DERIVING KEY DISTRIBUTION PROTOCOLS AND THEIR SECURITY PROPERTIES

Iliano Cervesato,[1] Catherine Meadows,[2] Dusko Pavlovic[3]*

[1] *Department of Mathematics, Tulane University, New Orleans, LA 70118*
iliano@math.tulane.edu

[2] *Code 5543, Naval Research Laboratory, Washington, DC 20375*
meadows@itd.nrl.navy.mil

[3] *Kestrel Institute, 3260 Hillview Avenue, Palo Alto, CA 94304*
dusko@kestrel.edu

Abstract We apply the derivational method of protocol verification to key distribution protocols. This method assembles the security properties of a protocol by composing the guarantees offered by embedded fragments and patterns. It has shed light on fundamental notions such as challenge-response and fed a growing taxonomy of protocols. Here, we similarly capture the essence of key distribution, authentication timestamps and key confirmation. With these building blocks, we derive the authentication properties of the Needham-Schroeder shared-key and the Denning-Sacco protocols, and of the cores of Kerberos 4 and 5.

Keywords: Cryptographic protocols, key distribution, verification by construction, protocol composition, logical inference.

1. Introduction

Key distribution is one of the most studied themes in security. The problem and the basic ideas for the solutions were first described in Needham and Schroeder's seminal 1978 paper [12]. On a network of computers, the users and processes often need to access remote resources. In order to prevent unauthorized use, these accesses need to be authenticated, and often protected by encryption. Key distribution protocols cater to this need by providing participating entities with a fresh shared key for direct and secure communication.

The most popular key distribution protocol is Kerberos. It was designed at MIT, originally just to protect the network services provided by Project Athena, an initiative developed in the eighties to integrate computers in the

*Supported by ONR N00014-03-C-0237 and by NSF CCR-0345397.

MIT curricula [17]. Distributed for free, it subsequently achieved a widespread use beyond MIT. While its earlier versions were not always suitable for large scale applications, versions 4 and now 5 have been redesigned for large systems [14, 15].

In the present paper, we present a formal reconstruction of the developments leading up to the Kerberos protocols. The starting point can be found in the original Needham-Schroeder Shared Key (NSSK) protocol, proposed in [12], which motivated the very idea of the Authentication Server. Along the way, the Denning-Sacco attack and protocol [4] championed timestamp-based security.[1] At their core, Kerberos 4 and 5 combine and extend these ideas into an industrial-strength single-logon authentication infrastructures [14, 15].

We recast these conceptual steps in the formal framework of the protocol derivation system, that evolved through [3, 10]. Such logical reconstructions of development histories allow classifying protocols according to the underlying security components and concepts. The resulting taxonomies then provide a foundation for a practical framework for secure reasoning, where the results of previously achieved protocol development efforts are available for reuse, refinement and composition. One such framework is being implemented as a tool, the Protocol Derivation Assistant, with all such reconstructions available as reusable libraries. In previous work, we have looked at electronic commerce protocols [3] and group protocols [10]. The protocol taxonomy obtained for them summarized recurrent security practices and supported recombining them to derive further protocols.

Presently, we only derive the basic components of the Kerberos protocols and their authenticity properties. The actual deployed protocols chains several (at least two) rounds of such components, bound together by secret data. The issues leading up to this composition, and arising from it, will be studied in a sequel paper.

This work is organized as follows: in Section 2 we explain the protocol derivation infrastructure. We use it to express the basic key distribution mechanism in Section 4. We extend in the direction of NSSK in Section 4 with nonce-based recency and key confirmation. We extend 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. Protocol Composition System

We outline the methodology underlying our analysis in Section 2.1 and formalizes the resulting framework, that we call the *Protocol Composition System*,

[1] Needham and Schroeder proposed an alternative fix to NSSK that does not rely on timestamps in [13].

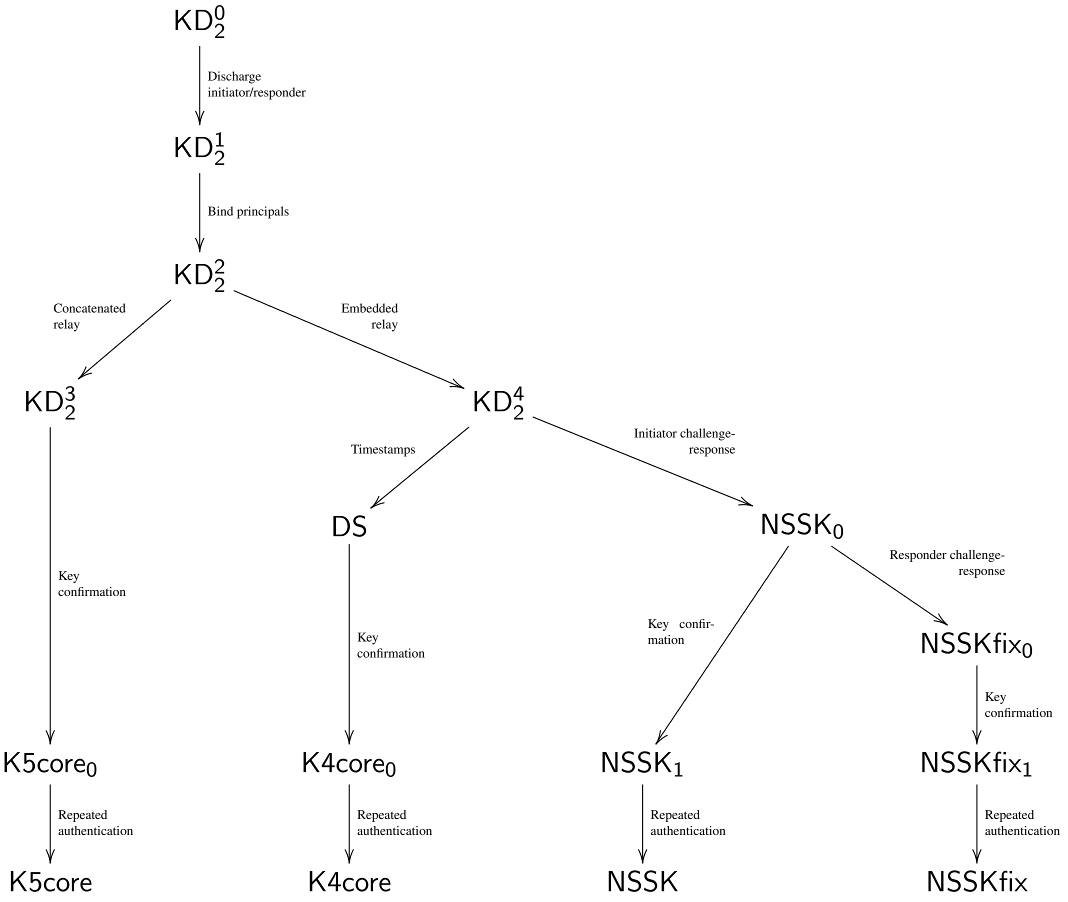


Figure 1. Overview of the Derivation of Key-Distribution Protocols

in Sections 2.2–2.5. It should be noted that, while the Protocol Composition System is clearly inspired by our previous work [3, 10], a number of notions, such as **???**[2], are novel. Therefore, the following material provides more than a review of established concepts.

[2] **EdNote:** spell out.

2.1 Overview

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 $\mathcal{R}_A$ 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 $\mathcal{R}_{\mathsf{Prop}}$ of legal runs for P, the verification task traditionally reduces to showing that $\mathcal{R}_A$ is contained in $\mathcal{R}_{\mathsf{Prop}}$, and similarly for the other parties in the protocol. Every run in $\mathcal{R}_A$ but not in $\mathcal{R}_{\mathsf{Prop}}$ is an attack on A with respect to Prop.

We take a different approach: rather than comparing $\mathcal{R}_A$ with the legal runs of a given authentication property, we synthesize a logical expression Φ_A describing $\mathcal{R}_A$. 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_A, that in a strong sense describes *the* properties of $\mathcal{R}_A$. Indeed, this constructions provides us with a clear view of the properties contributed by each component and whether they propagate to Φ_A. 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. Note that these assumptions are elective.

Rather than checking that a protocols 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 [10]. We can also scrutinize the formula Φ_A summarizing the possible runs of each principal A in the light of a well-known authentication property, such as matching histories [5] or agreement [9]. We do not, however, formalize traditional properties Prop as formulas in our logic for formal comparison with the deduced formulas Φ_A (this will be for another paper).

A crucial aspect of our approach is that component-formulas pairs can be reused whenever they occur in another protocols. Even more interesting is the fact that the composition operations for fragments and properties can be made systematic, which gives rise to protocol taxonomies [3]: a rational classification of protocols that not only aides our understanding of these complex objects, but also helps choosing or devising a protocol based on desired features and properties. We are working on a tool that will assist us building taxonomies that are much larger than what we have so far been able to construct by hand.[3]

Below, we give the necessary definitions to formalize the notions leading to the set of possible runs $\mathcal{R}_A$ deducible by a principal A and the corresponding formula Φ_A. We define the basic vocabulary of terms, actions and protocol specifications in Section 2.2. We introduce dynamic concepts such as runs and observations in Section 2.3. Section 2.4 sets the stage for the logical expression of the set of possible runs deducible from an observation, while Section 2.5 provides the logical means to perform such deductions. The remainder of this paper will apply these definitions in the study of key distribution protocols, starting from basic concepts all the way to Kerberos.

2.2 Syntactic Categories

In this section, we present the formal syntax used in the Protocol Composition System. In particular, we define principals, terms, patterns, actions, roles and protocols. These notions will be used in the sequel to define dynamic notions such as runs and local observations, and deductive reasoning will operate on them.

Principals. We model *principals* as a partially ordered set, or poset, $(\mathcal{W}, \Subset)$, where $\mathcal{W}$ enumerates the principals we are working with and the *subprincipal* relation $\Subset$ is a reflexive partial order on them. The subprincipal relation can represent, as needed, access to information or resources, or subsume e.g. the relations "speaks for" [1, 8], or "acts for" [11], or model groups and coalitions of principals. We will make limited use of this relation in this paper. We denote the class of variables ranging over principals with $\mathsf{Var}_{\mathcal{W}}$. We write $A, B, S, \ldots$ for generic principals, and use capital letters towards the end of the alphabet for elements of $\mathsf{Var}_{\mathcal{W}}$.

Terms. The set $\mathcal{T}$ of *terms* is an abstract term algebra constructed over a set of variables $\mathsf{Var}_{\mathcal{T}}$ and a set of operators $\mathsf{Op}_{\mathcal{T}}$ (some of which may be constants). Principals are a subclass of terms, i.e. $\mathcal{W} \hookrightarrow \mathcal{T}$ (and similarly for principal variables). We also assume the standard classes used in modeling cryptographic protocols: $\mathsf{Nnc}, \mathsf{Key}, \mathsf{Time} \ldots \hookrightarrow \mathcal{T}$ for nonces, keys, timestamps, and so on. We write m for a generic message, but use k, n, t, etc, for keys, nonces, timestamps and other specialized messages. The letters x, y, z, ...will denote variables. In this paper, we will rely on two specific construc-

[3] **EdNote**: insert a URL for the tool?

tors:

$$\begin{array}{ll} \_\,\_ : \mathsf{Key} \times \mathcal{T} \to \mathcal{T} & (k\ m \text{ is the encryption of } m \text{ with } k) \\ \_,\_ : \mathcal{T} \times \mathcal{T} \to \mathcal{T} & (m, m' \text{ is the concatenation of } m \text{ and } m') \end{array}$$

but $\mathcal{T}$ may contain more. The standard *subterm* relation $\sqsubset$ endows terms with the structure of a poset $(\mathcal{T}, \sqsubset)$.

Patterns. A *pattern* is a term p together with a list of distinguished variables $\vec{x}$ occurring exactly once in p that will be interpreted as *binders* — p may contain other variables. We mnemonically write this pattern as $p(\vec{x})$ but will often keep $\vec{x}$ implicit when clear from the context. The set $\mathcal{P}_\mathcal{T}$ of patterns on $\mathcal{T}$ is therefore defined as

$$\mathcal{P}_\mathcal{T} = \bigcup_{n\in\mathbb{N}} (\mathcal{T} \times \mathsf{Var}_\mathcal{T}^n)$$

We further restrict the class of admissible patterns to account for non-invertible cryptographic operations: for example, we reject patterns of the form “$x\ m$” which would allow extracting the key used to encrypt the term m.

Actions. Principals participate in a protocol by performing atomic *actions*. The set Σ of actions is generated from the set of terms $\mathcal{T}$ and the set of principals $\mathcal{W}$ by the following constructors:

Action	Constructor	Form	Informal meaning
send	$\mathcal{T} \times \mathcal{W}^2 \xhookrightarrow{\langle\rangle} \Sigma$	$\langle m : A \to B\rangle$	The term m is sent, purportedly from A to B
receive	$\mathsf{Var}_\mathcal{T} \times \mathsf{Var}_\mathcal{W}^2 \xhookrightarrow{()} \Sigma$	$(x : Y \to Z)$	A term, source and destination are received into the variables x, Y, and Z
match	$\mathcal{T} \times \mathcal{P}_\mathcal{T} \xhookrightarrow{/} \Sigma$	$(m/p(\vec{x}))$	The term m is matched with the pattern $p(\vec{x})$, binding $\vec{x}$
new	$\mathsf{Var}_\mathcal{T} \xhookrightarrow{\nu} \Sigma$	$(\nu\ x)$	A fresh value is created and stored in the variable x
now	$\mathsf{Var}_\mathcal{T} \xhookrightarrow{\tau} \Sigma$	$(\tau\ x)$	The system time is read and stored in the variable x

These actions will take the center stage in this paper. We will occasionally introduce internal actions to model protocol specific operations (e.g., looking up an internal table). Other actions can be added as needed. The variables x, Y, Z in **receive**, $\vec{x}$ in **match**, and x in **new** and **now** are binding occurrences, so that any subsequent mention in an expression involving actions (e.g., roles below) are interpreted as bound by them. We adopt the standard definitions of free and bound variables in an action. We will often use partial descriptions of actions, and elide e.g., the source and the destination, as in $\langle m\rangle$, or (y), or other parts, as in $\langle A \to C\rangle$, or $(x : A \to)$.

