PROCESS ALGEBRAIC ANALYSIS OF CRYPTOGRAPHIC PROTOCOLS

Michele Boreale Rocco De Nicola Rosario Pugliese
Dipartimento di Sistemi e Informatica, Università di Firenze
{boreale,denicola,pugliese}@dsi.unifi.it

Abstract Recent approaches to the analysis of crypto-protocols build on concepts which are well-established in the field of process algebras, such as labelled transition systems (lts) and observational semantics. We outline some recent work in this direction that stems from using cryptographic versions of the pi-calculus – most notably Abadi and Gordon's *spi-calculus* – as protocol description languages. We show the impact of these approaches on a specific example, a simplified version of the Kerberos protocol.

Keywords: Process calculi, Reasoning about security, Protocol verification, Semantics, Formal methods

1. INTRODUCTION

Recent approaches to the analysis of crypto-protocols build on a few concepts which are well-established in the field of process algebras, such as labelled transition systems (lts) and observational semantics. We outline here some recent work in this direction that stems from the use of cryptographic versions of the pi-calculus [10] – most notably Abadi and Gordon's *spi-calculus* [2, 1] – as protocol description languages. We show the impact of these approaches on a specific example, a simplified version of the Kerberos protocol [8].

A major line of research is centered around the notion of observational equivalence, which permits making such notions as 'attacker' and 'secrecy' rigorous. For instance, according to [2], a way of asserting that a protocol, represented by a process term $P(d)$, keeps datum d secret is requiring that $P(d)$ be *equivalent* to $P(d')$, for every other d'. Observational equivalences based on context-closure, like *may-testing* [6, 3, 2] and *barbed equivalence* [11], appear to be appropriate in this setting. The intuition behind them is precisely that no external context (which in the present setting can be read as 'attacker') may notice any difference when running in parallel with $P(d')$ or $P(d)$. The definitions of these equivalences, while rigorous and intuitive, suffer from universal quantification over contexts (attackers), that makes equivalence checking very hard. We have shown in previous work [4] how to avoid such quantification and obtain more tractable characterizations. The latters are based on an 'environment-sensitive' lts (as opposed to the 'standard' lts, which only explains process intentions). The states of this lts are *configurations* $\sigma \triangleright P$, where P is a process and σ represents the current environment's knowledge about names and keys. The moves of the process are constrained by this knowledge. Trace equivalence and weak bisimilarity are defined on top of this lts, and proven to be in agreement with may-testing and barbed equivalence, respectively. A major advantage of the new equivalences is represented by the congruence laws that make compositional reasoning on protocols possible. In this paper, we use bisimilarity and related compositional laws to formalize and verify a few security properties of a key-exchange protocol, inspired by the simplified version of Kerberos in [5]. The verification implies establishing a few process equalities that, we believe, would be hard to prove if one relied on the original, contextual definitions.

Another line of research based on environment-sensitive lts' relies on the analysis of the traces a protocol can execute, to find out whether any 'insecure' state (one, e.g., in which some confidential information has been passed to the environment) can ever be reached. The resulting method is less amenable to compositional reasoning, but seems well-suited for mechanization. We hint at some ongoing work (by one of us in collaboration with Martin Abadi and Marcelo Fiore), indicating that a full automation of the method can be achieved by considering a *symbolic* version of the environment-sensitive lts.

The rest of the paper is organized as follows. Section 2 introduces the language, the environment-sensitive lts and bisimilarity. Section 3 presents a number of properties of bisimilarity, mainly compositional proof rules. These rules are exploited in Section 4, to analyze a simplified version of Kerberos. The trace analysis of the protocol, based on a symbolic version of the lts, is touched upon in Section 5. A few concluding remarks are reported in Section 6.

2. THE LANGUAGE

In this section, we introduce syntax, operational semantics and bisimilarity for our language.

