is *honest* in order to draw interesting conclusions. By this, we mean that B does not deviate from his assigned role: if A ascertains that B has executed any action a_B in his role, she can be assured that he has executed all the actions preceeding a, or concurrent with it. The *honesty assumption* of the server of the Denning-Sacco protocol is as follows:

$$(A, B)_S \prec \begin{bmatrix} (\nu\, k)_S \\ (\tau\, t)_S \end{bmatrix} \prec \langle K^{AS}(B, k, t, K^{BS}(k, A, t)) \rangle_S$$

We abbreviate it as $\mathsf{honest}\ S$. We have extensively studied this notion in previous work [5, 11].

The *secrecy assumption* is novel: it allows specifying that certain keys have not been compromised. A shared key k is uncompromised for a group G of agents if the only principals that can perform an encryption or a decryption using k are the members of G. In symbols,

$$\begin{aligned} \mathsf{uncomp}(k, G) \quad \triangleq \quad & \langle\!\langle k\, m \rangle\!\rangle_{X<} \ \Rightarrow\ X \in G \\ \wedge\ & (x/k\, y)_X \ \Rightarrow\ X \in G \end{aligned}$$

Notice that the body of this definition expresses the semantics of shared-key cryptography: the first line says that only members of G can produce an encryption using k and send it in a message, the second says that only these principals can use the pattern $(k\ y)$ to access the contents of a term encrypted with k. Notice also that this expression defines the binding between a key and the principals who can use it.

uncomp acts as an interface between the logic of pure authentication developed in this paper, and the logic of secrecy which will be the subject of a sequel to this work. Here, we use it only as an *assumption*. There, we will be able to *prove* formulas of the form $\mathsf{uncomp}(k, G)$, which will permit discharging the assumption $\mathsf{uncomp}(k, G)$. This combination of logics will be particularly useful for studying staged protocols such as Kerberos, where a key is distributed for the purpose to protecting another key.

2.4 Axioms

We now describe some of the logical tools that allow extending a local observation into a compatible run. Most of these ideas have been extensively discussed elsewhere [5, 11], in which case we keep the presentation brief.

The *freshness axiom* (new) describes the behavior of the $(\nu\, n)$ action in logical terms:

$$\begin{aligned} (\nu\, n)_B \wedge a_A \Rightarrow (n \in FV(a)\ \Rightarrow\ & (\nu\, n)_B < a_A \\ \wedge\ (A \neq B \Rightarrow\ & (\nu\, n)_B < \langle n \rangle_B < (\!(n)\!)_A \leq a_A)) \end{aligned}$$

The first part says that ν is a binder, that is, any event a mentioning n necessarily occurs *after* $(\nu\ n)$. The second line requires that if the agent B executing $(\nu\ n)$ and the principal A executing a are different, then B must have used a send action to transmit n and A must have acquired it by means of a receive action.

The *receive axiom* (rcv) says that everything that is received must have been originated by someone:

$$A :\ (\!(m)\!)_A \Rightarrow\ \exists X.\ \langle\!\langle m \rangle\!\rangle_{X<} < (\!(m)\!)_A$$

The extensively studied *challenge-response axiom schema* (cr) [5, 11] abstractly describes the central concept of nonce-based challenge-response:

$$\begin{aligned} & A : \Phi' \ \wedge\ (\nu n)_A < \langle\!\langle c^{AX} n \rangle\!\rangle_{A<} < (\!(r^{AX} n)\!)_A \\ \Rightarrow\ & (\nu n)_A < \langle\!\langle c^{AX} n \rangle\!\rangle_{A<} < (\!(c^{AX} n)\!)_X < \langle\!\langle r^{AX} n \rangle\!\rangle_{X<} < (\!(r^{AX} n)\!)_A \end{aligned}$$

where c^{AX} is the challenge structure issued by A, r^{AX} is the corresponding response originated by X, and Φ' represents some additional precondition, usually an honesty or uncomp assumption.

One instance of (cr) that we will use in the sequel has c^{AX} be the identity (the nonce is sent in the clear), the response the encryption of the nonce with a key K^{AX} shared between A and X, and Φ' requiring K^{AX} not to be compromised for A and X. We obtain

$$A : \mathsf{uncomp}(K^{AX}, [A, X]) \wedge (\nu n)_A < \langle\!\langle n \rangle\!\rangle_{A<} < (\!(K^{AX} n)\!)_A$$
$$\Rightarrow\ (\nu n)_A < \langle\!\langle n \rangle\!\rangle_{A<} < (\!(n)\!)_X < \langle\!\langle K^{AX}\ n \rangle\!\rangle_{X<} < (\!(K^{AX}\ n)\!)_A$$

A proof of this instance of (cr) goes as follows: starting from A's own observations (the first line above), axiom (rcv) entails that some agent Y has originated $\langle\!\langle K^{AX}\ n \rangle\!\rangle$. By the uncomp assumption, Y must be either X or A; the latter possibility is excluded since no such actions occurs among A's observations. Axiom (new) completes the second line by sandwiching X's reception of n between A's transmission of the nonce and X's issuing of $\langle\!\langle K^{AX}\ n \rangle\!\rangle$.

The novel *timestamp axiom* (ts) describes the semantics of timestamps.

$$A : \mathsf{honest}\ X\ \ \wedge \langle\!\langle t \rangle\!\rangle_{X<} < (\!(t)\!)_A$$
$$\Rightarrow\ (\underline{\tau}\, t)_A < (\tau t)_X < \langle\!\langle t \rangle\!\rangle_{X<} < (\!(t)\!)_A < (\overline{\tau}\, t)_A$$

The antecedent of this formula assumes that A receives a message containing an acceptable timestamp t, and she has the certainty that an honest X has originated $\langle\!\langle t \rangle\!\rangle$. Given these hypotheses, she can deduce that X had indeed looked up t and sent it out, and that these actions took place within what she regards as the window of validity of this timestamp. Here, $(\underline{\tau}\ t)_A$ is the earliest point in time where A would accept t as valid, and $(\overline{\tau}\ t)_A$ is the dual upperbound. They are events internal to A representing time points calculated from t by considering what she deems as acceptable clock skews and network delays. What is important here is that they bound X's actions by events under A's control. In the sequel, we will discharge the assumption that A is certain that X has sent this timestamp whenever the message is authenticated.

Except for (new), all the axioms in this section are instances of the *send-receive axiom schema* (sr):

$$A\ :\ \exists X. \forall \vec{y}.\ (\!(f^{AX}(\vec{y}))\!)_A\ \wedge\ \Phi(X, \vec{y})$$
$$\Rightarrow\ \langle\!\langle f^{AX}(\vec{y}) \rangle\!\rangle_{X<} <\ (\!(f^{AX}(\vec{y}))\!)_A\ \wedge\ \Psi(X, \vec{y})$$

It says that A knows that, for some principal X, the message structure f^{AX} assures that, if she receives a message containing $f^{AX}(\vec{y})$, where X and $\vec{y}$ satisfy some precondition Φ, then X must have originated $f^{AX}(\vec{y})$, and that X and $\vec{y}$ satisfy some postcondition Ψ.