We will formalize the meaning of these actions in the next section, where we present the execution model of the Protocol Composition System.

Roles. 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 set $\mathcal{R}$ of roles is then defined as $\mathcal{R} = \mathcal{W} \times \Sigma^{(;\otimes)}$, where the second component is the algebra stemming from Σ and operations ";" and "$\otimes$". We tacitly use ";" as an associative operator, while "$\otimes$" will be viewed as associative and commutative.

Sequential composition ";" orders the actions in a role, while "$\otimes$" specifies clusters that can be executed in parallel. A binder occurring in an action has scope over the actions in all paths stemming from it. Care should be taken so that no variable is in the scope of more than one homonymous binder when disambiguation is not possible: we avoid this problem completely by requiring that every binder uses a different variable name. The free variables of a role are its parameters, and should be instantiated prior to executing the role. The principal executing the role is a distinguished parameter.

As an example, we show the server role of the Denning-Sacco protocol, further explored in Section 5:

$$
\begin{array}{rcl}
\mathit{DS\_Server}[S] & = & (m_0 : A \rightarrow S_0); (S_0/S); (m_0/A, B); \\
 & & (\mathrm{getKey}(A, K^{AS}) \otimes \mathrm{getKey}(B, K^{BS})); \\
 & & (\nu\, k \otimes \tau\, t); \\
 & & \langle K^{AS}(B, k, t, K^{BS}(k, A, t)) : S \rightarrow A \rangle
\end{array}
$$

This role has one parameter, the name of the server S executing it. With the actions on the first line, S receives a message m_0, purportedly from some principal A (this is the binding occurrence for this variable), he verifies that he was indeed the intended recipient, and that m_0 is a pair with A as its first component (this occurrence is in the scope of the A in the receive action) and some name B as its second component. The second line invokes some internal actions to retrieve the keys S shares with A and B, and bind them to the variables K^{AS} and K^{BS} respectively. Notice that this specification allows the concurrent execution of these two actions. On the third line, S generates a key, binding it to k, and looks up the current time into t, again concurrently. The last action sends the shown message.

Protocols. A *protocol* is a collection of roles that covers the actions of all parties involved in the protocol. In the case of Denning-Sacco, the protocol consists of three roles: the above server role, an initiator, and a responder.

2.3 Execution Model

This section defines the dynamic concepts of runs and local observations. We start with the preliminary notions of processes (a minimally connected collection of actions), then associate every receive action with a send action in the notion of run, then target the proper instantiation of variables by defining execution, and finally distill the local observation of principal from an executable run.

Events. An *event* associates an action to a principal, that we will understand has having executed this action. We denote an event by subscripting the action with the principal in question, writing for example $\langle m : A \rightarrow B \rangle_A$ for the event of principal A performing the action $\langle m : A \rightarrow B \rangle$.

Processes. A *process* is a partially ordered multiset (pomset) of events, i.e., actions attributed to principals. More precisely, given a set of action labels $\mathbb{L}$, a process L is an assignment

$$\mathbb{L} \overset{L}{\Longrightarrow} \Sigma \times \mathcal{W}$$

such that

- $(\mathbb{L}, <)$ is a well-founded partial order.
- $\ell < \ell'$ implies $L_{\mathcal{W}}(\ell) \Subset L_{\mathcal{W}}(\ell')$ or $L_{\mathcal{W}}(\ell) \Supset L_{\mathcal{W}}(\ell')$.

where $L_{\mathcal{W}}(\ell)$ is the name of the principal in event $L(\ell)$.

The relation $<$ orders the events in a process so that one event can be described as occurring before another in an abstraction of the temporal dimension of execution. Events that are not related by $<$ can have occurred in arbitrary order. The first condition simply prevents cycles.

The second condition specifies that only actions pertaining to the same principal, or one of its sub-/super-principals, can be ordered in this way (we will extend this ordering across principal cliques shortly). Like strands [7], related events in a process pertain to a single principal (or group of related principals). Unlike strands, processes are not bound to a single protocol execution, but may order events executed by a principal in several instances of the same protocol, possibly in several roles, and even while executing several different protocols. The idea is that a principal will know in what order she has executed actions, even when several protocols are involved. However, notice that the events of a principal do not need to be totally ordered: events can be unrelated if their exact order does not matter, or if the underlying execution model is actually parallel. Finally, processes may contain variables while strands are always ground.

In the sequel, processes and related notions will generally arise out of instantiated protocol roles. This will always be the case when reasoning about the observations of a principal, or when assuming that a principal is honest. However, the actions attributed to a principal that is not assumed to be honest may live outside of any role.

Before moving on, a few notational conventions will prove enormously helpful. We will often abuse notation and denote a label $\ell \in \mathbb{L}$ in a process L by the event $L(\ell)$ it points to. Furthermore, we will often speak, for example, of "the event $\langle m : A \to B\rangle_A$" in the context of a process L, although there may, of course, be several events of this form in L. We will resurrect labels only in case of ambiguity. We will also sometimes blur the distinction between an action and an event when the associated principal can easily be reconstructed, and speak of "the event $\langle m : A \to B\rangle$" for example. Finally, we will make liberal use of the convention of dropping parts of an action that can be reconstructed, and therefore may further streamline this example by speaking of "the event $\langle m\rangle$".

Several representations of processes and derived notions will prove convenient in different circumstance. Of course, a process is a directed acyclic graph (DAG) with events as nodes and $<$ as edges. Here is a simple example:

$$\begin{tikzcd}
 & (\nu y)_{A'} \arrow[rr] & & \langle f(x,y)\rangle_{A'} \arrow[dr] & \\
(\nu x)_A \arrow[ur] \arrow[rr] & & (z)_A \arrow[rr] & & \langle f(x,y)\rangle_A \\
 & & (u)_B & &
\end{tikzcd}$$

where the arrows flow in the opposite direction of $<$ and we assume $A' \Subset A$. We will sometimes explicitly render the ordering $<$, obtaining

$$\left[\begin{array}{c} (\nu x)_A < \left[\begin{array}{c} (\nu y)_{A'} < \langle f(x,y)\rangle_{A'} \\ (z)_A \end{array}\right] < \langle f(x,y)\rangle_A \\ \\ (u)_B \end{array}\right]$$

for the above example. Finally, we will occasionally use the conventions outlined in Section 2.2 to express roles, rendering $<$ as ";" and using "$\otimes$" to denote the absence of an ordering. The above example takes the following succinct form:

$$((\nu x)_A; (((\nu y)_{A'}; \langle f(x,y)\rangle_{A'}) \otimes (z)_A); \langle f(x,y)\rangle_A) \otimes (u)_B$$

It should be noted that not every DAG can be expressed in either of the last two notations (unless one is willing to repeat nodes, which would clash with our conventions). For example, the DAG at right cannot be rendered in these ways.

$$\begin{tikzcd}
a \arrow[r] & b \\
c \arrow[r] \arrow[ur] & d
\end{tikzcd}$$

Runs. A *run* of a process L assigns to each of its receive events a corresponding send event. Formally, a run is thus a pair

$$\begin{array}{ccccc}
\langle L, & \surd : & \mathsf{recvs}(L) & \longrightarrow & \mathsf{sends}(L)\rangle \\
 & & {\scriptstyle (x{:}Y\to Z)_A} & {\scriptstyle \longmapsto} & {\scriptstyle \langle m{:}S\to R\rangle_B}
\end{array}$$

such that

$$\sqrt{(x)} \not> (x)$$

The condition forces the send event mapped to a receive event to have occurred before this event. It prevents deadlocks, and protects the scope of the receiving variables (which is to the right, i.e., up in the partial order).

A run can also be viewed as an extension of the order $<$ of events in a process by adding $\sqrt{(x)} < (x)$ for every receive action $(x) \in \mathbb{L}$. We shall thus represent a run $\langle L, \surd\rangle$ simply by a process L where each receive event (x) has a unique predecessor $\langle m\rangle = \sqrt{(x)}$. A run of a process thus boils down to a Lamport order of actions.[4]

We pointed out earlier that a process corresponds to a collection of strands. In the same vein, a run is akin to a bundle in the strand world [7]. The main difference between our runs and bundles is that in the present framework, the variables can be used to follow the *execution* of the run, and track the data as it flows through it.[5]

Execution. The above definition falls short of capturing the intuitive notion of a run as a snapshot of the execution of a protocol. Indeed, while our runs correctly map receives to sends, they do not ensure that variables are properly handled. In particular, they allow events to take place past a failed match. Rather than giving syntactic restrictions to characterize a well-formed executable run, we keep our runs the way they are and define a notion of *execution* on them.

[4] This is in contrast with the representation of runs as process reductions à la Chemical Abstract Machine, used in the cord calculus [3, 6].

[5] If in a bundle a principal receives, say, the number 2, and then sends out the number 3, it is impossible to tell whether his program says to receive x and then send $x + 1$, or to send $2x - 1$, or perhaps to send 3 independently on what he receives.

A *slice* $(L^{\prec}, L^{\succ})$ is an order-conscious partition of a run $\langle L, \surd\rangle$, i.e., for every $l_1 \in L^{\prec}$ and $l_2 \in L^{\succ}$, it is not the case that $l_2 < l_1$. Every path in L will have a prefix in $L^{\prec}$ and the rest in $L^{\succ}$. We mark the meeting point with $^{\blacktriangledown}$.

Executing a run will consist of moving the markers $^{\blacktriangledown}$ rightward starting from an initial slice $(\emptyset, L)$ where there is a marker at the beginning of every path. The events in a run are executed in order: each $a \in \mathbb{L}$ can be executed only after all $b < a$ have been executed. Execution on any given path is specified by the following table:

Action	Form	If ...	... then do ...	... and write
send	$^{\blacktriangledown}\langle m : A \to B\rangle_C$	$FV(m) = \emptyset$		$\langle m : A \to B\rangle_C^{\blacktriangledown}$
receive	$^{\blacktriangledown}(x : Y \to Z)_D$	$\sqrt{(x : Y \to Z)_D}$ $= \langle m : A \to B\rangle_C$	in all $a > (x : Y \to Z)_D$ set $a(x := m, Y := A, Z := B)$	$(x : Y \to Z)_D^{\blacktriangledown}$
match	$^{\blacktriangledown}(m/p(\vec{x}))_D$	$\exists\vec{u}$ s.t. $m = p(\vec{u})$	in all $a > (m/p(\vec{x}))$ set $a(\vec{x} := \vec{u})$	$(m/p(\vec{x}))_D^{\blacktriangledown}$
		... *otherwise* ...	... *halt on this path*	$^{\blacktriangledown}(m/p(x))_D$
new	$^{\blacktriangledown}(\nu x)_D$			$(\nu x)_D^{\blacktriangledown}$
now	$^{\blacktriangledown}(\tau x)_D$			$(\tau x)_D^{\blacktriangledown}$

Note that execution on a path will stop when a match fails. It can however proceed on other paths.

REMARK 1 *In principle, a run all of whose actions have been successfully executed records all of the executed assignments. This distinguishes the computational assignment operation $x := m$ from the algebraic substitution operation (m/x). When substituting m for a variable x in a term a, we simply replace the occurrences of x by m; the result $a(m/x)$ generally bears no trace of x (unless it occurred in m). In contrast, when assigning m to x, we link x to m, thereby destroying any previous links of x, yet we do not erase the name of x itself. Indeed, in computation, x can later be reassigned to another term m'.*

When executing a run, the variables are assigned, but not destroyed. In this way, the data flow of a run is completely recorded, since each binding actions just performs assignments on some previously unassigned variables.

Executable Runs. The left component $L^{\prec}$ of a slice $(L^{\prec}, L^{\succ})$ of a run $\langle L, \surd\rangle$ satisfies the intuitive notion of run as a snapshot of the execution of a protocol. We adopt it as the definition of an *executable run*, and will denote such entities with the letter Q, variously annotated. From now on, all the runs we will be working with will be executable and we thus will generally drop this qualifier.

Local Observations. The *local observation* of a principal A consists of all the events performed by A in a run Q. It is simply defined as the projection of Q with respect to A:

$$Q_A \quad = \quad \{\ell \in Q \mid L_{\mathcal{W}}(\ell) = A\}$$

2.4 Logical Annotations

Having defined the notions of (executable) run and observation, we will now define a logical language to talk about these entities. This language applies the connectives and quantifiers of first-order logic to a base set of predicates. We will then be able to define a judgment that verifies that a formula constructed in this way is valid with respect to a run. It will then be a short step to use a formula to characterize all the runs in which it is valid, and to anchor it to the local observations of a principal.

Predicates. Our logical language contains just enough tools to query a run: the *event predicates* we will be relying on are

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

We will also admit the various relations participating in the definition of principals (e.g., $A \Subset B$), terms (e.g., $m \sqsubset m'$), etc, as additional predicates. A *formula* combines these atomic predicates by means of the traditional connectives and quantifiers of first-order logic. We will allow quantification over terms and principals appearing in an event, and, with a slight abuse of notation, over events themselves.