Syntax

The syntax of the calculus is summarized in Table 1. *Names* can be used as communication channels, primitive data or encryption keys: we do not distinguish between these three kinds of objects (notationally, we prefer letters $h, k, \dots$, when we want to stress the use of a name as a key). *Messages* are obtained from names either via pairing or via shared-key encryption. In particular, message $\{M\}_k$ represents the ciphertext obtained by encrypting message M under key k, using a shared-key encryption system. Encryptions can be arbitrarily nested. *Expressions* are obtained applying encryption, decryption, pairing and projection operators to names and ciphertexts. For example, the result of evaluating $\mathtt{dec}_\eta(\zeta)$ is the text obtained by decrypting the ciphertext ζ using the value of η as a key. Logical *formulae* generalize the usual equality operator of the π-calculus with a predicate $name(\cdot)$, which tests for the format of the argument (plain name or a compound ciphertext), and with a 'let' construct that binds the value of some expression ζ to a name z. *Processes* are built using a set of operators which include those from the standard π-calculus, plus two new operators: boolean guard and encryption/decryption.

An informal explanation of the operators might be the following: $\mathbf{0}$ is the process that does nothing; $\eta(x).P$ represents input of a generic message x along η: the only useful case is when η is a name, otherwise the whole process is stuck; $\overline{\eta}\zeta.P$ represents output of ζ along η: the only useful case is when η is a name and ζ is a message, otherwise the whole process is stuck; $P + Q$ can behave either as P or Q: the choice might either be triggered by the environment, or by internal computations of P or Q; $P \,|\, Q$ is the parallel execution of P and Q; $(\nu\, a)\, P$ creates a new name a which is only known to P; $!\,P$ behaves like unboudedly many copies of P running in parallel, i.e. $P \mid P \mid P \mid \cdots$; ϕP behaves like P if the formula ϕ is logically true, otherwise it is stuck; $\mathtt{let}\ z = \zeta\ \mathtt{in}\ P$ attempts evaluation of ζ: if the evaluation succeeds the result is bound to z within P, otherwise the whole process is stuck.

We shall often abbreviate $(\nu\, a)\,(\nu\, b)\, P$ as $(\nu\, a, b)\, P$. Notions of *free names* of a process P, $fn(P)$, of *bound names* of P, $bn(P)$, and of alpha-equivalence arise as expected; $n(P)$ is $fn(P) \cup bn(P)$. Often, we shall write $fn(P, Q)$ in place of $fn(P) \cup fn(Q)$ (similarly for $bn(\cdot)$ and $n(\cdot)$). Similar notations are used for formulae, expressions and messages. From now on, *we shall identify alpha-equivalent terms.*

A *substitution* σ is a finite partial map from $\mathcal{N}$ to the set of messages $\mathcal{M}$. The domain and proper co-domain of σ are written $dom(\sigma)$ and $range(\sigma)$, respectively. We let $n(\sigma) = dom(\sigma) \cup (\cup_{M \in range(\sigma)} n(M))$.

Table 1 Syntax of the calculus

$a, b \ldots, h, k, \ldots, x, y, z \ldots$			*names* $\mathcal{N}$
M, N	$::=$	$a \mid \langle M_1, M_2\rangle \mid \{M\}_k$	*messages* $\mathcal{M}$
η, ζ	$::=$	$a \mid \{\eta\}_\zeta \mid \mathtt{dec}_\eta(\zeta)$ $\mid \langle \zeta_1, \zeta_2\rangle \mid \pi_1(\zeta) \mid \pi_2(\zeta)$	*expressions* $\mathcal{Z}$
ϕ, ψ	$::=$	$tt \mid name(\zeta) \mid [\zeta = \eta]$ $\mid \mathtt{let}\ z = \zeta\ \mathtt{in}\ \phi \mid \phi \wedge \psi \mid \neg\phi$	*formulae* Φ
P, Q	$::=$		*processes* $\mathcal{P}$
		$\mathbf{0}$	(*null*)
	$\mid$	$\eta(x).\, P$	(*input prefix*)
	$\mid$	$\overline{\eta}\zeta.\, P$	(*output prefix*)
	$\mid$	$P + Q$	(*non − deterministic choice*)
	$\mid$	$P \mid Q$	(*parallel composition*)
	$\mid$	$(\nu\, a)\, P$	(*restriction*)
	$\mid$	$!\, P$	(*replication*)
	$\mid$	ϕP	(*boolean guard*)
	$\mid$	$\mathtt{let}\ z = \zeta\ \mathtt{in}\ P$	(*encryption/decryption*)