2.5 The Derivational Approach

Axioms suffice to extend a local observation into all of its compatible runs. Adopting a technique developed in previous work [5, 11], a more effective way to study a large protocol is to decompose it into elementary exchanges such as challenge-response and basic key distribution (see Section 3), derive the runs compatible with observations for these parts, and then reassemble them into formulas describing the compatible runs of the overall protocol. The logical tools that permit doing this are called refinements and transformations. A *refinement* reflects changes within one or more messages in a protocol into the formulas describing the runs compatible with an upgraded observation. For example, inserting a timestamp in a message has the effect of strengthening the recipient's guarantees. A *transformation* is similar but supports alterations to the exchange pattern of the protocol. For example, extending a protocol with an additional round of challenge-response is a transformation.

Besides the appeal of a divide-and-conquer methodology, the *derivational approach*, as this technique is known, supports reusing component-formulas pairs whenever they occur in another protocols. Moreover, the application of refinements and transformations can be made systematic, which gives rise to protocol taxonomies [5]: a rational classification of protocols that not only aids our understanding of these complex objects, but also helps choosing or devising a protocol based on desired features and properties. A tool is under development that will assist us building taxonomies that are much larger than what we have so far been able to construct by hand [1].

A detailed discussion of refinements, transformations and the derivational approach in general would be rather lengthy and technical. The reader is better served by consulting the existing literature [5]. Therefore, we will introduce these powerful tools only informally as we need them in the sequel. Section 5.1 gives some details of a new refinement.

3 Basic Key Distribution

In this section, we use the above logic to study the general mechanism of key distribution from an authentication perspective. Key distribution protocols feature complex interaction of secrecy and authentication requirements, and are therefore a prime target for our methodology. Indeed, their general goal is to authenticate two principals A and B to each other through communications with a server S along pre-established channels protected by secret keys. The distributed key protects future communications between A and B. We analyze the authentication aspects of such protocols, assuming that the keys to the server are not compromised. Proving that the distributed key is secret pertains to our forthcoming logic of secrecy, and is not addressed here.

For reasons of space, we give a complete proof of only the first derivation in this paper. The other proofs are anal-

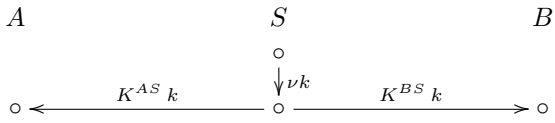
ogous. Other proofs and more detail may be found in [4].

We start with an abstract form of two-party key distribution, which we analyze from scratch, relying on axioms only. Then, through a series of refinements and transformations, we derive the skeleton of several well-known protocols, which we will further build up in the next sections.

In our first and most abstract protocol, KD_2^0, a server S spontaneously generates a key k and distributes it to known principals A and B encrypted with keys K^{AS} and K^{BS} he shares with each of them. The protocol is given by the following roles. The actions of A and B are symmetric at this stage (only A's role is shown).

$$\begin{array}{rcl} \mathsf{KD}_2^0\text{\_server}[S;A;B] & = & \nu\, k\ ; \quad \langle K^{AS}\, k : S \to A\rangle \\ & & \qquad \otimes\, \langle K^{BS}\, k : S \to B\rangle \\ \mathsf{KD}_2^0\text{\_client}[A;S,B] & = & (K^{AS}\, k : S \to A) \end{array}$$

The key server and the clients are given everybody's name as parameters to their respective roles. The expected run provides a clearer view of this exchange:



We will now take the point of view of each principal and infer a formula representing the runs that are compatible with its observations. We start with A (B is symmetric). The only event she observes is $(K^{AS}\ k : S \to A)$. Under the assumptions that K^{AS} is shared only by A and S and the honesty of S (derived from his role), A can reconstruct the expected run. The formal derivation is as follows:

$$\dfrac{\begin{array}{rl} \mathit{Obs} & : (K^{AS}\, k : S \to A)_A \\ (\mathsf{rcv}) & : \langle\!\langle K^{AS}\, k\rangle\!\rangle_{X<} < (K^{AS}\, k : S \to A)_A \\ \mathsf{uncomp} & : X = A \text{ or } X = S \\ \mathit{Obs} & : X \neq A \\ \mathsf{honest}\ S & : (\nu k)_S \prec \begin{bmatrix}\langle K^{AS} k : S \to A\rangle_{S<} \\ \langle K^{BS} k : S \to B\rangle_{S<}\end{bmatrix} \end{array}}{(\nu k)_S < \begin{bmatrix}\langle K^{AS} k : S \to A\rangle_{S<} \\ \langle K^{BS} k : S \to B\rangle_{S<}\end{bmatrix} < (K^{AS} k)_A}$$

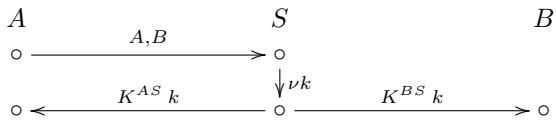
All the proofs in this paper have a similarly simple form. We will omit them for brevity. A compact representation of the overall formula follows:

$$\begin{array}{ll} A: & \mathsf{uncomp}(K^{AS}, [A,S]) \ \wedge\ \mathsf{honest}\ S\ \wedge (K^{AS}\ k)_A \\ \Rightarrow & (\nu\, k)_S < \begin{bmatrix}\langle K^{AS}\, k : S \to A\rangle_{S<} \\ \langle K^{BS}\, k : S \to B\rangle_{S<}\end{bmatrix} < \ (K^{AS}\ k)_A \end{array}$$

The server S does not conclude more than he observes since he is the recipient of no message.

In this protocol, each principal knows the identity of every other party ahead of time. This is not how typical key distribution protocols work: instead, one client, say A, takes the role of *initiator* by sending a message to the server saying she wants to communicate with B (the *responder*). This alteration is formalized by means of the *discharging transformation*, which replaces a parameter in a role with a received value.

We apply this transformation twice to protocol KD_0^2, discharging A and B as parameters to S's role. Before presenting the roles of the resulting protocol, KD_1^2, a glimpse at its expected run will help visualize what we have achieved:



The roles of KD_1^2 clearly show that S does not have A and B as parameters any more:

$$\begin{array}{rl} \mathsf{KD}_2^1\text{\_server}[S] & = (A, B : A \to S)\ ;\ \nu\, k\ ; \\ & \quad \langle K^{AS}\, k\colon S \to A\rangle \otimes \langle K^{BS}\, k\colon S \to B\rangle \\ \mathsf{KD}_2^1\text{\_init}[A;S,B] & = \langle A, B : A \to S\rangle\ ;\ (K^{AS}\, k : S \to A) \\ \mathsf{KD}_2^1\text{\_rsp}[B;S,A] & = (K^{BS}\, k : S \to B) \end{array}$$

Observe that the roles of A and B are not symmetric any more. Note also that it would make little difference if A transmitted just "B" as her first message since her name is present in the "from" field of this action.

The properties characterizing A's and B's views are summarized next.