Formulas can be used to describe runs or portions of runs: simply turn a pair of connected events "$a \to b$" into the atomic predicate $a < b$, add the occurrence predicate a for any isolated event "a", and glue them together with $\wedge$. We write Φ_Q for the formula obtain in this way from a run Q. For example, the following formula captures the first few steps of an instance of the Denning-Sacco server role from Section 2.2 for a given server S:

$$\begin{array}{rl} & (m_0 : A \to S_0)_S < (S_0/S)_S \wedge (S_0/S)_S < (m_0/A, B)_S \\ \wedge & (m_0/A, B)_S < \mathrm{getKey}(A, K^{AS})_S \\ \wedge & (m_0/A, B)_S < \mathrm{getKey}(B, K^{BS})_S \\ \wedge & \mathrm{getKey}(A, K^{AS})_S < (\nu\, k)_S \wedge \mathrm{getKey}(A, K^{AS})_S < (\tau\, t)_S \\ \wedge & \mathrm{getKey}(B, K^{BS})_S < (\nu\, k)_S \wedge \mathrm{getKey}(B, K^{BS})_S < (\tau\, t)_S \end{array}$$

An automated theorem prover can make use of this formula, but it looks rather obscure to a human. For this reason, we will rely on generous notational conventions and express it in the more readable format:

$$(A, B : A \to S)_S < \begin{bmatrix} \mathrm{getKey}(A, K^{AS})_S \\ \mathrm{getKey}(B, K^{BS})_S \end{bmatrix} < \begin{bmatrix} (\nu\, k)_S \\ (\tau\, t)_S \end{bmatrix}$$

The following table lists some of the least obvious abbreviations we use:

This ...	... abbreviates ...	*Notes*
$(p)_A$	$(x)_A < (x/p)_A$	*Binders in p usually implicit*
$(\!(p)\!)_A$	$(x)_A < (x/p')_A \wedge p \sqsubseteq p'$	*Same binders in p and p'*
$\langle\!\langle m \rangle\!\rangle_A$	$\langle m' \rangle_A \wedge m \sqsubseteq m'$	
$\langle m \rangle_{A<}$	$\exists a = \langle m \rangle_A \wedge \forall b = \langle\!\langle m \rangle\!\rangle_B .\ a \leq b$	
$\langle\!\langle m \rangle\!\rangle_{A<}$	$\exists a = \langle\!\langle m \rangle\!\rangle_A \wedge \forall b = \langle\!\langle m \rangle\!\rangle_B .\ a \leq b$	
$a \prec b$	$b \Rightarrow a < b$	

Especially in logical statements, we will often omit the intended sender and recipient in a send or receive action when unimportant or easily reconstructible from the context.

Validation. Given a run Q and a formula Φ, a first task is to verify if Φ is *valid* in Q. We express this classical model checking problem by means of the judgment

$$[Q]\ \Phi$$

As usual, the definition of validity is inductive, with the following table expressing the validity of our basic event predicates:

Predicate	Judgment	If and only if	Meaning
event	$[Q]\ a$	$a \in Q$	*a has occurred in Q*
order	$[Q]\ a < b$	$a < b \in Q$	*a has occurred before b in Q*
equality	$[Q]\ a = b$		*a and b are the same event in Q*

The relational predicates on terms and principals are self-validating. The logical connectives and quantifiers are processed in the usual way.

Statements. If only a formula Φ is given, the above judgment can be used to implicitly define the set of all the runs that satisfy Φ:

$$\mathcal{R}_\Phi = \{q : [q]\ \Phi\}$$

In particular, if Φ describes the observations Q_A of a principal A in a given run Q, a formula we wrote Φ_{Q_A} earlier, the above definition allows us to characterize all the runs q that are compatible with Q_A:

$$\mathcal{R}_A = \{q : [q]\ \Phi_{Q_A}\}$$

While this satisfies the requirement at the beginning of this section, expressing $\mathcal{R}_A$ in this way sheds little light on the structure that a run must have to be compatible with A's observations. In the next section, we will instead strive to explicitly characterize these runs by means of a formula Φ of maximal generality. Φ will be such that:

$$\mathcal{R}_A = \{q : [q]\ \Phi_{Q_A}\} = \{q : [q]\ \Phi_{Q_A} \wedge \Phi\}$$

We will generally keep Φ_{Q_A} explicit by expressing Φ as the logically equivalent $\Phi_{Q_A} \Rightarrow (\Phi_{Q_A} \wedge \Phi)$.

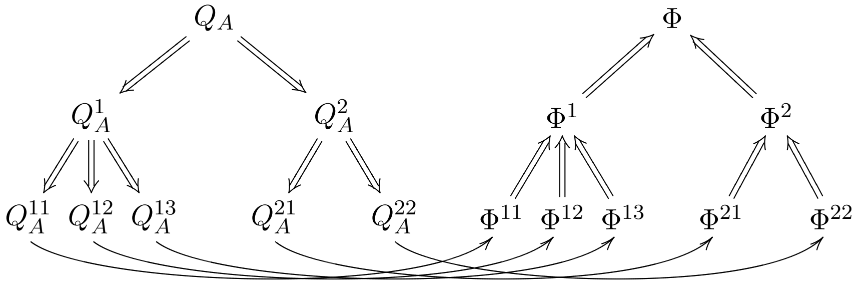
We will deduce this formula from the axioms and inference rules described in the next section to get as clear a picture as possible of $\mathcal{R}_A$. By having Φ be an implication $\Phi' \Rightarrow \Phi''$, we can characterize important or interesting portions of $\mathcal{R}_A$ that satisfy the assumption Φ': we will typically assume the honesty of principals or the fact that a key has not been compromise. Note that $\Phi_{Q_A} \Rightarrow (\Phi_{Q_A} \wedge (\Phi' \Rightarrow \Phi''))$ is logically equivalent to $(\Phi_{Q_A} \wedge \Phi') \Rightarrow (\Phi_{Q_A} \wedge \Phi'')$. Given the prominence of this notion in the rest of our work, we will abbreviate $\Phi_{Q_A} \wedge \Phi$ as

$$A : \Phi$$

Φ is then a description of the runs compatible with A's observations. We will often call Φ the *knowledge* of A. As we said, Φ will generally have the form $\Phi_{Q_A} \wedge \Phi' \Rightarrow \Phi_{Q_A} \wedge \Phi''$.

2.5 Axioms and Rules

Let A be a principal executing the role ρ of a protocol P. Given a run Q and local observation Q_A for A's execution of ρ, this section presents the tools to synthesize a formula Φ such that $A : \Phi$, i.e., that characterizes the runs compatible with Q_A (possibly restricted by appropriate assumptions). In order to do so, we isolate the elementary constituents of Q_A (for example challenge-response exchanges) and produce formulas that describe the runs compatible with them (the necessary behavior of a counterpart in the case of challenge-response). We then combine these formulas into larger formulas corresponding to bigger parts of Q_A, all the way to A's original observation. An intuitive picture of this process is given below:



More precisely, the derivation of a formula Φ characterizing the runs compatible with a local observation Q_A draws from two ingredients:

Axioms An axiom maps an elementary observation with a formula expressing the necessary behaviors of the interacting parties. Axioms are universal predications about basic patterns of events. In the illustration above, the axioms corresponds to the single arrows connecting the leaves of the trees. We will spend the rest of this section justifying a number of common axioms.

Transformations A transformation maps a method for building a complex observation from simpler ones to a method for upgrading the formulas associated with them to a formula describing the resulting observation. A transformation may extend a partial observation with additional events, or enrich individual events with new components, or combine events by merging common terms. We will see several transformations in the sections to come. Had we found a good way to draw transformations in the above illustration, they would relate the branches exiting an inner node on the left-hand side to the branches entering the corresponding node on the right-hand side.

Before we describe some of the most fundamental axioms of the Protocol Composition System, a few definitions will save us some space.

Honesty Assumption. In the sequel, we will occasionally need a principal A deducing $A : \Phi$ to assume that another principal B is *honest* in order to draw interesting or meaningful conclusions. By this, we mean that B does not deviate from his assigned role ρ' as he interacts with A. For the sake of illustration, let ρ' be completely sequential:

$$\rho'[B] \;=\; b^1; \ldots ; b^i; \ldots ; b^n$$

Therefore, if A is able to deduce that an honest B has executed any given action b_i in this role, she can safely infer that he has executed all the actions leading to b_i in ρ' as well. The resulting formula for the above example is as follows:

$$\mathsf{Honest}_{\rho'}\, B \;\triangleq\; (b^1)_B \prec \ldots \prec (b^i)_B \prec \ldots \prec (b^n)_B$$

(Recall that $a \prec b$ abbreviates $b \Rightarrow a < b$.) We will generally keep the role ρ' implicit. Clearly, honesty formulas are associated with every role, not just the sequential ones. For example, the honesty definition for the server role of the Denning-Sacco protocol in Section 2.2 is as follows:

$$(A, B : A \to S)_S \prec \begin{bmatrix} \mathrm{getKey}(A, K^{AS})_S \\ \mathrm{getKey}(B, K^{BS})_S \end{bmatrix} \prec \begin{bmatrix} (\nu\, k)_S \\ (\tau\, t)_S \end{bmatrix} \prec$$

$$\prec \langle K^{AS}(B, k, t, K^{BS}(k, A, t)) : S \to A\rangle_S$$

(We are relying on the abbreviations in Section 2.4 for succinctness.)

The honesty definition will be used exclusively as an assumption so that $A : \Phi$ will often have the form $A : \mathsf{Honest}\ B \Rightarrow \Phi'$. We will see that some principals need to be assumed honest for the formula inferred from A's observations to be compatible with the legal runs of the protocol, while other principals may be dishonest and yet cannot substantially deviate from the protocol given A's observations.

Uncompromised Key Assumption. Another important assumption we will need to make is 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,

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

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, while the second line 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.

In this paper, we will use $\mathsf{uncompromised}$ exclusively as an assumption. Moreover, we will make such an assumption for every key we need to believe is not compromised as our system does not contain any axiom explaining how shared keys ought to be used.

The reasons for this choice are rather subtle and deserve further explanation. Key distribution protocols juggle two long-scrutinized properties: secrecy and authentication. The distributed key can be secret only if it is transmitted inside authenticated messages. In turn, a message can be authenticated only if it protects its contents using a secret key, which brings us back to the problem of distributing this secret key. This is a chicken and egg situation. The only way to break this circularity is to assume either the existence of a shared secret key or the existence of an authenticated channel. We choose the first alternative, although the second option (e.g., using a private communication medium) would be equally valid. Assuming certain long-term keys to be secret (i.e., uncompromised) immediately yields that any message they encrypt are authenticated.

Now, a key distribution protocol transmits a freshly generated key k along these authenticated channels to some principals A and B. The next question becomes how to prove that the protocol ensures the secrecy of k, i.e., that $\mathsf{uncompromised}(k, [A, B])$ holds. This question will be the focus of a sequel to this paper, and we shall not address it further here. There, a *proof* of $\mathsf{uncompromised}(k, [A, B])$ will permit discharging an *assumption* of $\mathsf{uncompromised}(k, [A, B])$, which is very useful for staged protocols such as Kerberos, where a key is distributed for the purpose to protecting another key.

A number of authors have proposed techniques to prove secrecy properties, e.g., Schneider's rank functions [16] and Thayer *et al*'s ideals [18] just to cite a few. At heart, they are all based on a form of closed-word assumption which limits the class of available actions and then rely on an inductive argument to prove that the key cannot be revealed. The present paper is instead open-ended: all events are allowed unless expressly forbidden (e.g., by an uncompromised assumption).

We will now discuss a number of axioms and axiom schemas that will provide some of the foundation for the rest of the paper.

Freshness axiom. We start with a general axiom describing the behavior of the $(\nu\, n)$ action in logical terms:

$$(\nu\, n)_B \Rightarrow \forall a_A.\ \Big(n \in FV(a)\ \Rightarrow (\nu\, n)_B < a_A \\ \wedge\ \big(A \neq B \Rightarrow (\nu\, n)_B < \langle\!\langle n \rangle\!\rangle_B < (\!(n)\!)_A \leq a_A\big)\Big) \quad \text{(new)}$$

The first part implies that ν is a binder, which means that any event a mentioning n necessarily occurs *after* $(\nu\ n)$ (recall that we required binders not to recycle variable names for simplicity). 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; said in other words, values freshly generated using ν can only be transmitted using the send/receive mechanism.

"Not Me!" Axiom. The next axiom is equally general: it says that if an observer A is aware that some X has executed an action a, but A never executed any such action, then X cannot be A:

$$A:\ a_X \wedge \neg a_A \rightarrow X \neq A \quad \text{(notme)}$$

This axiom relies on the fact that an observer is aware of all of its actions. It will turn useful, for example, in conjunction with the uncompromised assumption for A to deduce that an encrypted message originated by the principal she is sharing the key with (and not herself).

Send-Receive Axiom Schemas. Next, we examine a general class of axioms allowing a principal A to infer the existence of a specific send event matching a receive she has observed. They are all subsumed by the following schema:

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

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 moreover X and $\vec{y}$ do satisfy some postcondition Ψ.

A number of important axioms capturing the semantics of interaction through send and receive events are subsumed under this schema, by instantiating f^{AX}, Φ and Ψ. We will now examine a few.

Receive Axiom. In the simplest case, where Φ and Ψ are taken to be trivially true, and f^{AX} is arbitrary, the axiom just says that everything that is received must have been sent by someone:

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

Challenge-Response Axiom Schema. Perhaps the most useful instance of (sr) is another axiom schema describing the requirements for nonce-based challenge-response exchanges. It is obtained for:

$$\begin{aligned} f^{AX}(y) &= r^{AX}(y) \\ \Phi(X,y) &= \Phi' \wedge (\nu\, y)_A < \langle\!\langle c^{AX}(y)\rangle\!\rangle_{A<} < (\!(r^{AX}(y))\!)_A \\ \Psi(X,y) &= (\nu\, y)_A < \langle\!\langle c^{AX}(y)\rangle\!\rangle_{A<} < (\!(c^{AX}(y))\!)_X < \langle\!\langle r^{AX}(y)\rangle\!\rangle_{X<} \end{aligned}$$

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 $\mathsf{uncompromised}$ assumption.

Simplifying yields:

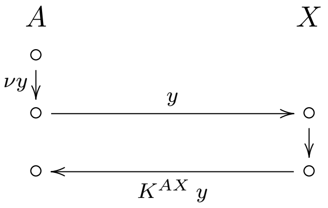
$$\begin{aligned} A:\Phi' \wedge (\nu y)_A < \langle\!\langle c^{AX}y\rangle\!\rangle_{A<} &\qquad\qquad\qquad\qquad < (\!(r^{AX}y)\!)_A \\ \Rightarrow (\nu y)_A < \langle\!\langle c^{AX}y\rangle\!\rangle_{A<} < (\!(c^{AX}y)\!)_X &< \langle\!\langle r^{AX}y\rangle\!\rangle_{X<} < (\!(r^{AX}y)\!)_A \end{aligned}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

As an example of an actual instance of this axiom, we consider the case in which c^{AX} is the identity (the nonce is sent in the clear), the response encrypts the nonce with a key K^{AX} shared between A and X, and Φ' requires K^{AX} not to be compromised for A and X. We obtain

$$\begin{aligned} A:\quad & \mathsf{uncompromised}(K^{AX},[A,X]) \wedge \\ & (\nu y)_A < \langle\!\langle y\rangle\!\rangle_{A<} \qquad\qquad\qquad\qquad < (\!(K^{AX}y)\!)_A \\ \Rightarrow\ & (\nu y)_A < \langle\!\langle y\rangle\!\rangle_{A<} < (\!(y)\!)_X < \langle\!\langle K^{AX}\ y\rangle\!\rangle_{X<} < (\!(K^{AX}\ y)\!)_A \end{aligned}$$

A proof of this axiom goes as follows: starting from A's own observations (the second line above), axiom rcv entails that some agent Y has originated $\langle\!\langle K^{AX}\ y\rangle\!\rangle$. By the $\mathsf{uncompromised}$ assumption, Y must be either X or A, with axiom notme excluding the latter possibility. Axiom new completes the second line by sandwiching X's reception of y between A's transmission of the nonce and X's issuing of $\langle\!\langle K^{AX}\ y\rangle\!\rangle$.