It is assumed that $\mathtt{dec}.(\cdot)$ does not occur in $name(\zeta)$, $[\zeta = \eta]$, $\eta(x).$ and $\overline{\eta}\zeta$. Operators $a(x).\cdot$, $(\nu\, a)\cdot$ and $\mathtt{let}\ z = \zeta\ \mathtt{in}\ \cdot$ are *binders*, with the obvious scope, for names x, a and z, respectively. In $\mathtt{let}\ z = \zeta\ \mathtt{in}\ \cdot$, it is assumed that $z \notin n(\zeta)$.

Given a tuple of distinct names $\widetilde{x} = (x_1, \ldots, x_n)$ and a tuple of messages $\widetilde{M} = (M_1, \ldots, M_n)$, the substitution mapping each x_i to M_i will be sometimes written as $[\widetilde{M}/\widetilde{x}]$ or $[{M_i}/{x_i}]_{i\in 1..n}$. When $\widetilde{x} \cap dom(\sigma) = \emptyset$ (if convenient, we regard a tuple as the set of its components), we write $\sigma[\widetilde{M}/\widetilde{x}]$ for the substitution σ' which is the union of σ and $[\widetilde{M}/\widetilde{x}]$ (in this case we say that σ' *extends* σ). For a given $V \subseteq_{\mathrm{fin}} \mathcal{N}$, we write ϵ_V for the substitution with $dom(\epsilon_V) = V$ that acts as the identity on V. For any term (name/expression/formula/process) t, $t\sigma$ denotes the term obtained by simultaneously replacing each $x \in fn(t) \cap dom(\sigma)$ with $\sigma(x)$, with renaming of bound names of t possibly involved to avoid captures.

Operational semantics

The operational semantics makes use of two evaluation functions: one for expressions, the other for formulae. The *evaluation function for expressions*, $\widehat{\cdot} : \mathcal{Z} \to \mathcal{M} \cup \{\bot\}$ (where $\bot$ is a distinct symbol), is defined by induction on ζ as follows:

- $\widehat{a} = a$

- $\widehat{\{\zeta_1\}_{\zeta_2}} = \begin{cases} \{M\}_k & \text{if } \widehat{\zeta_1} = M \text{ and } \widehat{\zeta_2} = k, \text{ for some } M \text{ and } k \in \mathcal{N} \\ \bot & \text{otherwise} \end{cases}$

- $\widehat{\mathtt{dec}_{\zeta_2}(\zeta_1)} = \begin{cases} M & \text{if } \widehat{\zeta_1} = \{M\}_k \text{ and } \widehat{\zeta_2} = k, \text{ for some } M \text{ and } k \in \mathcal{N} \\ \bot & \text{otherwise} \end{cases}$

- $\widehat{\langle \zeta_1, \zeta_2 \rangle} = \begin{cases} \langle M_1, M_2 \rangle & \text{if } \widehat{\zeta_1} = M_1 \text{ and } \widehat{\zeta_2} = M_2, \text{ for some } M_1 \text{ and } M_2 \\ \bot & \text{otherwise} \end{cases}$

- for $i = 1, 2$, $\widehat{\pi_i(\zeta)} = \begin{cases} M_i & \text{if } \widehat{\zeta} = \langle M_1, M_2 \rangle, \text{ for some } M_1 \text{ and } M_2 \\ \bot & \text{otherwise} \end{cases}$

The *evaluation function for formulae*, $[\![\cdot]\!] : \Phi \to \{tt, ff\}$, is defined by induction on ϕ. The only non-standard clause is

- $[\![\mathtt{let}\ z = \zeta\ \mathtt{in}\ \phi]\!] = \begin{cases} [\![\phi[\widehat{\zeta}/z]]\!] & \text{if } \widehat{\zeta} \neq \bot \\ ff & \text{otherwise.} \end{cases}$

The operational semantics is defined by the inference rules of Table 2. All rules in the first part of the table, but the last two, are standard from π-calculus. Rule (Guard) says that process ϕP behaves like P provided that ϕ evaluates to true; otherwise, process ϕP is stuck. Rule (Let) attempts evaluation of expression ζ: if the evaluation succeeds, then process $\mathtt{let}\ z = \zeta\ \mathtt{in}\ P$ behaves like process $P[\widehat{\zeta}/z]$, otherwise $\mathtt{let}\ z = \zeta\ \mathtt{in}\ P$ is stuck.