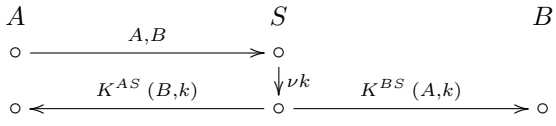
$$\begin{array}{l} A : \mathsf{uncomp}(K^{AS}, [A,S]) \ \wedge\ \mathsf{honest}\ S \ \wedge \\ \qquad\qquad \langle A, B\rangle_A \qquad\qquad < (K^{AS}\ k)_A \\ \Rightarrow \begin{bmatrix} \langle A, B\rangle_A \\ (A,X)_S < (\nu\, k)_S < \begin{bmatrix}\langle K^{AS}\, k\rangle_{S<} \\ \langle K^{XS}\, k\rangle_{S<}\end{bmatrix}\end{bmatrix} < (K^{AS}\ k)_A \end{array}$$

$$\begin{array}{l} B : \mathsf{uncomp}(K^{BS}, [B,S]) \ \wedge\ \mathsf{honest}\ S\ \wedge (K^{BS}\ k)_B \\ \Rightarrow\ (X,B)_S < (\nu\, k)_S < \begin{bmatrix}\langle K^{XS}\, k\rangle_{S<} \\ \langle K^{BS}\, k\rangle_{S<}\end{bmatrix} < (K^{BS}\ k)_B \end{array}$$

Observe that A has no way to determine whether S transmitted the key k to B or to some other party X. She can only infer that S received a request for a key involving herself and some X, not necessarily B. By a similar argument, B cannot ascertain to whom k was distributed even if A appears among the parameters of his role.

This problem is traditionally solved by having S include B's name into the message directed to A, and A's name into B's message. The *binding refinement* of [5] achieves precisely this effect: it modifies a submessage $k\ m$ (encrypted with an uncompromised key k) into the term $k\,(m, m')$ thereby cryptographically authenticating m' to any party able to access the ciphertext. By applying this refinement

twice (once for A and once for B), S can inform A and B of whom it created k for. This also allows us to discharge A as a parameter in B's role. Let KD_2^2 be the resulting protocol. Its expected run is given by the following diagram:



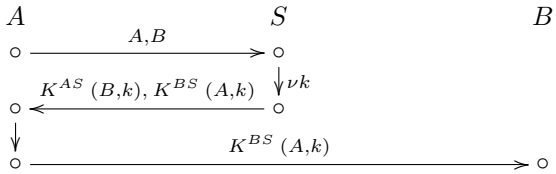
(We stop showing roles for space reasons.) A and B can derive the following properties, respectively:

$$A : \mathsf{uncomp}(K^{AS}, [A,S]) \ \wedge \ \mathsf{honest}\ S \wedge \langle A,B\rangle_A < (K^{AS}\ (B,k))_A$$
$$\Rightarrow \begin{bmatrix} \langle A,B\rangle_A \\ (A,B)_S < (\nu\, k)_S < \begin{bmatrix}\langle K^{AS}\ (B,k)\rangle_{S<} \\ \langle K^{BS}\ (A,k)\rangle_{S<}\end{bmatrix} \\ < (K^{AS}\ (B,k))_A \end{bmatrix}$$

$$B : \mathsf{uncomp}(K^{BS}, [B,S]) \ \wedge \ \mathsf{honest}\ S \ \wedge (K^{BS}\ (A,k))_B$$
$$\Rightarrow \ (A,B)_S < (\nu\, k)_S < \begin{bmatrix}\langle K^{AS}(B,k)\rangle_{S<} \\ \langle K^{BS}(A,k)\rangle_{S<}\end{bmatrix} < (K^{BS}(A,k))_B$$

While these formulas are similar to what we derived for protocol KD_2^1, A and B now know that the key k is intended for the two of them to communicate, not a third party (assuming, of course that S is honest and that the keys K^{AS} and K^{BS} are not compromised). This correction becomes crucially important when A and B attempt to use k.

While KD_2^2 achieves a minimal form of key distribution, few actual protocols have this message structure. Indeed, with the exception of recent group protocols [11], nearly all key distribution protocols based on shared keys have the server send both components $K^{AS}\ (B,k)$ and $K^{BS}\ (A,k)$ to one principal, who then forwards the part he does not understand to the other. This intuition is logically harnessed by means of the *relay transformation* which yields the following exchange structure:



In the corresponding protocol, which we will call KD_2^3, S concatenates $K^{AS}\ (B,k)$ and $K^{BS}\ (A,k)$, and sends the resulting message to A, who then forwards $K^{BS}\ (A,k)$ to B. Several protocols, e.g., Kerberos 5, follow this pattern. Note that the component $K^{BS}\ (A,k)$ is opaque to A, so her role mentions a generic message M.

Applying the relay transformation to the formula characterizing A's view in KD_2^2 yields the following expression:

$$A : \mathsf{uncomp}(K^{AS}, [A,S]) \ \wedge \ \mathsf{honest}\ S \wedge \langle A,B\rangle_A < (K^{AS}\ (B,k),\ M)_A < \langle M\rangle_A$$
$$\Rightarrow \begin{bmatrix} \langle A,B\rangle_A \\ (A,B)_S < (\nu\, k)_S < \langle K^{AS}\ (B,k), \boxed{K^{BS}\ (A,k)}\rangle_{\boxed{S<}} \end{bmatrix} \leq$$
$$\leq \langle K^{AS}\ (B,k), \boxed{M}\rangle_{\boxed{X<}} < (K^{AS}\ (B,k), \boxed{M})_A < \langle M\rangle_A$$

Compared to the analogous property of KD_2^2, A's receive action contains a generic M, and the server sends a concatenated message rather than the two components separately. This has two implications, highlighted in the boxes:

1. While, by the honesty assumption, A knows that S has sent $K^{AS}\ (B,k), K^{BS}\ (A,k)$, she cannot ascertain that the generic message M she receives is indeed $K^{BS}\ (A,k)$.

2. Since K^{AS} is uncompromised, A knows that S has originated $K^{AS}\ (B,k)$, but she cannot be sure of who originated the message $K^{AS}\ (B,k), M$ she received: hence the variable X for its originator, and the $\leq$ relation, a result of applying axiom rcv. Indeed an attacker could have replaced $K^{BS}(A,k)$ with an arbitrary message in an undetectable way. Such a behavior has been documented for Kerberos 5 [3].

Additionally, observe that A's last send has little bearing on the overall property and could be dropped.

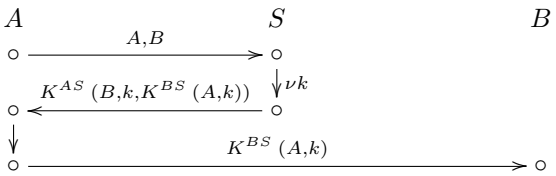
For similar reasons, B has no way to know who forwarded the message he receives.

$$B : \mathsf{uncomp}(K^{BS}, [B,S]) \ \wedge \ \mathsf{honest}\ S \ \wedge (K^{BS}\ (A,k))_B$$
$$\Rightarrow \ (A,B)_S < (\nu\, k)_S < \langle K^{AS}\ (B,k),\ K^{BS}\ (A,k)\rangle_{S<} <$$
$$< \langle K^{BS}\ (A,k)\rangle_{X<} < (K^{BS}\ (A,k))_B$$

Note that if B were able to infer that X is indeed A, he could also conclude that A knows the key k.

We conclude this section by deriving a variant of KD_2^3, in which B's component is embedded in A's rather than concatenated with it. Protocols that follow this approach include NSSK, Denning-Sacco and Kerberos 4.