In the sequel, we will represent a run of this challenge-response exchange by means of the following diagram:



Timestamps. Other useful instances of the *sr* axiom schema describe the semantics of timestamps. Here is one possibility. Consider the following values for our various meta-variables:

$$\begin{aligned} f^{AX}(t) &= t \\ \Phi(X,t) &= \langle\!\langle t\rangle\!\rangle_{X<} \\ \Psi(X,t) &= \mathsf{honest}\, X \Rightarrow \big((\underline{\tau}\, t)_A < (\tau t)_X < \langle\!\langle t\rangle\!\rangle_{X<} \ \wedge\ (\!(t)\!)_A < (\overline{\tau}\, t)_A\big) \end{aligned}$$

Let us instantiate and simplify sr before commenting on it:

$$\begin{aligned} A: \quad & \mathsf{honest}\, X \ \wedge \qquad \langle\!\langle t\rangle\!\rangle_{X<} < (\!(t)\!)_A \\ & \Rightarrow (\underline{\tau}\, t)_A < (\tau t)_X < \langle\!\langle t\rangle\!\rangle_{X<} < (\!(t)\!)_A < (\overline{\tau}\, t)_A \end{aligned} \qquad (\mathsf{ts})$$

The antecedent of this formula assumes that X is honest (which here means that his expected behavior is to look up a timestamp and send it out), A receives a message containing an acceptable timestamp t, and she has the certainty that 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.

Note that there are other options for giving the semantics of timestamps as an instance of sr: the above approach will however prove particularly convenient in the sequel.

Diffie-Hellman. Although we will not make use of this mechanism in this paper, it is interesting to note that the sr axiom schema also specializes to a crude logical description of the Diffie-Hellman exchange (where, for simplicity, we have the responder transmit the shared secret in some message). We instantiate the various schematic variables as follows:

$$\begin{aligned} f^{AX}(g,u,y) &= u^y \\ \Phi(X,g,u,y) &= (\nu\, y)_A < \langle g^y\rangle_{A<} < (u)_A \wedge \neg\langle y\rangle_A \\ \Psi(X,g,u,y) &= \langle u\rangle_{X<} \wedge (\neg\langle \log u\rangle_X \Rightarrow (g^y)_X < \langle\!\langle u^y\rangle\!\rangle_{X<}) \end{aligned}$$

for appropriate term constructors for exponentiation and discrete logarithm. Here g is the group generator, y is A's random number, u is X's returned value ($u = g^z$ where z is X's random number), and therefore $u^y = g^{yz}$ is the shared secret. Simplifying and rearranging this time yields:

$$\begin{array}{ll} A: & \neg\langle y\rangle_A \wedge \neg\langle \log u\rangle_Z \wedge \\ & (\nu y)_A < \langle g^y\rangle_{A<} \qquad\qquad\qquad\qquad < \begin{bmatrix} (u)_A \\ ((u^y))_A \end{bmatrix} \qquad \text{(dh)} \\ \multicolumn{2}{l}{\Rightarrow \exists X.\ (\nu y)_A < \langle g^y\rangle_{A<} < (g^y)_X < \begin{bmatrix} \langle u\rangle_{X<} \\ \langle\!\langle u^y\rangle\!\rangle_{X<} \end{bmatrix} < \begin{bmatrix} (u)_A \\ ((u^y))_A \end{bmatrix}} \end{array}$$

This formula states that if A receives her counterpart's share of the secret and a message containing the secret, and neither exponent has been leaked, then she can rest assured that some X has received her own g^y and send those two messages.

3. Basic Key Distribution

In this section, we apply the methodology just outlined to obtain the core protocols and logical guarantees for key distribution. For the time being, we are only interested in the manner a key server can distribute a fresh key to clients. We will examine other important aspects of key distribution, namely recency and key confirmation, in Sections 4 and 5, where we derive NSSK and Kerberos, respectively.

We warm up in Section 3.1 with an illuminating exercise in futility: having a server distribute a fresh key to a single principal. We take it as a template for the more useful two-client setting in Section 3.2.

3.1 One-Party Key Distribution

We begin with a very simple setup consisting of a key server S and one client A. While the resulting protocol, which distributes a secret key to a single principal, makes little sense in practice, it will serve as a useful illustration of the concepts introduced so far and help gain familiarity with transformations. Later, when applying these techniques to more realistic protocols, we will be able to concentrate on the derived properties rather than on minor technicalities.

In our initial version of this protocol, both the key server and the client are given each other's name as a parameter to their respective roles. A fully spelled-out specification of these roles is as follows:

$$\begin{array}{rcl}
\mathsf{KD}^0_1\_\mathsf{server}[S;A] & = & \text{getKey}(A, K^{AS}) \ ; \ \nu\, k \ ; \ \langle K^{AS}\, k : S \to A \rangle \\
\mathsf{KD}^0_1\_\mathsf{client}[A;S] & = & (m_0 : S_0 \to A_0) \ ; \ (A_0/A) \ ; \ (S_0/S) \ ; \\
 & & \text{getKey}(S, K^{AS}) \ ; \ (m_0/K^{AS}\, k)
\end{array}$$

(We will abbreviate it shortly.) The server "knows" it should send the new key k to A because A appears in its parameter list.

A process involving one instance of each of these roles is given by the nodes and solid lines in the graph below (please ignore everything else for now). We have written A and S for the instantiating values of the parameters A and S, and distinguished binders with the same name in the two roles by means of primes. The expected run of this protocol bridges these two halves by means of the dashed arrow.

$$\begin{tikzcd}
 & \mathrm{getKey}(\mathsf{A}, K^{AS})_{\mathsf{S}} \arrow[d, "{\{K^{AS}:=\mathsf{k}^{\mathsf{AS}}\}}"] \\
 & (\nu k)_{\mathsf{S}} \arrow[d] \\
(m_0 : S_0 \to A_0)_{\mathsf{A}} \arrow[d, "{\{\ldots,m_0:=K^{AS}\,k,S_0:=\mathsf{S},A_0:=\mathsf{A}\}}"'] & \langle K^{AS}\,k : \mathsf{S} \to \mathsf{A}\rangle_{\mathsf{S}} \arrow[l, dashed] \\
(A_0/\mathsf{A})_{\mathsf{A}} \arrow[d] & \\
(S_0/\mathsf{S})_{\mathsf{A}} \arrow[d] & \\
\mathrm{getKey}(\mathsf{S}, K'^{AS})_{\mathsf{A}} \arrow[d, "{\{\ldots,K'^{AS}:=\mathsf{k}^{\mathsf{AS}}\}}"'] & \\
(m_0/K'^{AS}\,k')_{\mathsf{A}} \arrow[d, "{\{\ldots,k':=k\}}"'] & \\
 &
\end{tikzcd}$$

Another expression for this process is

$$\begin{array}{ll} & \mathrm{getKey}(\mathsf{A}, K^{AS})\ ;\ \nu\, k\ ;\ \langle K^{AS}\,k : \mathsf{S} \to \mathsf{A}\rangle \\ \otimes & (m_0 : S_0 \to A_0)\ ;\ (A_0/\mathsf{A})\ ;\ (S_0/\mathsf{S})\ ;\ \mathrm{getKey}(\mathsf{S}, K'^{AS})\ ;\ (m_0/K'^{AS}\,k') \end{array}$$

The run assigns $\sqrt{(m_0 : S_0 \to A_0)_{\mathsf{A}}} = \langle K^{SA}\,k : \mathsf{S} \to \mathsf{A}\rangle_{\mathsf{S}}$.

An execution of this run accumulates binding variable assignments in an environment that we have expressed above as annotation to the arrows. By the time the last action has been executed, this environment contains:

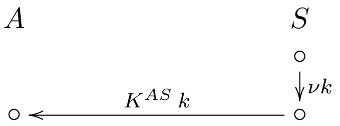
$$\{K^{AS} := \mathsf{k}^{\mathsf{AS}},\ m_0 := K^{AS}\,k,\ S_0 := \mathsf{S},\ A_0 := \mathsf{A},\ K'^{AS} := \mathsf{k}^{\mathsf{AS}},\ k' := k\}$$

By the end of this run, the observations Q_{A} and Q_{S} of A and S correspond to the left and right column of the above figure, respectively.

While we hope this demonstrated the definitions in Section 2.3, the remainder of this paper will use a leaner notation based on the conventions introduced in the previous section. This will make our treatment more readable and save space. In particular, we will liberally fold matches inside receive actions, occasionally omit senders and receivers, and keep the internal actions getKey implicit. The roles in this example then reduces to:

$$\begin{array}{rcl} \mathsf{KD}^0_1\_\mathsf{server}[S;A] & = & \nu\, k\ ;\ \langle K^{AS}\, k : S \rightarrow A\rangle \\ \mathsf{KD}^0_1\_\mathsf{client}[A;S] & = & (K^{AS}\, k : S \rightarrow A) \end{array}$$

We also summarize the above run in the following strand-like picture:



We are now ready to take the point of view of each principal and infer a formula representing the runs that are compatible with its observations. Let us take the part of A first. In our abbreviated notation, the only event she observes is $(K^{AS}\ k\ :\ S \rightarrow A)$. Under the assumptions that K^{AS} is uncompromised for A and S and the honesty of S (derived from his role), A can completely reconstruct the expected run. In symbols:

$$\begin{array}{rr} A: & \mathsf{uncompromised}(K^{AS},[A,S])\ \wedge\ \mathsf{honest}\ S\ \wedge \\ & (K^{AS}\ k : S \rightarrow A)_A \\ & \Rightarrow (\nu\, k)_S\ <\ \langle K^{AS}\ k : S \rightarrow A\rangle_{S<}\ <\ (K^{AS}\ k : S \rightarrow A)_A \end{array}$$

This formula means that, under the stated hypotheses, all runs that are compatible with A's observation must include the shown actions of S, and in the prescribed order.

The uncompromised hypothesis identifies S as the originator of the transmitted message (A is excluded thanks to the notme axiom) while the honesty assumption completes the sequence of messages that preceded this send. A more formal derivation is given by:

$$\begin{array}{rl} Q_A & :\ (K^{SA}\, k : S \rightarrow A)_A \\ (\mathsf{rcv}) & :\ \langle\!\langle K^{SA}\, k\rangle\!\rangle_{X<} < (K^{SA}\ k : S \rightarrow A)_A \\ \mathsf{uncompromised}(K^{AS},[A,S]) & :\ X = A \text{ or } X = S \\ (\mathsf{notme}) & :\ X \neq A \\ \mathsf{honest}\ S & :\ (\nu k)_S \prec \langle K^{SA}(k) : S \rightarrow A\rangle_S \\ \hline & (\nu k)_S < \langle K^{SA}\ k : S \rightarrow A\rangle_{S<} < (K^{SA}\ k)_A \end{array}$$

On the other hand, S does not conclude more than he observes since he is the recipient of no message.

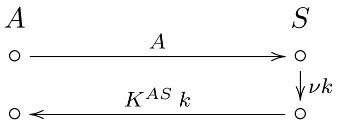
In this example, A is a parameter in S's role and S is a parameter in A's. In a real system, this would mean that these values appear in some configuration file on the client and server's machines. While the former may be acceptable if there is only one server in the system, the latter is certainly not as a server should be able to distribute keys to more than one client.

We introduce the *discharging transformation* $\mathbb{DC}$ to transform a role that gets a value from a parameter into a role that acquires this value as part of the protocol run. This transformation simply turns a parameter into a binder.[6] For simplicity, take a sequential role $[x]\rho; \rho'(x)$ with parameter x, an action prefix ρ that does not refer to x, and remaining actions $\rho'(x)$ that may reference x. Then, $\mathbb{DC}$ is defined as follows on this role:

$$\mathbb{DC}[\,[x]\rho; \rho'(x)\,] \;\triangleq\; \rho;\, a^x; \rho'(x)$$

where a^x is some action that binds x (most interesting is a receive).[7] The generalization to non-sequential roles is trivial, but harder to typeset.

For example, an instance of $\mathbb{DC}$ discharges A in KD^0_1_server above by having A send her name to S in a request message:



The roles give a more precise account of the operations of $\mathbb{DC}$:

$$\begin{array}{lcl} \mathsf{KD}^1_1\text{\_server}[S] & = & (A : A \to S)\ ;\ \nu\, k\ ;\ \langle K^{AS}\, k : S \to A\rangle \\ \mathsf{KD}^1_1\text{\_client}[A; S] & = & \langle A : A \to S\rangle\ ;\ (K^{AS}\, k : S \to A) \end{array}$$

Observe that KD^1_1_server does not have A as a parameter. S obtains A's name from the first message, either from the body or from its putative sender at the implementor's choice. A simple prefixing transformation is used to have the client send this message, upgrading KD^0_1_client to KD^1_1_client. Note that it does not discharge S as a parameter.

A transformation operates not only on the syntactic specification of a role, but also on its inferable properties. In its generality, the transformation $\mathbb{DC}$ is rather limited in this respect: it extends the observations of principal executing the affected role with the action a^x — here $(A : A \to S)$ — and only influences the deductions of other principals through its altered honesty assumption. From the point of view of the principal executing the transformed role,

[6] There are other ways to discharge a parameter x: for example another useful transformation removes a match (x_0/x) against x and replaces other occurrences with x_0. This would be how to discharge S in KD^0_1_client above.

[7] **EdNote:** Specialize to just receive?