The rules in the upper part of Table 2 only account for process intentions. The rules in the lower part of the table build on those in the upper part, and account for interaction with the environment. They define an environment-sensitive lts whose states are *configurations* of the form $\sigma \rhd P$, where P is a process and substitution σ represents the environment (from now on, terms 'substitution' and 'environment' will be used interchangeably). Transitions take the form

$$\sigma \rhd P \overset{\mu}{\underset{\delta}{\longmapsto}} \sigma' \rhd P'$$

and represent atomic interactions between process P and environment σ, μ is the *process action* and δ is a corresponding *environment action.* Process actions, ranged over by μ, can be of three forms: τ (internal action), $a\,\langle M \rangle$ (input at a where message M is received), and $(\nu\, \widetilde{b})\, \overline{a}\langle M \rangle$ (output at a where message M containing the fresh, private names $\widetilde{b}$ is sent). We shall write $\overline{a}\langle M \rangle$ instead of $(\nu\, \widetilde{b})\, \overline{a}\langle M \rangle$ whenever $\widetilde{b} = \emptyset$. Environment actions, ranged over by δ, can be of three forms: $(\nu\, \widetilde{b})\, \overline{\eta}\langle \zeta \rangle$ (output), $\eta(x)$ (input), and $-$ (no action). Free names and bound names of δ are defined as expected, in particular $bn(\eta(x)) = \{x\}$.

Table 2 Operational semantics (symmetric of (Sum), (Par) and (Com) omitted)

$$(\textsc{Inp})\ a(x).\,P \xrightarrow{a\,\langle M\rangle} P[{}^{M}\!/\!{}_{x}] \qquad\qquad (\textsc{Out})\ \overline{a}M.\,P \xrightarrow{\overline{a}\langle M\rangle} P$$

$$(\textsc{Sum})\ \frac{P \xrightarrow{\mu} P'}{P+Q \xrightarrow{\mu} P'} \qquad\qquad (\textsc{Ide})\ \frac{P[{}^{\tilde v}\!/\!{}_{\tilde x}] \xrightarrow{\mu} P'}{A(\tilde v) \xrightarrow{\mu} P'}\ A(\tilde x) \Leftarrow P$$

$$(\textsc{Par})\ \frac{P \xrightarrow{\mu} P'}{P \mid Q \xrightarrow{\mu} P' \mid Q} \qquad\qquad (\textsc{Com})\ \frac{P \xrightarrow{(\nu\,\widetilde{b})\,\overline{a}\langle M\rangle} P' \qquad Q \xrightarrow{a\,\langle M\rangle} Q'}{P \mid Q \xrightarrow{\tau} (\nu\,\widetilde{b})\,(P' \mid Q')}$$

$$(\textsc{Res})\ \frac{P \xrightarrow{\mu} P'}{(\nu\,c)\,P \xrightarrow{\mu} (\nu\,c)\,P'}\ c \notin n(\mu) \qquad\qquad (\textsc{Open})\ \frac{P \xrightarrow{(\nu\,\widetilde{b})\,\overline{a}\langle M\rangle} P'}{(\nu\,c)\,P \xrightarrow{(\nu\,\widetilde{b}c)\,\overline{a}\langle M\rangle} P'}\ \begin{array}{l} c \neq a, \\ c \in n(M) - \widetilde{b} \end{array}$$

$$(\textsc{Guard})\ \frac{[\![\,\phi\,]\!] = \mathit{tt} \qquad P \xrightarrow{\mu} P'}{\phi P \xrightarrow{\mu} P'} \qquad\qquad (\textsc{Let})\ \frac{\widehat{\zeta} \neq \bot \qquad P[{}^{\widehat{\zeta}}\!/\!{}_{z}] \xrightarrow{\mu} P'}{\texttt{let}\ z = \zeta\ \texttt{in}\ P \xrightarrow{\mu} P'}$$