This is formally achieved through a variant of the binding refinement used earlier. Applying it to KD_2^3 yields protocol KD_2^4, which has the following expected run:



A's resulting property enhances what she could deduce from KD_2^3 with the certainty that the opaque submessage M she receives is $K^{BS}\,(A,k)$:

$$\begin{array}{l} A : \mathsf{uncomp}(K^{AS},[A,S]) \;\wedge\; \mathsf{honest}\ S \;\wedge \\ \quad \langle A,B\rangle_A < (K^{AS}\,(B,k,M))_A < \langle M\rangle_A \\ \Rightarrow \left[\begin{array}{c} \langle A,B\rangle_A \\ (A,B)_S < (\nu\, k)_S < \langle K^{AS}\,(B,k,K^{BS}\,(A,k))\rangle_{S<} \end{array}\right] < \\ \quad < (K^{AS}\,(B,k,\boxed{K^{BS}\,(A,k)}))_A \;=\; (K^{AS}\,(B,k,\boxed{M}))_A \end{array}$$

$$=$$

At first sight, B's view does not significantly differ from what he could infer in KD_2^3:

$$\begin{array}{ll} B : \mathsf{uncomp}(K^{BS},[B,S]) \;\wedge\; \mathsf{honest}\ S \;\wedge & \\ \quad (K^{BS}\,(A,k))_B & \\ \Rightarrow \quad (A,B)_S < (\nu\, k)_S < \langle K^{AS}(B,k,K^{BS}(A,k))\rangle_{S<} < & \\ \quad < \langle K^{BS}\,(A,k)\rangle_{X<} < (K^{BS}\,(A,k))_B & \end{array}$$

Assuming S honest and K^{BS} uncompromised, B can deduce that S did its part in the protocol, and that some principal X forwarded $K^{BS}\ (A,k)$ to him. Under the additional assumption that K^{AS} is not compromised either, B can infer that it is A who forwarded this message to him. In particular, this tells B that A knows k.

$$\begin{array}{l} B : \mathsf{uncomp}(K^{BS},[B,S]) \;\wedge\; \mathsf{honest}\ S \;\wedge \\ \quad \mathsf{uncomp}(K^{AS},[A,S]) \;\wedge\; (K^{BS}\,(A,k))_B \\ \Rightarrow \; (A,B)_S < (\nu\, k)_S < \langle K^{AS}\,(B,k,K^{BS}\,(A,k))\rangle_{S<} < \\ \quad < \langle K^{BS}\,(A,k)\rangle_{A<} < (K^{BS}\,(A,k))_B \end{array}$$

Note that the assumption of $\mathsf{uncomp}(K^{AS},[A,S])$ would be irrelevant in any of B's previous inferences. Note also that the assumption that K^{AS} is uncompromised does not mean that A is bound to be honest: she could indeed deviate substantially from the protocol, passing information (but not K^{AS}) to arbitrary parties, but she certainly has decrypted S's message and certainly sent out $K^{BS}\ (A,k)$ (although not necessarily to B).

While most academic and industrial key distribution protocols based on shared keys are derived from either KD_2^3 or KD_2^4, these fragments lack two important guarantees: *recency* and *key confirmation*. Both KD_2^3 and KD_2^4 give the clients A and B assurance that the key k has been generated by the server for their exclusive communication needs, but they provide no verifiable guarantee that k was generated recently: an old k is more likely to have been compromised than one produced within a short time frame. None of the properties in this section binds the generation of k by any event controlled by the client receiving it. Key confirmation is about a client having some reason to believe that his counterpart has knowledge of k as well: only KD_2^4's B is able to gather this type of evidence (under assumptions). In the next sections, we will follow the development of two known families of protocols and observe how they address these issues.

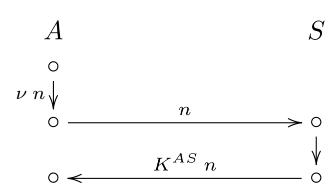
4 Derivations of NSSK

This section extends the results we just obtained in the direction of the Needham-Schroeder shared-key protocol (NSSK) [12]. In Section 4.1, we describe how a challenge-response exchange is used to guarantee the recency of the key, but also point out how a partial application of this technique leads to Denning and Sacco's classical attack on NSSK [6]. We then show how Needham and Schroeder's subsequent fix to the original NSSK [13] completes the application of nonce-based recency in Section 4.2. Finally, we address key confirmation in Section 4.3.

4.1 Guaranteeing Recency with Nonces

The core key distribution protocols in Section 3 do not guarantee to the clients that the server has generated the key recently. Indeed, none of the derived client formulas bounds the actions of an honest server: the key could have been produced at an arbitrary moment in the past, and now replayed. A client can assure key recency by bracketing its generation between two events whose occurrence it can guarantee. One approach to doing so is using the challenge-response mechanism: the client issues a challenge at the time she sends the key distribution request to the server. The server cryptographically binds the response to the challenge to the key distribution request. We dedicate this section to examining one concrete realization of this idea, adopted in NSSK and other protocols. A different approach, using time-stamps, will be examined in Section 5 when analyzing the Kerberos family.

We use the specific instance of the (cr) axiom shown in Section 2.2, which sends the challenge in the clear (the challenge function is the identity) and returns the response encrypted with an uncompromised shared key. Pictorially:



This challenge-response is composed with protocol KD_2^4 by means of the *merging transformation* exhaustively studied in [11]. It embeds these messages within the exchange between A and S in this protocol. The overall process is de-

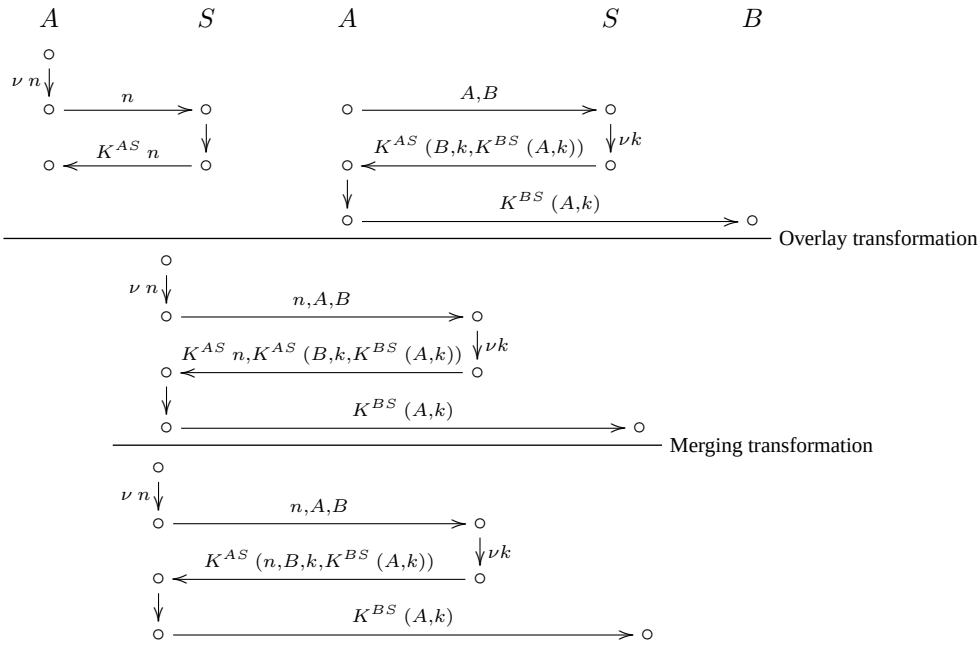
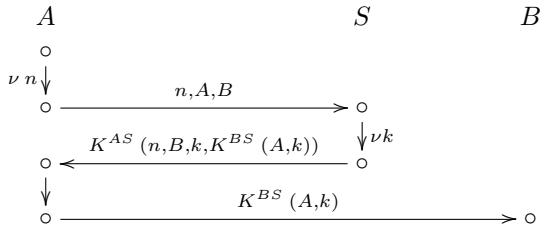


Figure 2. Derivation of NSSK_0

picted in Figure 2. The following diagram reports its effect:



The resulting protocol, NSSK_0, includes the first three steps of NSSK (the addition of key-confirmation will complete it in Section 4.3). B's role does not change at all from KD_2^4; the server's is modified to return the nonce n; most changes occur in A's role.