$\mathbb{DC}$ operates as follow on the sequential illustration above, where we make an intuitive use of the symbols.

$$\begin{array}{lclcl} A: & \Phi_\rho < \Phi_{\rho'} \wedge \Psi & \rightarrow & \Phi'_\rho < \Phi'_{\rho'} \wedge \Psi' \\ & & \Big\Downarrow \mathbb{DC} & & \\ A: & \Phi_\rho < a^x < \Phi_{\rho'} \wedge \Psi & \rightarrow & \Phi'_\rho < a^x < \Phi'_{\rho'} \wedge \Psi' \end{array}$$

We will see transformations that have more interesting effects shortly. Note however that the presence of event a^x may enable further inferences.

This makes S's point of view marginally more interesting than in the first version of this protocol: upon receiving the first message, he can use axiom rcv to infer that someone sent it, although not necessarily A:

$$\begin{array}{lrl} S: & & (A : A \rightarrow S)_S \ < \ (\nu\, k)_S \ < \ \langle K^{AS}\, k\rangle_S \\ & \langle A : A \rightarrow S\rangle_{X<} \ < & (A : A \rightarrow S)_S \ < \ (\nu\, k)_S \ < \ \langle K^{AS}\, k\rangle_S \end{array}$$

We next examine the property deducible by A: it illustrates the effect of $\mathbb{DC}$ on the other party, and describe the effect of the prefixing transformation. Her view is summarized by the following formula:

$$\begin{array}{lll} A: & \mathsf{uncompromised}(K^{AS}, [A, S]) \ \wedge \ \mathsf{honest}\ S \ \wedge & \\ & \qquad\qquad \langle A : A \rightarrow S\rangle_A & < \ (K^{AS}\ k)_A \\ \Rightarrow & \left[\begin{array}{c} \langle A : A \rightarrow S\rangle_A \\ (A : A \rightarrow S)_S \ < \ (\nu\, k)_S \ < \ \langle K^{AS}\ k\rangle_{S<} \end{array}\right] & < \ (K^{AS}\ k)_A \end{array}$$

The two occurrences of $\langle A : A \rightarrow S\rangle_A$ are the result of the prefixing transformation on the client's role. The matching receive action $(A : A \rightarrow S)_S$ is deduced from the honesty of S. Observe that A is unable to correlate her sending of this message with S's reception. Indeed, S will perform its role not in response to A's request but following the reception of any message of the form $(A : A \rightarrow S)$, whoever the actual sender is. Of course, A will not accept an unsolicited key, but if she sent a request there is no guarantee that S's response has any relation to it. While this property does not exactly match the expected run of this updated protocol, it may be acceptable since A still gets what she asked for (even if she was not heard). We will examine variants of this protocol that enforce stronger guarantees between request and response.

3.2 Two-Party Key Distribution

With this exercise under our belt, we will now examine protocols in which a server S generates a key k and distributes it to two parties A and B. This is the setting underlying NSSK and Kerberos, which we will study in sections to come. Note that our analysis generalizes to an arbitrary number of parties.

We start with the 2-party variant of the basic scheme presented in Section 3.2. The expected run is as follows:

$$\begin{tikzcd}
A & S & B \\
 & \circ \arrow[d, "\nu k"] & \\
\circ & \circ \arrow[l, "K^{AS}\,k"'] \arrow[r, "K^{BS}\,k"] & \circ
\end{tikzcd}$$

While we take it as primitive for simplicity, it is easy to define a transformation that produces an n-party variant of that basic protocol in Section 3.1 for any given number n.[8]

The roles of this protocol are defined next. Notice that the actions of A and B are totally symmetric at this stage (only A's role is shown).

$$\begin{array}{lcl}
\mathsf{KD}^0_2\text{\_}\mathsf{server}[S;A;B] & = & \nu\,k\;;\;\langle K^{AS}\,k : S \to A\rangle \otimes \langle K^{BS}\,k : S \to B\rangle \\
\mathsf{KD}^0_2\text{\_}\mathsf{client}[A;S,B] & = & (K^{AS}\,k : S \to A)
\end{array}$$

Next we take the point of view of a client (A for example) and follow our footprints from Section 3.1 to derive the property characterizing her observations:

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

Next we use the discharging transformation $\mathbb{DC}$ to have A pass the names of the two clients to S, discharging A and B as parameters in $\mathsf{KD}^0_2\text{\_}\mathsf{server}$. The resulting run is given by:

$$\begin{tikzcd}
A & S & B \\
\circ \arrow[r, "A{,}B"] & \circ \arrow[d, "\nu k"] & \\
\circ & \circ \arrow[l, "K^{AS}\,k"'] \arrow[r, "K^{BS}\,k"] & \circ
\end{tikzcd}$$

and the roles by:

$$\begin{array}{lcl}
\mathsf{KD}^1_2\text{\_}\mathsf{server}[S] & = & (A,B : A \to S)\;;\;\nu\,k\;; \\
 & & \langle K^{AS}\,k : S \to A\rangle \otimes \langle K^{BS}\,k : S \to B\rangle \\
\mathsf{KD}^1_2\text{\_}\mathsf{iclient}[A;S,B] & = & \langle A,B : A \to S\rangle\;;\;(K^{AS}\,k : S \to A) \\
\mathsf{KD}^1_2\text{\_}\mathsf{rclient}[B;S,A] & = & (K^{BS}\,k : S \to B)
\end{array}$$

[8] **EdNote:** Cathy wrote "Composing $\mathbb{KD}_1[A,S]$ with $\mathbb{KD}_1[B,S]$ yields $[\mathbb{KD}_2]$". I tried to define such a "composition" in a number of ways, but I could not find anything satisfactory.

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 derived as in the previous section. Let us examine them:

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

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

Observe that A has no way to determine whether S transmitted the key k to B or to some other party X. Indeed, 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. In our setting, this is achieved by a transformation $\mathbb{CA}$ that inserts a new term into an existing encryption:

$$\mathbb{CA}_{m'}[\,k\,m\,] \;\triangleq\; k\,(m, m')$$

This has the effect of cryptographically authenticating m' (hence the name $\mathbb{CA}$) to any party entitled to access the ciphertext. It operates as follows on a property derivable to a party receiving this message (we omit additional formulas that may occur in the antecedant or consequent):

$$\begin{array}{rl}
A: & \mathsf{uncompromised}(k, [A,B]) \wedge (\!(k\; m)\!)_A \\
\Longrightarrow & \langle\!\langle k\; m\rangle\!\rangle_{B<} < (\!(k\; m)\!)_A
\end{array}$$

$$\Big\Downarrow \mathbb{CA}_{m'}$$

$$\begin{array}{rl}
A: & \mathsf{uncompromised}(k, [A,B]) \wedge (\!(k\;(m,m'))\!)_A \\
\Longrightarrow & \langle\!\langle k\;(m,m')\rangle\!\rangle_{B<} < (\!(k\;(m,m'))\!)_A
\end{array}$$

The transformation simply extends to the added component m' the fact that a message encrypted with an uncompromised key is authenticated. Note that B's honesty is not required as long as k is not compromised.[9]

By applying this transformation 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. The expected run is now given by the following diagram:

$$\begin{tikzcd}
A & S & B \\
\circ \arrow[r, "{A,B}"] & \circ \arrow[d, "\nu k"] & \\
\circ & \circ \arrow[l, "{K^{AS}\,(B,k)}"'] \arrow[r, "{K^{BS}\,(A,k)}"] & \circ
\end{tikzcd}$$

while the roles become:

$$\begin{array}{lcl}
\mathsf{KD}_2^2\mathsf{\_server}[S] & = & (A, B : A \to S)\ ;\ \nu\, k\ ; \\
 & & \langle K^{AS}\,(B,k) : S \to A\rangle \otimes \langle K^{BS}\,(A,k) : S \to B\rangle \\
\mathsf{KD}_2^2\mathsf{\_iclient}[A; S, B] & = & \langle A, B : A \to S\rangle\ ;\ (K^{AS}\,(B,k) : S \to A) \\
\mathsf{KD}_2^2\mathsf{\_rclient}[B; S] & = & (K^{BS}\,(A,k) : S \to B)
\end{array}$$

It is easy to see that the application of $\mathbb{CA}$ solves the problem outlined earlier. Indeed, A and B can derive the following properties:

$$\begin{array}{ll}
A: & \mathsf{uncompromised}(K^{AS}, [A,S])\ \wedge\ \mathsf{honest}\ S\ \wedge \\
 & \qquad\qquad \langle A,B\rangle_A \qquad\qquad\qquad\qquad < \ (K^{AS}\,(B,k))_A \\
\Rightarrow & \left[\begin{array}{c} \langle A,B\rangle_A \\ (A,B)_S \ < \ (\nu\, k)_S \ < \ \left[\begin{array}{l}\langle K^{AS}\,(B,k)\rangle_{S<} \\ \langle K^{BS}\,(A,k)\rangle_{S<}\end{array}\right]\end{array}\right] \ < \ (K^{AS}\,(B,k))_A
\end{array}$$

$$\begin{array}{ll}
B: & \mathsf{uncompromised}(K^{BS}, [B,S])\ \wedge\ \mathsf{honest}\ S\ \wedge \\
 & \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad (K^{BS}\,(A,k))_B \\
\Rightarrow & (A,B)_S \ < \ (\nu\, k)_S \ < \ \left[\begin{array}{l}\langle K^{AS}\,(B,k)\rangle_{S<} \\ \langle K^{BS}\,(A,k)\rangle_{S<}\end{array}\right] < \ \ (K^{BS}\,(A,k))_B
\end{array}$$

While these formulas are very 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). Clearly, this correction becomes crucially important when A and B attempt to use k.

While KD_2^2 achieves a minimal form of key distribution (we will soon extend this basic functionality with additional guarantees), few actual protocols

[9] **EdNote:** Collect all transformations, except maybe 1 or 2, in the appendix?

have this message structure. Indeed, with the exception of recent group protocols [10], 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 relays the part he does not understand to the other.

Appendix A.0 describes the relay transformation $\mathbb{RT}$ that has the ability to turn KD_2^2 into a more common form of key distribution. The resulting run is as follows:

$$\begin{tikzcd}
A & S & B \\
\circ \arrow[r, "{A,B}"] & \circ \arrow[d, "\nu k"] & \\
\circ \arrow[d] & \circ \arrow[l, "{K^{AS}\ (B,k),\ K^{BS}\ (A,k)}"'] & \\
\circ \arrow[rr, "{K^{BS}\ (A,k)}"] & & \circ
\end{tikzcd}$$

In this 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 academic and industrial protocols, e.g., Kerberos 5, follow this pattern. The role specification is as follows:

$$\begin{aligned}
\mathsf{KD}_2^3\_\mathsf{server}[S] &= (A,B : A \to S)\ ;\ \nu\, k\ ; \\
&\ \langle K^{AS}\ (B,k),\ K^{BS}\ (A,k) : S \to A\rangle \\
\mathsf{KD}_2^3\_\mathsf{iclient}[A;S,B] &= \langle A,B : A \to S\rangle\ ;\ (K^{AS}\ (B,k), M : S \to A)\ ; \\
&\ \langle M : A \to B\rangle \\
\mathsf{KD}_2^3\_\mathsf{rclient}[B;S] &= (K^{BS}\ (A,k) : A \to B)
\end{aligned}$$

Clearly, the component $K^{BS}\ (A,k)$ is opaque to A. Hence her role mentions a generic message M.

Transformation $\mathbb{RT}$ alters the properties derivable to A and B in a rather subtle way. We examine its effect one principal at a time.

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

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 major implications. We highlighted them using boxes:

1 While, by the honesty assumption, A knows that S has sent $K^{AS}\ (B,k)$, $K^{BS}\ (A,k)$, she has no means to 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 direct 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 [2].

Additionally, observe that A's last send has little bearing on the overall property and could be dropped without significant consequences (it is the same underlying reason that makes the property derivable by the server so uninteresting).

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

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

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

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

The transformation $\mathbb{EA}$ that produces this modified protocol is similar to $\mathbb{CA}$:

$$\mathbb{EA}[\,(k\,m),\ m'\,]\ \triangleq\ k\,(m,m')$$

It pushes an existing message into an encrypted component it is concatenated with. The effect of $\mathbb{EA}$ over properties is to authenticate m' in addition to m:

$$\begin{array}{rl} A: & \mathsf{uncompromised}(k,[A,B]) \wedge (\!(k\,m,\ m')\!)_A \\ \Longrightarrow & \langle\!\langle k\,m\rangle\!\rangle_{B<} \leq \langle\!\langle k\,m,\ m'\rangle\!\rangle_{X<} < (\!(k\,m,\ m')\!)_A \end{array}$$

$$\Big\Downarrow\ \mathbb{EA}$$

$$\begin{array}{rl} A: & \mathsf{uncompromised}(k,[A,B]) \wedge (\!(k\,(m,m'))\!)_A \\ \Longrightarrow & \langle\!\langle k\,(m,m')\rangle\!\rangle_{B<} < (\!(k\,(m,m'))\!)_A \end{array}$$

Notice the relation $\langle\!\langle k\,m\rangle\!\rangle_{B<} \leq \langle\!\langle k\,m,\ m'\rangle\!\rangle_{X<}$ before applying the transformation: it says that B has originated a message containing $k\,m$, which may or may not be $k\,m,\ m'$, and that what B received may have been put together by a principal X. Recall that we ran into this issue several times while examining KD_2^3. This transformation removes this source of uncertainty.

Applying $\mathbb{EA}$ to KD_2^3 yields protocol KD_2^4, which has the following expected run:

$$\begin{tikzcd}
A & S & B \\
\circ \arrow[r, "{A,B}"] & \circ \arrow[d, "\nu k"] & \\
\circ \arrow[d] & \circ \arrow[l, "{K^{AS}\,(B,k,K^{BS}\,(A,k))}"'] & \\
\circ \arrow[rr, "{K^{BS}\,(A,k)}"] & & \circ
\end{tikzcd}$$

KD_2^4 is more formally defined by the following roles:

$$\begin{array}{lcl}
\mathsf{KD}_2^4\text{\_server}[S] & = & (A,B : A \to S)\ ;\ \nu\, k\ ; \\
 & & \langle K^{AS}\,(B,k,K^{BS}\,(A,k)) : S \to A\rangle \\
\mathsf{KD}_2^4\text{\_iclient}[A;S,B] & = & \langle A,B : A \to S\rangle\ ;\ (K^{AS}\,(B,k,M) : S \to A)\ ; \\
 & & \langle M : A \to B\rangle \\
\mathsf{KD}_2^4\text{\_rclient}[B;S] & = & (K^{BS}\,(A,k) : A \to B)
\end{array}$$

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

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

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

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

Indeed, assuming S honest and K^{BS} uncompromised, he can deduce that S did its part in the protocol, and that some principal X (not necessarily A) forwarded $K^{BS}\,(A,k)$ to him.