$$(\textsc{E-Out})\ \frac{P \xrightarrow{(\nu\,\widetilde{b})\,\overline{a}\langle M\rangle} P' \qquad \widehat{\eta\sigma} = a}{\sigma \rhd P \xmapsto[\eta(x)]{(\nu\,\widetilde{b})\,\overline{a}\langle M\rangle} \sigma[{}^{M}\!/\!{}_{x}] \rhd P'} \qquad\qquad (\textsc{E-Tau})\ \frac{P \xrightarrow{\tau} P'}{\sigma \rhd P \xmapsto[-]{\tau} \sigma \rhd P'}$$

$$(\textsc{E-Inp})\ \frac{P \xrightarrow{a\,\langle M\rangle} P' \qquad \widehat{\eta\sigma} = a \qquad M = \widehat{\zeta\sigma} \qquad \widetilde{b} \stackrel{\mathrm{def}}{=} (n(\zeta) - \mathit{dom}(\sigma))}{\sigma \rhd P \xmapsto[(\nu\,\widetilde{b})\,\overline{\eta}\langle\zeta\rangle]{a\,\langle M\rangle} \sigma[{}^{\widetilde{b}}\!/\!{}_{\widetilde{b}}] \rhd P'}$$

Assumptions: $n(\eta) \subseteq \mathit{dom}(\sigma)$ and names in $\widetilde{b}$ fresh for σ and P.

Let us have a closer look at the rules of the environment-sensitive lts. In rule (E-Out), the environment receives a message M and updates its knowledge accordingly. For the transition to take place, channel a must belong to the knowledge of the environment, thus η is some expression describing how a can be obtained out of σ (this is what $\widehat{\eta\sigma} = a$ means). In rule (E-Inp), the environment sends a message M to the process. Message M is not arbitrary: expression ζ describes how it is built out of σ and of the names $\widetilde{b}$ ($\widehat{\zeta\sigma} = M$). The update $[{}^{\widetilde{b}}\!/\!{}_{\widetilde{b}}]$ records the creation of the new names $\widetilde{b}$. Like in the previous rule, a must belong to the knowledge of σ.

Bisimilarity

Bisimilarity relies on a notion of equivalence on environments. Informally, two environments σ and σ' are equivalent whenever they are logically indistinguishable.

Definition 2.1 (equivalence on environments) *Two substitutions σ and σ' are* equivalent, *written $\sigma \sim \sigma'$, if $dom(\sigma) = dom(\sigma')$ and for each formula ϕ with $fn(\phi) \subseteq dom(\sigma)$, it holds that $[\![\phi\sigma]\!] = [\![\phi\sigma']\!]$.* $\diamond$

This logical characterization contains a quantification on all formulae. In [4], we give an equivalent definition that avoids explicitly considering all formulae and is amenable to automatic checking. However, the above definition will suffice for the purposes of this paper.

In the following, input and output actions will be called *visible* actions and we shall use s to range over sequences of visible process actions and u to range over sequences of visible environment actions. We write $\Longmapsto$ to denote the reflexive and transitive closure of $\xmapsto[-]{\tau}$ and, inductively, write $\overset{s}{\underset{u}{\Longmapsto}}$ for $\Longmapsto\xmapsto[\delta]{\mu}\overset{s'}{\underset{u'}{\Longmapsto}}$ when $s = \mu \cdot s'$ and $u = \delta \cdot u'$. $\sigma \rhd P \overset{s}{\underset{u}{\Longmapsto}}$ will stand for 'there are P' and σ' s.t. $\sigma \rhd P \overset{s}{\underset{u}{\Longmapsto}} \sigma' \rhd P'$'. Finally, $\sigma \rhd P \overset{\widehat{\mu}}{\underset{\delta}{\Longmapsto}} \sigma' \rhd P'$ stands for $\sigma \rhd P \overset{\mu}{\underset{\delta}{\Longmapsto}} \sigma' \rhd P'$ if $\mu \neq \tau$, and for $\sigma \rhd P \Longmapsto \sigma' \rhd P'$ if $\mu = \tau$ (the $\widehat{\cdot}$ defined here has of course nothing to do with the evaluation function on expressions previously defined).