It is interesting to compare how the properties derivable to A and B change from what we obtained for KD_2^4. Because A created the nonce n fresh and it is returned cryptographically authenticated together with the key k, A can be certain that the server has generated k *after* her request. Thus, NSSK ensures the recency of the key to A.

$$
\begin{array}{l}
A : \mathsf{uncomp}(K^{AS}, [A,S]) \ \wedge\ \mathsf{honest}\ S \wedge \\
\quad (\nu\, n)_A < \langle n, A, B\rangle_A \qquad\quad < (K^{AS}(n,B,k,M))_A \\
\Rightarrow\ (\nu\, n)_A < \langle n, A, B\rangle_A < (n,A,B)_S < \\
\quad < (\nu\, k)_S < \langle K^{AS}(n,B,k,K^{BS}(A,k))\rangle_{S<} < \\
\quad < \qquad\qquad\qquad (K^{AS}(n,B,k,K^{BS}(A,k))_A
\end{array}
$$

The guarantees derivable to B are however much the same as in KD_2^4: B gets to deduce that some nonce n has been exchanged from S's honesty. However, no event controlled by B necessarily precedes the generation of k:

$$
\begin{array}{l}
B : \mathsf{uncomp}(K^{BS}, [B,S]) \ \wedge\ \mathsf{honest}\ S \wedge \\
\quad \mathsf{uncomp}(K^{AS}, [A,S]) \ \wedge\ (K^{BS}\ (A,k))_B \\
\Rightarrow\ (n,A,B)_S < (\nu\, k)_S < \langle K^{AS}(n,B,k,K^{BS}(A,k))\rangle_{S<} < \\
\quad < \langle K^{BS}(A,k)\rangle_{A<} < (K^{BS}(A,k) : X \rightarrow B)_B
\end{array}
$$

Therefore, NSSK does not ensures the recency of the key to B. This is the essence of Denning and Sacco's attack on NSSK [6].

4.2 NSSK-fix

A few years after Denning and Sacco pointed out the absence of recency guarantees for the responder [6], Needham and Schroeder came forth with a "fix" for their original protocol [13]. This adjustment inserts an additional challenge response between B and the server.

B's challenge differs from A's in order to avoid confusion. B generates a nonce n_B (for symmetry we rename A's nonce n_A), sends $K^{BS}(A, n_B)$ and expects it back as $K^{BS}\, n_B$. One can then verify that the properties of this exchange satisfy the challenge-response schema.

Some preliminary work is needed in order to compose NSSK_0 with this exchange. We first need to apply two instances of the relay transformation to the challenge-response in order to put it in the right "shape" for the merging transformation. Finally, we apply the discharging trans-

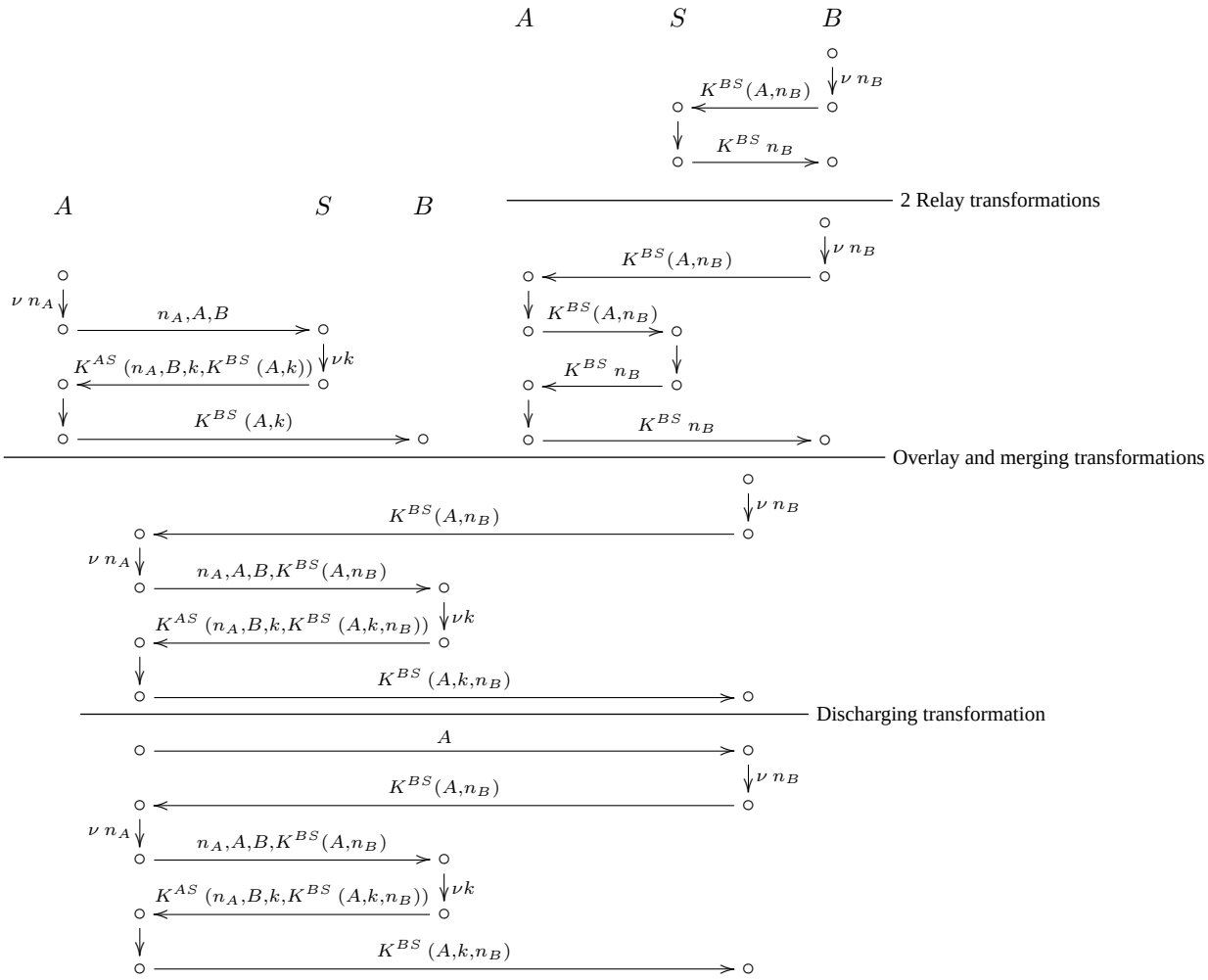
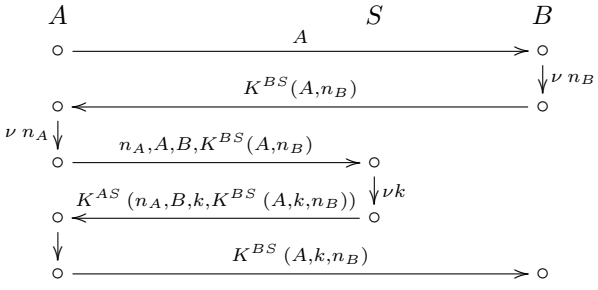


Figure 3. Derivation of $\mathsf{NSSKfix_0}$

formation to maintain A as the initiator of the resulting protocol. This is summarized in Figure 3. We call this protocol $\mathsf{NSSKfix_0}$. Its expected run is as follows:



This protocol differs from NSSK-fix only by the absence of the final key-confirmation steps. They will be added in Section 4.3. Like many other authors, we cannot avoid noting the complexity of this protocol, compared to NSSK or Denning-Sacco.