However, 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.

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

Note that the assumption of $\mathsf{uncompromised}(K^{AS},[A,S])$ would be irrelevant in any of B's previous inferences: only A could decrypt $K^{AS}\ (B,k,$ $K^{BS}\ (A,k))$ to forward $K^{BS}\ (A,k)$, hence accessing k. 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. Indeed, 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 [4]. We then show how Needham and Schroeder's subsequent fix to the original NSSK [13] essentially completes the application of nonce-based recency in Section 4.2. Finally, we address key confirmation as implemented in most protocols in Section 4.3.

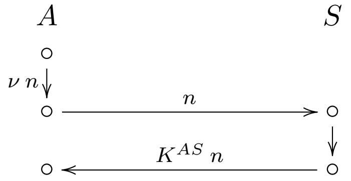
4.1 Guaranteeing Recency with Nonces

As mentioned earlier, the core key distribution protocols derived in Section 3 do not guarantee to the clients that the server has generated the key recently.

Indeed, none of the formulas we have derived for any of our clients bounds the actions of an honest server so that it follows that the key could not have been produced at an arbitrary moment in the past. Note that this is not a failure of honesty: the server may have received a fake request long before our clients felt any need to communicate; the response could have been cached by a dishonest agent, who also intercepted the clients' request and replayed that response in a timely manner.

A controllable way for a client to ensure that the key is recent is to bracket its generation between two of its own events. One approach to doing so is using the challenge-response mechanism: the client issues a fresh challenge at the time she sends the key distribution request to the server. The server cryptographically binds the response to the challenge and the response to the key distribution request. We dedicate this section to examining one of the possible concrete realizations 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.

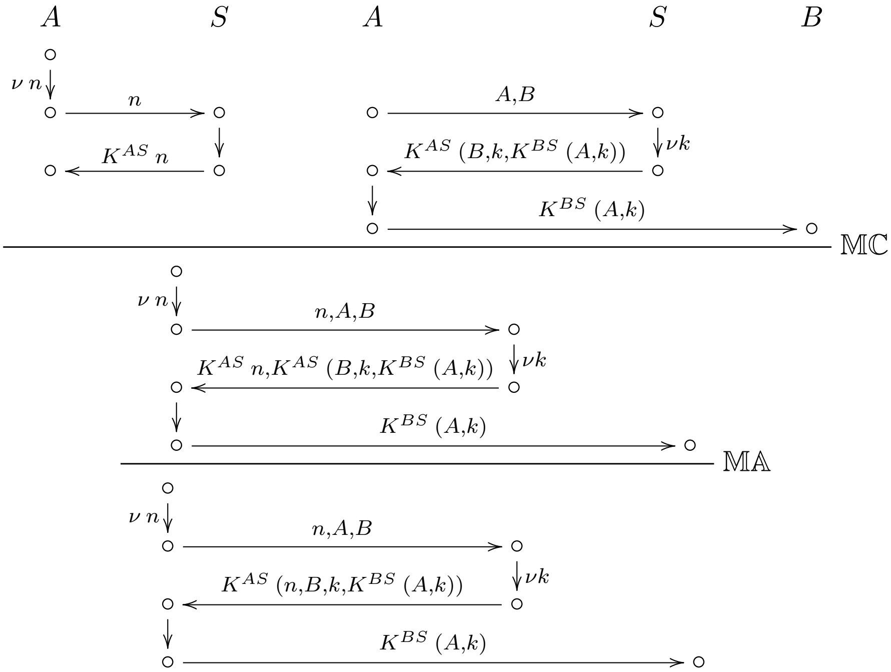
We use a specific instance of the cr axiom from Section 2.5 which sends the challenge in the clear (the challenge function is the identity) and returns the response encrypted with an uncompromised shared key: we have used it as an example in Section 2.5. As a refresher, the run of this protocol is as follows, where we write the parameters as we will use them:



The specific guarantees of this protocol are the following:

$$\begin{array}{rlllll}
A: & \mathsf{uncompromised}(K^{AS}, [A, S]) \wedge & & & & \\
 & (\nu n)_A \;<\; \langle n\rangle_A & & & & <\; (K^{AS}\, n)_A \\
\Longrightarrow & (\nu n)_A \;<\; \langle n\rangle_A & <\; (\!(n)\!)_S & <\; \langle\!\langle K^{AS}\, n\rangle\!\rangle_{S<} & & <\; (K^{AS}\, n)_A
\end{array}$$

The transformation allowing to embed a challenge-response exchange in another protocol has been extensively discussed in [10]. We present it only informally here, using protocol KD_2^4 as our case study since it is at the core of NSSK. The following diagram intuitively renders the overall effect of this transformation:


A
S
A
S
B
ν n
n
K^{AS} n
A,B
νk
K^{AS} (B,k,K^{BS} (A,k))
K^{BS} (A,k)
𝕄ℂ
ν n
n,A,B
νk
K^{AS} n,K^{AS} (B,k,K^{BS} (A,k))
K^{BS} (A,k)
𝕄𝔸
ν n
n,A,B
νk
K^{AS} (n,B,k,K^{BS} (A,k))
K^{BS} (A,k)

Intuitively, the transformation $\mathbb{MC}$ has the effect of merging two independent protocols by identifying some sends and receives between the same principals and fusing them through concatenation. Events that do not involve communication are compounded. Here, the challenge message $(n : A \to S)$ is concatenated with A's request to S $(A, B : A \to S)$ into the message $(n, A, B : A \to S)$. The two responses are processed similarly. The properties induced by this transformation are little more than what holds of the two protocols separately. Transformation $\mathbb{MA}$ consolidates the two encryptions with K^{AS} into one. It has a similar binding power to $\mathbb{EA}$ from section 3.2.

The resulting protocol includes the first three steps of NSSK (the addition of key-confirmation will complete it in Section 4.3). We formalize by presenting its roles.

$$\begin{array}{lcl}
\mathsf{NSSK_0\_server}[S] & = & (n, A, B : A \to S)\ ;\ \nu\, k\ ; \\
 & & \langle K^{AS}\ (n, B, k, K^{BS}\ (A, k)) : S \to A\rangle \\
\mathsf{NSSK_0\_iclient}[A; S, B] & = & \nu\, n\ ;\ \langle n, A, B : A \to S\rangle\ ; \\
 & & (K^{AS}\ (n, B, k, M) : S \to A)\ ;\ \langle M : A \to B\rangle \\
\mathsf{NSSK_0\_rclient}[B; S] & = & (K^{BS}\ (A, k) : A \to B)
\end{array}$$

B's role does not change at all from KD_2^4, the server's changes only marginally, while most changes occur in A's role.

It is particularly 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. The analogous property for KD_2^4 left the relation between the (actual) request and the generation of the key totally open. Thus, NSSK ensures the recency of the key to A.

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

We have dropped the last message ($\langle M\rangle_A$) since it does not influence the resulting property.

The guarantees derivable to B are however pretty 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. We use the stronger version, in which K^{AS} is assumed uncompromised.

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

Therefore, NSSK does not ensures the recency of the key to B. This is the gist of Denning and Sacco's attack on NSSK [4].

4.2 NSSK-fix

A few years after Denning and Sacco pointed out the absence of recency guarantees for the responder [4], Needham and Schroeder came forth with a "fix" for their original protocol [13]. This adjustment simply inserts an additional challenge response, between B and the server, to provide the required assurance. Minor complications are called for in order to maintain A as the initiator and avoid message confusion. We will now examine this amended protocol.

B's challenge response 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 it encrypted to the responder and expects it back also encrypted, but somehow transformed.

The expected run is as follows:

$$\begin{tikzcd}
S & B \\
 & \circ \arrow[d, "\nu\, n_B"] \\
\circ \arrow[d] & \circ \arrow[l, "K^{BS}(f(n_B))"'] \\
\circ \arrow[r, "K^{BS}(g(n_B))"] & \circ
\end{tikzcd}$$

with f and g two different message structures parameterized by n_B. The properties of this exchange, from the point of view of B are typical of a challenge-response with shared keys:

$$\begin{array}{rll}
B: & \mathsf{uncompromised}(K^{BS}, [B, S]) \wedge & \\
 & (\nu n_B)_B \; < \; \langle K^{BS}(f(n_B))\rangle_B & < \; (K^{BS}(g(n_B)))_B \\
\Longrightarrow & (\nu n_B)_B \; < \; \langle K^{BS}(f(n_B))\rangle_B \; < & \\
 & < \; (\!(K^{BS}(f(n_B)))\!)_S \; < \; \langle\!\langle K^{BS}(g(n_B))\rangle\!\rangle_{S<} & < \; (K^{BS}(g(n_B)))_B
\end{array}$$

The proof is similar to what we saw in Section 2.5.

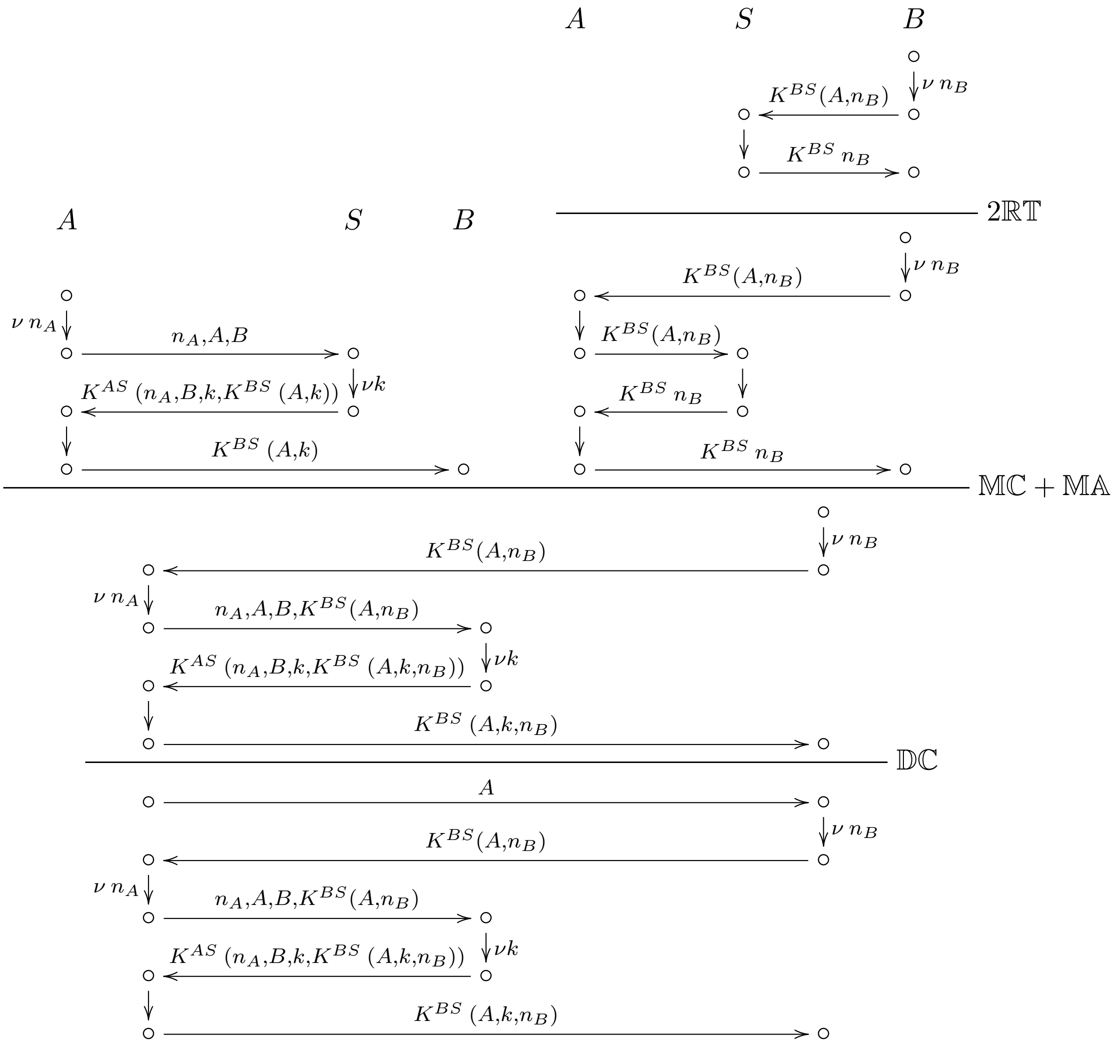
The specific instance used in NSSK-fix takes $f(n_B) \triangleq (A, n_B)$ and $g(n_B) \triangleq (n_B)$, although any functions would do, as long as they are not identical and they truly depend on n_B. NSSK-fix itself is obtained by applying a series of transformations to NSSK and this challenge-response exchange:

- Two applications of the routing transformation $\mathbb{RT}$ modify the challenge-response so that B and S communicate through A.

- Similarly to Section 4.1, transformations $\mathbb{MC}$ and $\mathbb{MA}$ merge this modified challenge-response and NSSK_0, and cryptographically bind B's nonce within the the key distribution submessage S intends for B.

- Finally, transformation $\mathbb{DC}$ discharges A from B's roles, allowing that principal to remain the initiator of the final protocol.

The overall transformation is summarized in the following diagram:



Observe that the resulting protocol is substantially more complex than NSSK_0 (in the upper left corner): it contains two additional steps and one more cryptographic operation. Note that it may be rather complicated to extend this protocol to an n-party key distribution.