We say that a pair of configurations $(\sigma_1 \rhd P, \sigma_2 \rhd Q)$ is *compatible* if $\sigma_1 \sim \sigma_2$. A relation $\mathcal{R}$ is *compatible* if it only contains compatible pairs of configurations. Given a binary relation $\mathcal{R}$, we write $(\sigma_1, \sigma_2) \vdash P \,\mathcal{R}\, Q$ if $(\sigma_1 \rhd P\,,\, \sigma_2 \rhd Q) \in \mathcal{R}$. The intuition of $(\sigma_1, \sigma_2) \vdash P \approx Q$, defined below, is that P and Q are indistinguishable when placed in environments σ_1 and σ_2, respectively. Thus the definition requires, at each step, checking process actions against the same environment action δ.

Definition 2.2 (weak bisimilarity) Weak bisimilarity, *written $\approx$, is the largest binary compatible relation of configurations such that whenever $(\sigma_1, \sigma_2) \vdash P \approx Q$ and $\sigma_1 \rhd P \xmapsto[\delta]{\mu} \sigma_1' \rhd P'$ then there are μ', σ_2' and Q' such that $\sigma_2 \rhd Q \overset{\widehat{\mu'}}{\underset{\delta}{\Longmapsto}} \sigma_2' \rhd Q'$ and $(\sigma_1', \sigma_2') \vdash P' \approx Q'$, and the converse on the transitions of Q and P.* $\diamond$

In [4], we prove that weak bisimilarity is in agreement with *barbed equivalence* ($\cong$), a contextual process semantics. More precisely, weak bisimilarity is included in barbed equivalence: $(\epsilon_V, \epsilon_V) \vdash P \approx Q$ implies $P \cong Q$, for any processes P and Q and $V = fn(P, Q)$. The opposite

Table 3 Basic properties of $\approx$

(REFLEXIVITY)	For any σ and P, $(\sigma,\sigma) \vdash P \approx P$.
(TRANSITIVITY)	If $(\sigma_1,\sigma_2) \vdash P \approx Q$ and $(\sigma_2,\sigma_3) \vdash Q \approx R$ then $(\sigma_1,\sigma_3) \vdash P \approx R$.
(WEAKENING)	Suppose that $(\sigma_1[{}^{M}\!/x], \sigma_2[{}^{N}\!/x]) \vdash P \approx Q$. Then $(\sigma_1,\sigma_2) \vdash P \approx Q$.
(CONTRACTION)	Suppose that $(\sigma_1,\sigma_2) \vdash P \approx Q$ and consider any ζ such that $n(\zeta) \subseteq dom(\sigma_1)$ and $\widehat{\zeta\sigma_1} \neq \bot$. Then $(\sigma_1[{}^{\widehat{\zeta\sigma_1}}\!/x], \sigma_2[{}^{\widehat{\zeta\sigma_2}}\!/x]) \vdash P \approx Q$.
(STRUCT-EQ)	Suppose that $P \equiv Q$. Then, for any σ, $(\sigma,\sigma) \vdash P \approx Q$.

inclusion holds only for the class of *structurally image-finite* processes (see [4]), a sufficiently broad class of processes to ensure that $\approx$ is a fairly general proof technique.

3. COMPOSITIONAL REASONING

We discuss in this section a few laws which are very are useful for compositional reasoning on processes. Table 3 contains a few basic properties of $\approx$, while Table 4 contains some congruence rules for $\approx$. In Table 3, $\equiv$ denotes *structural equivalence*, which is defined as the least equivalence relation over processes that is preserved by parallel composition and restriction, and satisfies the structural laws of [9] plus the laws:

$$(\texttt{let}\ z = \zeta\ \texttt{in}\ P) \equiv P[{}^{\widehat{\zeta}}\!/z] \quad \text{if } \widehat{\zeta} \neq \bot \qquad \text{and} \qquad \phi P \equiv P \quad \text{if } [\![\, \phi \,]\!] = tt$$

The next definition formalizes the concept of *decryption closure* of an environment σ, written $dc(\sigma)$. Elements in $dc(\sigma)$ represent the basic knowledge derivable from σ, i.e. the "building blocks" the environment can use to synthesize more complex messages.