The lengthy formula characterizing the runs compatible with A's observations does not substantially change the properties available to this principal: if S is honest and K^{AS} is uncompromised, she can still deduce that S has generated k and that he has done so recently.

The interesting changes occur from B's perspective. As in A's case in $\mathsf{NSSK_0}$, B's nonce is cryptographically bound to the key k he receives by protocol's end. Since an honest server will construct this key only after retrieving this nonce from B's encrypted message, the generation of the key is sandwiched between two events under B's control, hence ensuring its recency. The rest of this property allows him to draw similar conclusions as in $\mathsf{NSSK_0}$, namely that S produced the key, forwarded it to A who learned it and forwarded it to B. This is summarized in the following

property.

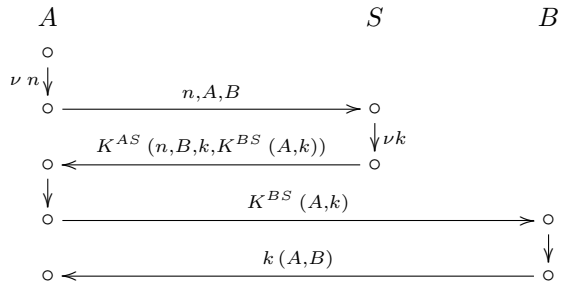
$$
\begin{array}{l}
B : \mathsf{uncomp}(K^{BS}, [B,S]) \ \wedge \ \mathsf{honest}\ S \ \wedge \\
\quad \mathsf{uncomp}(K^{AS}, [A,S]) \ \wedge \\
(A)_B < (\nu\, n_B)_B < \langle K^{BS}(A, n_B)\rangle_B < (K^{BS}(A,k,n_B))_B \\
\Rightarrow \ (A)_B < (\nu\, n_B)_B < \langle K^{BS}(A,n_B)\rangle_B < \\
\quad < (n_A, A, B, K^{BS}\,(A,n_B))_S < (\nu\, k)_S < \\
\quad < \langle K^{AS}\,(n_A, B, k, K^{BS}\,(A,k,n_B))\rangle_{S<} < \\
\quad < \langle K^{BS}\,(A,k,n_B)\rangle_{A<} < (K^{BS}\,(A,k,n_B))_B
\end{array}
$$

As in NSSK_0, dropping the assumption that K^{AS} is uncompromised implies that B does not know who has originated the message $K^{BS}\,(A,k,n_B)$ and that he cannot be certain that A knows k.

4.3 Key Confirmation

The previous two sections have shown how to extend the core key distribution protocol KD_2^4 with the recency guarantees of NSSK(-fix). The remaining issue, addressed in this section, is ensuring to both recipients that their counterpart also knows the new shared key. So far, only B has this guarantee.

The simplest way to achieve this is by having B send A a pre-agreed message m (e.g., (A,B)) encrypted with k. Post-composing NSSK_0 with this transmission yields the protocol NSSK_1, which has the following expected run:



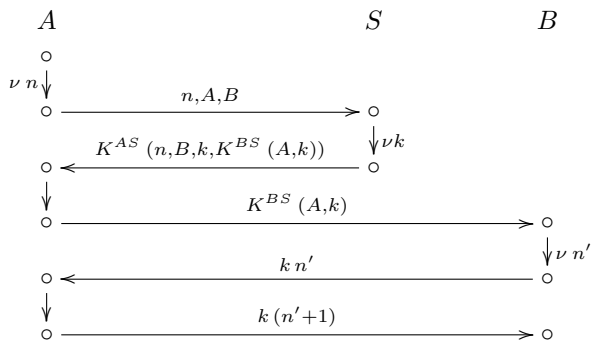
A's observations lead her to conclude:

$$
\begin{array}{l}
A : \mathsf{uncomp}(K^{AS}, [A,S]) \ \wedge \ \mathsf{honest}\ S \ \wedge \\
\quad \boxed{\mathsf{uncomp}(k, [A,B,S])} \ \wedge \ (\nu\, n)_A < \langle n, A, B\rangle_A < \\
\quad < (K^{AS}(n,B,k,M))_A < \boxed{\langle M\rangle_A < (k(A,B))_A} \\
\Rightarrow \ (\nu\, n)_A < \langle n, A, B\rangle_A \ < (n, A, B)_S < (\nu\, k)_S < \\
\quad < \ \langle K^{AS}(n, B, k, K^{BS}\,(A,k))\rangle_{S<} < \\
\quad < \ (K^{AS}(n, B, k, K^{BS}(A,k))_A < \\
\quad < \ \boxed{\begin{array}{l} \langle K^{BS}(A,k)\rangle_A < (K^{BS}(A,k))_B < \\ < \langle k(A,B)\rangle_{B<} < (k(A,B))_A \end{array}}
\end{array}
$$

We have highlighted the additions with respect to NSSK_0 (see Section 4.1). We had omitted the then trailing $\langle M\rangle_A$ and $\langle K^{BS}\,(A,k)\rangle_A$ since they did not add substantial information. Now they clearly do, as they allow A to infer that B has received this message and originated $k\,(A,B)$.

The last addition, $\mathsf{uncomp}(k, [A,B,S])$, deserves some discussion. Clearly, we need to know that k is uncompromised to infer anything useful involving it. However, most formal systems would *derive* this fact rather than *assume* it. This may be where the strict separation between authentication and secrecy is most evident in this work. Recall that our logical system is just powerful enough to reason about the order of actions, the structure underlying authentication. In particular it does not embed the logical principles to derive that k must indeed be secret. The assumption $\mathsf{uncomp}(k, [A,B,S])$ is an interface to a secrecy logic.

Applying the above extension to $\mathsf{NSSKfix}_0$ yields $\mathsf{NSSKfix}_1$. This protocol has then the typical properties of a key distribution protocol: both clients receive assurance that the key has been generated by the expected server, that this key is controllably recent, and that they both know the key. However, the actual NSSK-fix is different: B encrypts a new nonce with k and sends it to A, and expects this same nonce back from A, transformed in a predictable way. We will now discuss what additional properties are achieved by doing so. For the sake of succinctness, we operate on NSSK_1, which differs from the original NSSK in precisely the same way as $\mathsf{NSSKfix}_1$ is different from NSSK-fix. Here is the expected run of NSSK:



First, notice that having A send something encrypted with k back to B does not produce any new knowledge (besides the obvious, i.e., that a new message has been transmitted). From the point of view of B, the last two messages of NSSK implement a challenge-response exchange: B generates the nonce n', sends it to A encrypted (with k), and expects it back from her transformed. B thus ascertains that A in alive at this particular point of the protocol. Note that B could repeat this same exchange an arbitrary number of times (each with a new nonce) and obtain the same guarantee: that A was recently alive. De facto, this implements a crude single-authentication, repeated-request client-server mechanism, with the initiator acting as the server and the responder as the client.