This protocol differs from NSSK-fix only by the absence of the final key-confirmation steps. They will be added in Section 4.3. Its roles are given next.

$$\begin{array}{lcl}
\mathsf{NSSKfix_0\_server}[S] & = & (n_A, A, B, K^{BS}(A, n_B) : A \rightarrow S)\ ;\ \nu\, k\ ; \\
 & & \langle K^{AS}\,(n_A, B, k, K^{BS}\,(A, k, n_B)) : S \rightarrow A\rangle \\
\mathsf{NSSKfix_0\_iclient}[A; S, B] & = & \langle A : A \rightarrow B\rangle\ ;\ (M' : B \rightarrow A)\ ; \\
 & & \nu\, n_A\ ;\ \langle n_A, A, B, M' : A \rightarrow S\rangle\ ; \\
 & & (K^{AS}\,(n_A, B, k, M) : S \rightarrow A)\ ;\ \langle M : A \rightarrow B\rangle \\
\mathsf{NSSKfix_0\_rclient}[B; S] & = & (A : A \rightarrow B)\ ;\ \nu\, n_B\ ;\ \langle K^{BS}(A, n_B) : B \rightarrow A\rangle\ ; \\
 & & (K^{BS}\,(A, k, n_B) : A \rightarrow B)
\end{array}$$

We now turn to the properties that each principal can derive. A's deduction differ from $\mathsf{NSSK_0}$ only by the presence of her two extra actions, and by the fact that an honest server will correctly interpret the added fields, both in her request and in its response. In particular, A is perfectly aware that the component she forwards to B in her last message (omitted below) has the structure $K^{BS}\,(A, k, n_B)$ for some value n_B. The logical statement is as follows:

$$\begin{array}{ll}
A: & \mathsf{uncompromised}(K^{AS}, [A, S])\ \wedge\ \mathsf{honest}\ S\ \wedge \\
 & \quad \langle A\rangle_A\ <\ (M')_A\ <\ (\nu\, n_A)_A\ <\ \langle n_A, A, B, M'\rangle_A\ < \\
 & \quad <\ (K^{AS}\,(n_A, B, k, M))_A \\
\Longrightarrow & \quad \langle A\rangle_A\ <\ (M')_A\ <\ (\nu\, n_A)_A\ <\ \langle n_A, A, B, M'\rangle_A\ < \\
 & \quad <\ (n_A, A, B, K^{BS}(A, n_B))_S\ <\ (\nu\, k)_S\ < \\
 & \quad <\ \langle K^{AS}\,(n_A, B, k, K^{BS}\,(A, k, n_B))\rangle_{S<}\ < \\
 & \qquad\qquad\qquad <\ (K^{AS}\,(n_A, B, k, K^{BS}\,(A, k, n_B))_A
\end{array}$$

Since A now supposedly receives a message from B, it makes sense to ask what would be the effect of strengthening the assumptions of this property with $\mathsf{uncompromised}(K^{BS}, [B, S])$. This brings no advantage since A simply forwards B's first message and has no way to inspect or verify its contents, even indirectly. The additional assumption that B is honest brings some marginal additional insight, namely, that B performed its initial three actions (with the right parameters) before S started processing, but she has no way of ordering these added events with respect to her own initial actions.

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.

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

As in NSSK_0, dropping the assumption that K^{AS} is uncompromised simply 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 in with the recency guarantees of NSSK(-fix). The remaining issue to address is ensuring to both recipients that their counterpart also knows the new shared key. As we observed, under assumptions, these protocols already guarantee this to B, but A has no means to be sure that B ever learned k.

In order to make this concept more explicit, we define the predicate $\mathsf{has}(A,m)$ that holds only if principal A has seen term m. Intuitively, this is the case whenever we know that A has performed an action on m. Here is a partial definition, incomplete but sufficient for our needs. It could clearly be extended with additional cases.

$$\mathsf{has}(A,m) \triangleq \begin{cases} (x/K\,(m,m'))_A \\ \langle\!\langle K\,(m,m')\rangle\!\rangle_{A<} \\ (x/m\,m')_A \\ \langle\!\langle m\,m'\rangle\!\rangle_{A<} \end{cases}$$

The first two cases describe situations where m is encrypted with a shared key K. In the first line, A decrypt a message containing m, thus exposing this term, in the second she builds such a term for export. The last two cases are similar, except that m is the key itself. Note that in all cases, the action is known to have been performed by A. For this reason, there is no need to assume the key to be uncompromised. It is a simple exercise to verify that the proposition $\mathsf{has}(X,k)$ holds in exactly the three situations below, with respect to all the formulas we have derived in this paper:

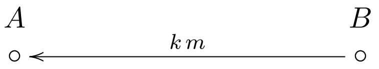
1 $X = S$, i.e., the server S who generated k knows k.

2 X is the observer of the formula, obviously.

3 $X = A$, the observer is B, the key distribution protocol is a descendent of KD_2^4 (i.e., S sends k to B cryptographically embedded in the message for A), and the key K^{AS} is assumed to be uncompromised.

In particular, it has never been the case that $\mathsf{has}(B, k)$ from the point of view of A.

Since k is now a shared secret between A and B (supposedly), the easiest way to provide the missing guarantee is for B to send A a pre-agreed message encrypted with k. Consider the following protocol fragment:



where m is arbitrary. The two simple roles are as follows:

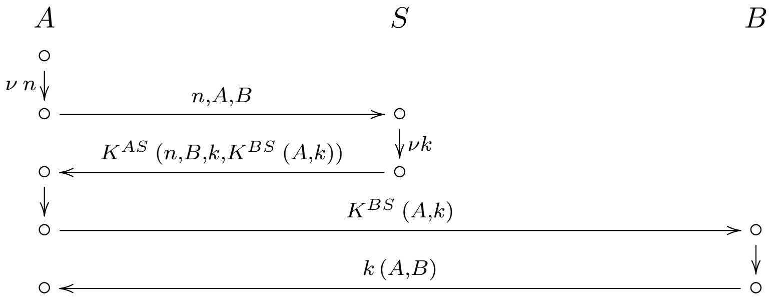
$$\begin{aligned} \mathsf{enc\_to}[A; B, k, m] &= (k, m : B \to A) \\ \mathsf{enc\_from}[B; A, k, m] &= \langle k, m : B \to A \rangle \end{aligned}$$

B is unable to infer anything interesting from his observations since he never receives anything from A. On the other hand, under the assumption that k is uncompromised, A deduces that it is B who sent this message:

$$\begin{array}{rrcl} A: & \mathsf{uncompromised}(k, [A, B]) & & (k\, m)_A \\ \Longrightarrow & \langle k\, m \rangle_{B<} & < & (k\, m)_A \end{array}$$

Notice that, in this formula, the proposition $\mathsf{has}(A, k)$ now holds.

At this point, we can simply use the extending transformation $\mathbb{XT}$ (which simply adds an action at the end of a protocol) and, by a number of applications of the discharging transformation $\mathbb{DC}$, we can augment NSSK_0 and $\mathsf{NSSKfix}_0$ with a send action intended for B to confirm to A that he knows the key. The message m can be arbitrary, for example A, B. The resulting run in the case of (the shorter) NSSK is as follows:



Let us call this protocol NSSK_1. A's observations lead her to conclude:

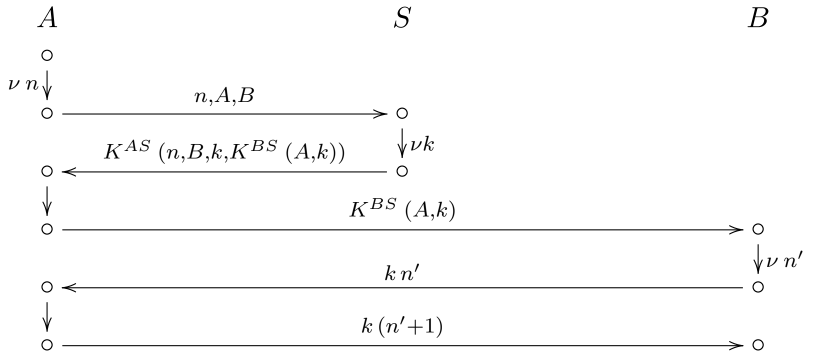
$$
\begin{aligned}
A:\ & \mathsf{uncompromised}(K^{AS},[A,S]) \ \wedge\ \mathsf{honest}\ S \ \wedge \\
& \boxed{\mathsf{uncompromised}(k,[A,B])} \ \wedge \\
& \quad (\nu\, n)_A \ <\ \langle n,A,B\rangle_A \ <\ (K^{AS}\,(n,B,k,M))_A \ < \\
& \quad <\ \boxed{\langle M\rangle_A \ <\ (k\,(A,B))_A} \\
\Longrightarrow\ & \quad (\nu\, n)_A \ <\ \langle n,A,B\rangle_A \ < \\
& \qquad <\ (n,A,B)_S \ <\ (\nu\, k)_S \ <\ \langle K^{AS}\,(n,B,k,K^{BS}\,(A,k))\rangle_{S<} \ < \\
& \quad <\ (K^{AS}\,(n,B,k,K^{BS}\,(A,k))_A \ < \\
& \quad <\ \boxed{\langle K^{BS}\,(A,k)\rangle_A \ <\ (K^{BS}\,(A,k))_B \ <\ \langle k\,(A,B)\rangle_{B<} \ <\ (k\,(A,B))_A}
\end{aligned}
$$

We have highlighted the additions with respect to NSSK_0 (see Section 4.1) by enclosing them in boxes. Recall that 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 $k\,(A,B)$. It is easy to verify that within this formula, $\mathsf{has}(B,k)$ holds, which achieves our goal.

The last addition, $\mathsf{uncompromised}(k,[A,B])$, 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 closed-world assumption, nor the induction principles to reason about it. Deriving that k must indeed be secret would rely on such devices. We intend to develop the secrecy facet of this logic in future work. The assumption $\mathsf{uncompromised}(k,[A,B])$ is an interface to this future extension.

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 expect this same nonce back from A, transformed in a predictable way. We will now analyze 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). It does make the hypothesis that K^{AS} was uncompromised (which ultimately was the reason why B could conclude that A had knowledge of k) unnecessary, but the gain is rather slim: a compromised K^{AS} immediately allows compromising k. These two propositions are however distinct in our logic since we never derive an uncompromised fact.

It should however be observed that, from the point of view of B, the last two messages NSSK implement a challenge-response exchange: B generates the nonce n', sends it to A encrypted (with k), and expected it back from her transformed. By doing so, B ascertains that A in indeed 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. If B's challenges include a request for a service (e.g., retrieving a file) and A's responses embed an outcome for this service (e.g., the file itself, or an error message), this protocol implements a crude (and rather lopsided) single-authentication, repeated-request client-server mechanism: NSSK_0 realizes the initial authentication and key distribution, the added challenge-response forms the basis of each instance of a subsequent client-server exchange, protected by the key obtained in the first phase. This interpretation of NSSK is clearly not realistic since it implies that the service provider (A) initiates the exchange while the client (B) just gets to issues the requests for service. However, we will see in Section 5 that a nearly identical mechanism is used in Kerberos to support repeated service requests based on a single initial key distribution.

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. How they stand now, both NSSK and NSSK-fix have a flavor of repeated client-server protocols with the initiator and responder roles inverted.

5. Derivations of Kerberos

Kerberos is a complex and versatile protocol that has been the subject of intense scrutiny over the years [14, 15]. 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 apply it to the derivation of 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.[10]

We formalize this intuition as a transformation $\mathbb{TS}$. We define it by describing how it operates on a process P, how it consequently alters the representation of the honesty of the participants, and how their knowledge gets upgraded.

Roles. Given a roles ρ and ρ' embedding the sending and receiving of $K^{AS}\,m$, respectively, the transformation $\mathbb{TS}$ is described as follows:

$$\begin{cases} \mathbb{TS}[\,\langle\!\langle K^{AS}\,m : S \to A\rangle\!\rangle\,] & = \quad (\tau\, t)\ ;\ \langle\!\langle K^{AS}(m,t) : S \to A\rangle\!\rangle \\ \mathbb{TS}[\,(\!(K^{AS}\,m : S \to A)\!)\,] & = \quad (\!(K^{AS}(m,t) : S \to A)\!) \end{cases}$$

Recall that the event $(\tau\ t)$ represents S's looking up of his current local time and instantiating t to it.

[10] This assessment takes into considerations clock skews between hosts, typical network delays, etc.

Honesty. The honesty formula of both principals is derived from the transformed process. In particular S's honesty formula is updated as follows:

$$\cdots \prec \langle\!\langle K^{AS}\, m : S \to A \rangle\!\rangle_{S<} \prec \cdots$$

$$\Big\Downarrow \mathbb{TS}(P)$$

$$\cdots \prec (\tau\, t)_S \prec \langle\!\langle K^{AS}(m,t) : S \to A \rangle\!\rangle_{S<} \prec \cdots$$

A's honesty is updated similarly (but it will not play any role in the sequel).

Knowledge. More interesting is the description of how $\mathbb{TS}$ alters the guarantees that each principal can deduce. Given the particular format of this transformation (S does not receive a message back), we concentrate on the knowledge accessible to A.

In the interest of space, we elide the source and destination directives.

$$\begin{array}{ll} A: & \mathsf{uncompromised}(k,[A,B]) \;\wedge\; (\!(K^{AS}\, m)\!)_A \\ \Longrightarrow & \langle\!\langle K^{AS}\, m \rangle\!\rangle_{S<} \;<\; (\!(K^{AS}\, m)\!)_A \end{array}$$

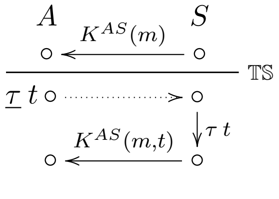
$$\Big\Downarrow \mathbb{TS}(P)$$

$$\begin{array}{ll} A: & \mathsf{uncompromised}(k,[A,B]) \;\wedge\; \mathsf{honest}\ S \;\wedge\; (\!(K^{AS}(m,t))\!)_A \\ \Longrightarrow & (\underline{\tau}\, t)_A \;<\; (\tau\, t)_S \;<\; \langle\!\langle K^{AS}(m,t) \rangle\!\rangle_{S<} \;<\; (\!(K^{AS}(m,t))\!)_A \end{array}$$

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 the *pseudo-event* $(\underline{\tau}\ t)$ represents the earliest point in A's local time where she will accept the time t as valid i.e., "recent enough" in our context. Notice that 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 from its clock.