In summary, our analysis shows that NSSK-fix achieves key distribution with recency guarantees and key confirmation for both parties. NSSK provides recency assurance

only to the initiator. Our work also shows that the same guarantees are also supported by simpler protocols that drop the last message and rely on any pre-arranged message instead of the final nonce.

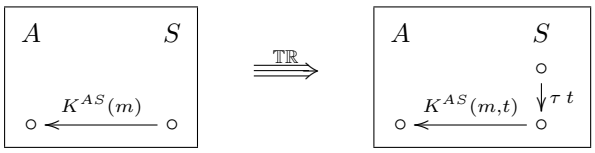
5 Derivations of Kerberos

Kerberos [14, 15] is a complex and versatile protocol that has been the subject of intense scrutiny over the years. In this section, we will apply the methods outlined above to derive the core authentication functionalities of versions 4 and 5 of this protocol. We concentrate on the basic key distribution exchange of which each version contains two instances. As a preparatory step, we formalize the use of timestamps for authentication and derive the Denning-Sacco protocol, a core component of Kerberos 4.

5.1 Guaranteeing Recency with Timestamps

Timestamps have a number of applications in cryptographic protocols. In this section, we examine and formalize their use for the purpose of guaranteeing the recency of an already authenticated message. Consider a principal A receiving a message $K^{AS}\ m$ from an honest agent S: if the key is uncompromised, A can only deduce that S originated this message in the (possibly distant) past; if however S includes a timestamp t within the encryption and sends $K^{AS}(m,t)$, A can assess the age of the message and reject it if it falls outside of her window of validity. This assessment takes into considerations clock skews between hosts, typical network delays, etc.

We formalize this intuition as a new variant of the binding refinement [5] used in Section 3 in which the bound term is a timestamp. We call it the *timestamping refinement* and denote it $\mathbb{TR}$. It transform the exchange on the left below into the exchange on the right:



This refinement allows upgrading the logical guarantees that each principal can deduce. Given the particular format of this transformation (S does not receive a message back), we concentrate on the formulas derivable by A. Schematically:

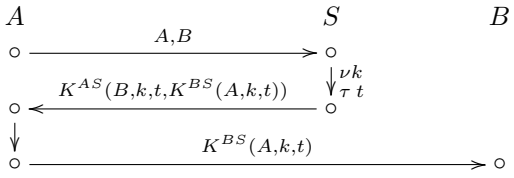
$$
\begin{array}{ll}
A: & \mathsf{uncomp}(K^{AS},[A,S]) \wedge (\!(K^{AS}\ m)\!)_A \\
\Rightarrow & \langle\!\langle K^{AS}\ m \rangle\!\rangle_{S<} < (\!(K^{AS}\ m)\!)_A \\
 & \qquad\qquad \Big\Downarrow \mathbb{TR} \\
\multicolumn{2}{l}{A: \mathsf{uncomp}(K^{AS},[A,S]) \wedge \mathsf{honest}\ S \wedge (\!(K^{AS}(m,t))\!)_A} \\
\multicolumn{2}{l}{\Rightarrow (\underline{\tau}\ t)_A < (\tau\ t)_S < \langle\!\langle K^{AS}(m,t) \rangle\!\rangle_{S<} < (\!(K^{AS}(m,t))\!)_A}
\end{array}
$$

The top formula describes how A can extend her knowledge after receiving $K^{AS}\ m$ whenever the original protocol guarantees the authenticity of m: note that, as long as K^{AS} is not compromised, S is not required to be honest. The bottom lines show the upgraded formula. Recall that $(\underline{\tau}\ t)$ represents the earliest point in A's local time where she will accept the time t as valid. It is now important that S is believed to be honest: without this, S could guess an appropriate value for t rather than looking it up on its clock.

We obtain this formula by homomorphically replacing $K^{AS}\ m$ with $K^{AS}(m,t)$ in the derivation of the top formula. The atom $(\tau\ t)_S$ comes from the upgraded honesty axiom and is justified by axiom (ts).

5.2 The Denning-Sacco Protocol

The Denning-Sacco protocol [6] applies the timestamping refinement to the basic key distribution protocol with nested encryption KD_2^4 where the authenticated message (m above) is (k,X), where k is the newly generated key and X is either A or B. S applies this refinement twice, adding the same timestamp next to each key distribution submessage. As a consequence, by the completion of the protocol, each principal has the certainty that S has generated k recently. As in KD_2^4, because of the nested encryption, B additionally knows that A has seen k (but A cannot be certain that B ever receives k). We have shown the resulting protocol in Section 2. Its expected run is as follows:



From the sole observation of her actions and the honesty of the server, A can reconstruct the whole protocol, save for B's reception of her last message:

$$
\begin{array}{l}
A: \mathsf{uncomp}(K^{AS},[A,S])\ \wedge\ \mathsf{honest}\ S \wedge \\
\quad \langle A,B\rangle_A \qquad\qquad\qquad\qquad\qquad\qquad < (K^{AS}(B,k,t,M))_A \\
\Rightarrow \begin{bmatrix} \langle A,B\rangle_A < (A,B)_S \\ (\underline{\tau}\ t)_A \end{bmatrix} < \begin{bmatrix} (\nu\ k)_S \\ (\tau\ t)_S \end{bmatrix} < \\
\quad < \langle K^{AS}(B,k,t,K^{BS}(A,k,t))\rangle_{S<} \\
\qquad\qquad\qquad\qquad < (K^{AS}(B,k,t,K^{BS}(A,k,t)))_A
\end{array}
$$

We have elided A's final send action as it does not contribute added knowledge. Note that S's generation of k is now bounded by $\underline{\tau}\, t$, which is under the control of A.

B's conclusions merge the recency assurance provided by timestamps with what he could infer by means of KD_2^4, i.e., that S has generated k and that A has seen it in order to forward the message he receives:

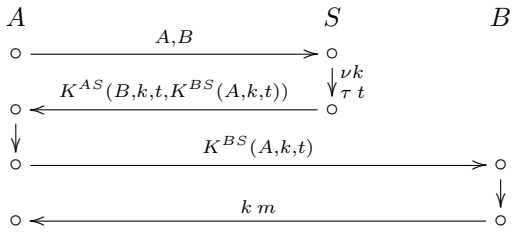
$$\begin{aligned} B : \mathsf{honest}\ S\ \wedge\ \mathsf{uncomp}(K^{BS}, [B,S]) \wedge \\ \wedge\ \mathsf{uncomp}(K^{AS}, [A,S])\ \wedge\ (K^{BS}(A,k,t))_B \\ \Rightarrow\ \begin{bmatrix} (A,B)_S \\ (\underline{\tau}\, t)_B \end{bmatrix} < \begin{bmatrix} (\nu\, k)_S \\ (\tau\, t)_S \end{bmatrix} < \\ < \langle K^{AS}(B,k,t,K^{BS}(A,k,t))\rangle_{S<} < \\ < \langle K^{BS}(A,k,t)\rangle_{A<} < (K^{BS}(A,k,t))_B \end{aligned}$$

Denning and Sacco prominently pointed out in their original paper [6] that this protocol provides full recency guarantees with a minimum number of messages.

5.3 Kerberos 4

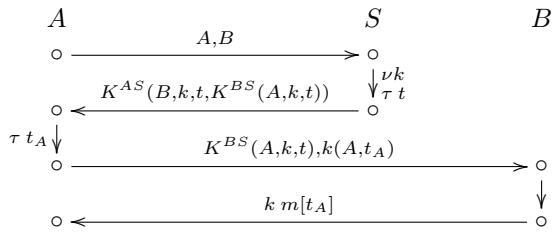
The core authentication functionalities of Kerberos 4 [14] are obtained by simply extending the Denning-Sacco protocol by means of a key confirmation exchange similar to the way we obtained NSSK(-fix) in Section 4.3.

Adding key confirmation We extend DS by having B send A some (recognizable) message m encrypted with k. The resulting run is as follows:



The formula characterizing the runs compatible with each principal's observations is extended as in Section 4.3 [4].

Adding repeated authentication Kerberos was designed as a *repeated* authentication protocol: each time A presents the *ticket* $K^{BS}(A,k,t)$, B will provide some predetermined service (up to an end-date that we can abstractly think of as a function of t). The protocol we just derived is clearly inadequate for this purpose as anybody can replay the ticket $K^{BS}(A,k,t)$. B needs to authenticate that a subsequent request comes from A, and assess that it was made recently enough. Kerberos 4 realizes these two goals by having A generate a timestamp t_A just prior to issuing a new request, and embedding into it an *authenticator* $k\,(A,t_A)$ (any message mentioning t_A and encrypted with k would do). The intended run of the resulting protocol is as follows:



where the last message is made dependent on t_A (although Kerberos does not always enforce this).

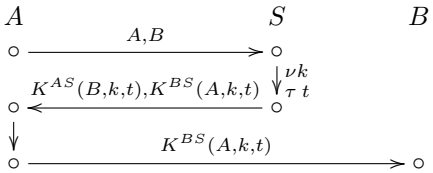
Observe that, differently from NSSK(-fix), it is the initiator of the protocol (the client, A) that requests the service provided by the responder (B). Indeed, A generates the timestamp t_A that is included in the authenticator.

Kerberos 4 [14] extends this core protocol with numerous fields primarily meant to negotiate parameters of the resulting authentication: added timestamps, options and flags, access control information, etc. For maximum flexibility, Kerberos chains two instances of the core protocol, by which a client (A) first obtains a master ticket (TGT) which simplifies the issuance of tickets for individual services.

5.4 Kerberos 5

As far as authentication is concerned, Kerberos 5, the most recent version of this protocol [14, 15], differs from Kerberos 4 only by the form of the basic key distribution mechanism it relies on: while version 4 was built up from the nested variant KD_2^4 , Kerberos 5 starts with the concatenated variant KD_2^3. Given this different starting point, the core protocol is however derived by applying the exact same steps as for Kerberos 4. It is interesting to examine them as the conclusions available to the various principals are not the same throughout.

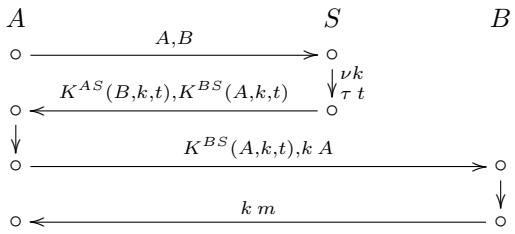
The concatenated variant of the Denning-Sacco protocol has the following expected run:



The knowledge derivable by A is similar to the Denning-Sacco protocol, except that she can never be certain that the encrypted component she receives corresponds to what S has sent. More interesting is the knowledge inferable by B: differently from the Denning-Sacco protocol, B cannot reach any conclusion on whether A ever saw the key k:

indeed, the assumption $\mathsf{uncomp}(K^{AS}, [A, S])$ becomes irrelevant. B knows that the server sent the appropriate messages and that some principal X forwarded the correct component to him. This makes B's knowledge very similar to A's.

With both A and B unaware of whether its counterpart has seen k, each party needs to inform the other of its knowledge of k. We rely on the device already used in Kerberos 4 to accomplish this: A will concatenate the component $(k\, A)$ as she forwards $K^{BS}(A, k, t)$ to B. As in version 4, B will confirm k with a response $k\, m$ for some recognizable m. We obtain the following exchange:



This protocol fragment is extended to allow repeated authentication using k exactly as for Kerberos 4: A generates a timestamp t_A and includes it in her authenticator; B optionally returns t_A in the last message.

This is the authentication core of Kerberos 5. As in its predecessor, two instances of this fragment are chained together, and numerous fields add a great deal of flexibility [14, 15]. It should be noted that, in Kerberos 5, the timestamp-based recency assessment (using t) is supplemented with a nonce-based guarantee by which A sends S a nonce n with her initial request and expects it back within $K^{AS}(B, k, t)$.

6 Conclusions

We have proposed a simple logic of partial order to analyze the authentication properties of security protocols. It encapsulates necessary secrecy guarantees as assumptions to be proved in a different formalism. This clean separation was motivated by the observation that secrecy and authentication properties rely on very different proof methods, often intertwined in complex analyses. We use this logic to drive the derivational approach in organizing a large class of authentication protocols into a taxonomy cataloged by the authentication properties of their components and how they are combined.

In a natural continuation of this work, we are developing a logic of pure secrecy that encapsulates authentication requirements as assumptions. The authentication logic is being incorporated into the Protocol Derivation Assistant [1]. We are also currently investigating the automation of the process by which a principal's observation is completed into the set of compatible runs. Efforts in this direction touch the question of the decidability of verifying authentication (with encapsulated secrecy assumptions).

References

[1] M. Anlauff and D. Pavlovic. The Protocol Derivation Assistant (version 1.8.20). Available electronically at `http://www.kestrel.edu/software/pda`, Dec. 2004.

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

[3] F. Butler, I. Cervesato, A. D. Jaggard, and A. Scedrov. A Formal Analysis of Some Properties of Kerberos 5 using MSR. In *Proc. CSFW-02*, pages 175–190. IEEE Computer Society Press, 2002.

[4] I. Cervesato, C. Meadows, and D. Pavlovic. Deriving key distribution protocols and their security properties. Available upon request to the program chair, 2005.

[5] A. Datta, A. Derek, J. Mitchell, and D. Pavlovic. A derivation system and compositional logic for security protocols. *Journal of Computer Security*, 2004. To appear.

[6] D. E. Denning and G. M. Sacco. Timestamps in key distribution protocols. *Communications of the ACM*, 24(8):533–536, 1981.

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

[8] N. Durgin, J. Mitchell, and D. Pavlovic. A compositional logic for proving security properties of protocols. *Journal of Computer Security*, 11(4):667–721, 2003.

[9] G. Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. *Software - Concepts and Tools*, 17:93–102, 1996.

[10] G. Lowe. A herarchy of authentication specifications. In *Proc. 10th IEEE Computer Security Foundations Workshop*, pages 31–43. IEEE Computer Society Press, 1997.

[11] C. Meadows and D. Pavlovic. Deriving, attacking and defending the GDOI protocol. In *Proc. ESORICS 2004*, pages 33–53. Springer-Verlag LNCS 3193, 2004.

[12] R. M. Needham and M. D. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12):993–999, 1978.

[13] R. M. Needham and M. D. Schroeder. Authentication revisited. *ACM Operating Systems Review*, 21(1):7, 1987.

[14] B. C. Neuman and T. Ts'o. Kerberos: An Authentication Service for Computer Networks. *IEEE Communications*, 32(9):33–38, 1994.

[15] C. Neuman, J. Kohl, T. Ts'o, T. Yu, S. Hartman, and K. Raeburn. The Kerberos Network Authentication Service (V5), September 7 2004. Internet draft, expires 7 March 2005.