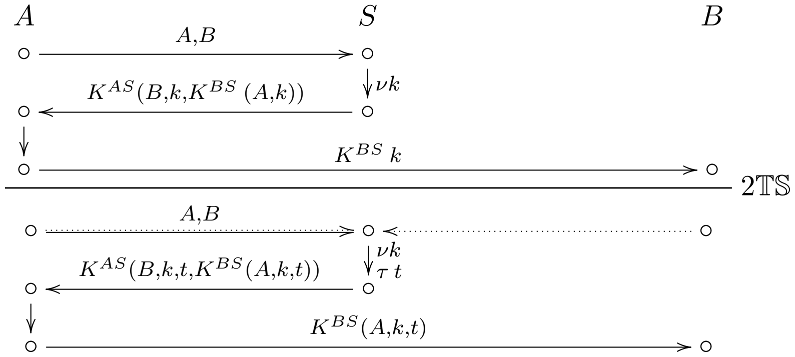
We obtain this formula by homomorphically replacing $K^{AS}m$ with $K^{AS}(m,t)$ in the derivation of the top formula. The atom $(\tau\, t)_S$ comes from the upgraded honesty axiom. The token $(\underline{\tau}\ t)_A$ represents A's acceptance of the validity of t.

We schematically represent this transformation by the inference rule at right. The dotted arrow links the pseudo-event $(\underline{\tau})$ to the beginning of the protocol in A's view. This transformation is closely related to $\mathbb{CA}$ from Section 3.2.



5.2 The Denning-Sacco Protocol

The Denning-Sacco protocol [4] applies the transformation $\mathbb{TS}$ just described 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 transformation 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). This derivation is summarized as follows:



The Denning-Sacco is therefore characterized by the following roles:

$$\begin{array}{lcl}
\mathsf{NS\_server}[S] & = & (A,B : A \to S)\ ;\ (\nu\, k\ \otimes\ \tau\, t)\ ; \\
 & & \langle K^{AS}\ (B,k,t,K^{BS}\ (A,k,t)) : S \to A\rangle \\
\mathsf{NS\_iclient}[A;S,B] & = & \langle A,B : A \to S\rangle\ ;\ (K^{AS}\ (B,k,t,M) : S \to A)\ ; \\
 & & \langle M : A \to B\rangle \\
\mathsf{NS\_rclient}[B;S] & = & (K^{BS}\ (A,k,t) : A \to B)
\end{array}$$

The verification of the timestamp t occurs in the implicit match. This is from this operation that the pseudo-events $\underline{\tau}\ t$ stem.

As usual, we summarize next the information gained by each principal as she reaches the end of her run. 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:

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

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.

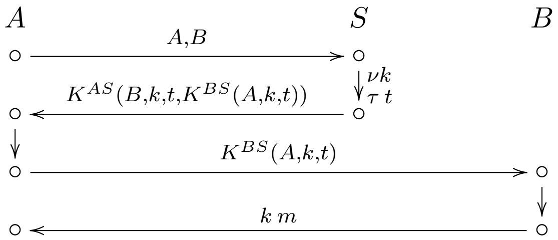
$$\begin{array}{rl} B: & \mathsf{honest}\ S\ \wedge\ \mathsf{uncompromised}(K^{BS},[B,S])\ \wedge \\ & \wedge\ \mathsf{uncompromised}(K^{AS},[A,S])\quad \wedge\ (K^{BS}(A,k,t))_B \\ \Longrightarrow & \begin{bmatrix}(A,B)_S\\(\underline{\tau}\,t)_B\end{bmatrix} < \begin{bmatrix}(\nu\,k)_S\\(\tau\,t)_S\end{bmatrix} <\ \langle K^{AS}(B,k,t,K^{BS}(A,k,t))\rangle_{S<}\ < \\ & <\ \langle K^{BS}(A,k,t)\rangle_{A<} \qquad\qquad\qquad <\ (K^{BS}(A,k,t))_B \end{array}$$

Denning and Sacco prominently pointed out in their original paper [4] that this protocol provides full recency guarantees with a minimum number of messages.

5.3 Kerberos 4

We will now see that 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. In Section 5.2, we observed that, by the protocol's end, B is able to determine that A knows the distributed key k, but that A has no such certainty. In our first step, we simply use the transformation $\mathbb{XT}$ from Section 4.3 in order for B to acknowledge the receipt of A's last transmission by sending her some (recognizable) message m encrypted with k. The resulting run is as follows:



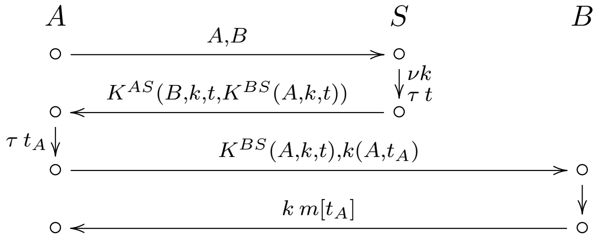
The corresponding protocol is a simple extension of DS.

As in the case of NSSK_1, A's knowledge is extended with the certainty that B has seen (actually used) k, under the assumption that the master keys are not compromised. The following formula makes this intuition precise:

$$\begin{aligned}
A:\ & \mathsf{uncompromised}(K^{AS},[A,S]) \ \wedge\ \mathsf{honest}\ S \wedge \\
& \mathsf{uncompromised}(k,[A,B]) \wedge \\
& \quad \langle A,B\rangle_A \ <\ (K^{AS}\,(B,k,t,M))_A \ <\ \langle M\rangle_A \ <\ (k\,m)_A \\
\Longrightarrow\ & \begin{bmatrix}(A,B)_S \\ (\underline{\tau}\,t)_B\end{bmatrix} < \begin{bmatrix}(\nu\,k)_S \\ (\tau\,t)_S\end{bmatrix} < \ \langle K^{AS}(B,k,t,K^{BS}(A,k,t))\rangle_{S<} \ < \\
& \quad <\ \langle K^{BS}\,(A,k,t)\rangle_A \ <\ (K^{BS}\,(A,k,t))_B \ <\ \langle k\,m\rangle_{B<} \ <\ (k\,m)_A
\end{aligned}$$

The other remarks about NSSK_1 and $\mathsf{NSSKfix}_1$ from Section 4.3 hold here as well.

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). Technically, this protocol is obtained by first extending the third message with the token $k\ A$ (which is completely redundant at this point) using transformation $\mathbb{MC}$ and then applying the transformation $\mathbb{TS}$ to it, and possibly pushing t_A into m. Note that if t_A is indeed returned in the last message, this extension can be seen as a timestamp-based challenge-response.

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.

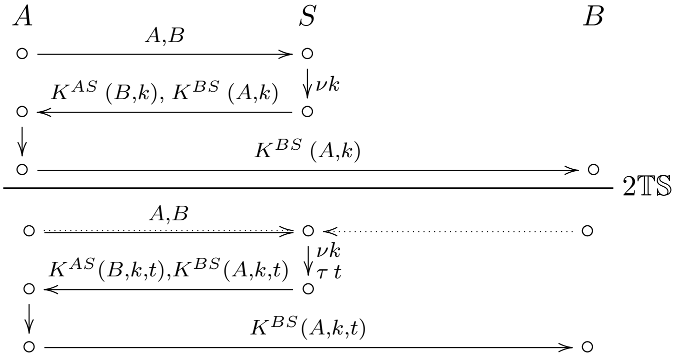
Kerberos 4 [14] extends this core protocol with numerous fields primarily meant to negotiate parameters of the resulting authentication: added timestamps, options and flags, access control information, etc. For maximum flexibility, Kerberos chains two instances of the core protocol, by which a client

(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^4_2 , Kerberos 5 starts with the concatenated variant KD^3_2. Given this different starting point, the core protocol is however derived by applying the exact same steps as in Kerberos 4. It is interesting to examine them as the conclusions available to the various principals are not the same throughout.

The derivation of the analogous of the Denning-Sacco protocol is summarized as follows:



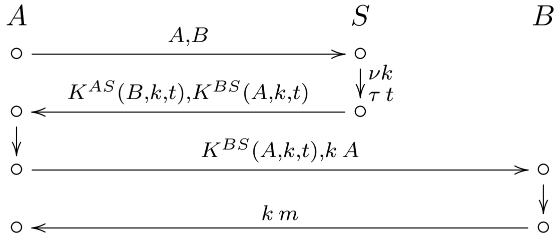
The knowledge derivable by A is similar to the Denning-Sacco protocol, except that she can never be certain that the encrypted component she receives corresponds to what S sent.

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

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 $\mathsf{uncompromised}(K^{AS},[A,S])$ 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.

$$\begin{array}{ll} B: & \textsf{honest}\ S\ \wedge\ \textsf{uncompromised}(K^{BS},[B,S])\ \wedge\ (K^{BS}(A,k,t))_B \\ \Longrightarrow & \begin{bmatrix} \langle A,B\rangle_A < (A,B)_S \\ (\underline{\tau}\,t)_B \end{bmatrix} < \begin{bmatrix} (\nu\,k)_S \\ (\tau\,t)_S \end{bmatrix} < \\ & <\ \langle K^{AS}(B,k,t),K^{BS}(A,k,t)\rangle_{S<}\ <\ \langle K^{BS}(A,k,t)\rangle_X\ < \\ & <\ (K^{BS}(A,k,t))_B \end{array}$$

Adding key confirmation. 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 $k\ A$ (any message encrypted with k will do, but this happens to be the core of the Kerberos authenticator) 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 $K^{AS}(B,k,t)$. As we saw in Section 4.1, certain nonce-based challenge-response exchanges are alternative mechanisms for ensuring the recency of an action. They do not rely on loosely synchronized clocks, but generally involve communication overhead (this is why B's recency guarantees do not rely on nonces).

6. Conclusions

Appendix: Relays and the equivalence of runs

Two processes should be considered indistinguishable if they have the same executable runs.[11] But for processes that run on a network with routers and relays, a run where $(x : A \to B)_B$ and $\langle t : A \to B\rangle_A$ interact directly, i.e. $\sqrt{(x : A \to B)_B} = \langle t : A \to B\rangle_A$ is indistinguishable from the runs where of these two actions interact through any number of relays in the form $(x : Y \to Z)_C ; \langle x : Y \to Z\rangle_C$, so that $\sqrt{(x : A \to B)_B} = \langle x : Y \to Z\rangle_C$ and $\sqrt{(x : Y \to Z)_C} = \langle t : A \to B\rangle_A$.

The consequence of this is that the process

$$\begin{pmatrix} \langle t : S \to A\rangle_S \\ \otimes \\ \langle u : S \to B\rangle_S \end{pmatrix} ; \begin{pmatrix} (x : S \to A)_A \\ \otimes \\ (x : S \to B)_B \end{pmatrix}$$

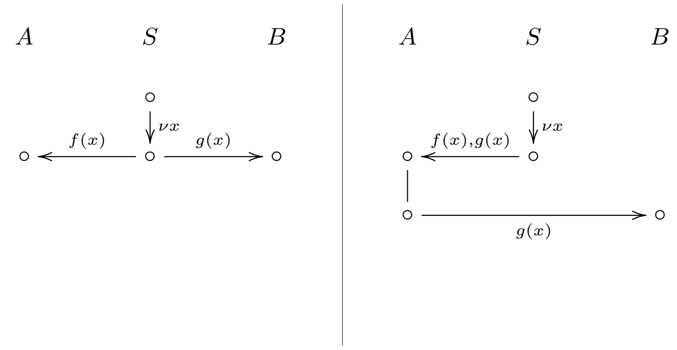
can be reasonably viewed as equivalent with

$$\begin{pmatrix} \langle t : S \to A\rangle_S \\ \otimes \\ \langle u : S \to B\rangle_S \end{pmatrix} ; \begin{pmatrix} (x : S \to A)_A \ ; \ (y : S \to B)_A \ ; \ \langle y : S \to B\rangle_A \\ \otimes \\ (x : S \to B)_B \end{pmatrix}$$

By bundling the two interactions between S and A, , we get the process

$$\langle t, u : S \to A\rangle_S \ ; \begin{pmatrix} (x, y : S \to A)_A \ ; \ \langle y : S \to B\rangle_A \\ \otimes \\ (x : S \to B)_B \end{pmatrix}$$

which is still equivalent with the ones above, but one interaction has been moved from S to A. This explains the transformation



References

[1] M. Abadi, M. Burrows, B. Lampson, and G. Plotkin. A calculus for access control in distributed systems. *ACM Transactions on Programming Languages and Systems*, 21(4):706–734, September 1993.

[2] F. Butler, I. Cervesato, A. D. Jaggard, and A. Scedrov. A Formal Analysis of Some Properties of Kerberos 5 using MSR. In *Proc. of the 15th IEEE Computer Security Foundations Workshop — CSFW-02*, pages 175–190. IEEE Computer Society Press, 24-26 June 2002.

[11] A finer equivalence would also require that their non-executable runs fail in the same ways. The whole linear-time branching-time spectrum of concurrent systems opens up.

[3] 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.

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

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

[6] 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.

[7] J. T. Fabrega, J. Herzog, and J. Guttman. Strand spaces.

[8] B. Lampson, M. Abadi, M. Burrows, and E. Wobber. Authentication in distributed systems: theory and practice. *ACM Trans. on Comput. Syst.*, 10(4):265–310, November 1992.

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

[10] C. Meadows and D. Pavlovic. Deriving, attacking and defending the gdoi protocol. In *Computer Security — ESORICS 2004*, volume 3193 of *Lecture Notes in Computer Science*, pages 33–53, Sophia-Antipolis, France, September 2004. Springer-Verlag.

[11] A. C. Myers and B. Liskov. Protecting privacy using the decentralized label model. *ACM Transactions on Software Engineering and Methodology*, 9(4):410–442, October 2000.

[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, December 1978.

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

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

[16] S. Schneider. Verifying Authentication Protocols in CSP. *IEEE Transactions on Software Engineering*, 24(9):741–758, 1998.

[17] J. G. Steiner, C. Neuman, and J. I. Schiller. Kerberos: An Authentication Service for Open Network Systems. In *Proceedings of the Winter 1988 Usenix Conference*, 1998.

[18] F. J. Thayer Fábrega, J. Herzog, and J. D. Guttman. Honest ideals on strand spaces. In *Proceedings, 1998 Computer Security Foundations Workshop*, 1998.

Contents (temporary –ic)