Definition 3.1 (decryption closure and knowledge) *Let W be a set of messages. The* decryption closure *of W, written $dc(W)$, is the set of messages defined inductively as follows:*

(i) $W \subseteq dc(W)$;

(ii) if $k \in dc(W)$ *and* $\{M\}_k \in dc(W)$ *then* $M \in dc(W)$;

(iii) if $\langle M_1, M_2\rangle \in dc(W)$ *then* $M_1, M_2 \in dc(W)$.

The knowledge *of W, written $kn(W)$, is the set of names in $dc(W)$, i.e.: $kn(W) \stackrel{\text{def}}{=} dc(W) \cap \mathcal{N}$. Let σ be a substitution; we let $dc(\sigma) \stackrel{\text{def}}{=} dc(range(\sigma))$ and $kn(\sigma) \stackrel{\text{def}}{=} kn(range(\sigma))$.* $\diamond$

Table 4 Congruence rules for $\approx$

(C-INP)	Suppose that for all ζ such that $\widetilde{b} \stackrel{\mathrm{def}}{=} (n(\zeta) - dom(\sigma_1))$ are fresh and $\widehat{\zeta\sigma_1} \neq \bot$ it holds: $(\sigma_1[\widetilde{b}/\widetilde{b}], \sigma_2[\widetilde{b}/\widetilde{b}]) \vdash P[\widehat{\zeta\sigma_1}/x] \;\approx\; Q[\widehat{\zeta\sigma_2}/x]$. Suppose that $a_i = \widehat{\eta\sigma_i}$ $(i = 1, 2)$ with $n(\eta) \subseteq dom(\sigma_1)$. Then $(\sigma_1, \sigma_2) \vdash a_1(x).P \;\approx\; a_2(x).Q$.
(C-OUT)	Suppose that $(\sigma_1[M_1/x], \sigma_2[M_2/x]) \vdash P \;\approx\; Q$ and that $a_i = \widehat{\eta\sigma_i}$ $(i = 1, 2)$ with $n(\eta) \subseteq dom(\sigma_1)$. Then $(\sigma_1[M_1/x], \sigma_2[M_2/x]) \vdash \overline{a}_1 M_1.P \;\approx\; \overline{a}_2 M_2.Q$.
(C-PAR)	Suppose that $fn(R) \subseteq dom(\sigma_1)$ and $(\sigma_1, \sigma_2) \vdash P \approx Q$. Then $(\sigma_1, \sigma_2) \vdash P \mid R\sigma_1 \;\approx\; Q \mid R\sigma_2$.
(C-RES)	Suppose that $(\sigma_1, \sigma_2) \vdash P \;\approx\; Q$, that $\widetilde{k} \cap n(\sigma_1) = \emptyset$ and that $\widetilde{h} \cap n(\sigma_2) = \emptyset$. Then $(\sigma_1, \sigma_2) \vdash (\nu\, \widetilde{k})\, P \;\approx\; (\nu\, \widetilde{h})\, Q$.

Note that both $dc(\sigma)$ and $kn(\sigma)$ are finite, and can be effectively computed. The next proposition is useful when reasoning on the values a given expression ζ can take on when evaluated under a substitution σ.

Proposition 3.2 *Let ζ and σ be such that $n(\zeta) \subseteq dom(\sigma)$.*

- *If $\widehat{\zeta\sigma}$ is a name then $\widehat{\zeta\sigma} \in kn(\sigma)$.*
- *If $\widehat{\zeta\sigma} = \{M\}_k$ and $k \notin kn(\sigma)$ then $\{M\}_k \in dc(\sigma)$.*
- *If $\widehat{\zeta\sigma} = \langle M_1, M_2\rangle$ then, for $i = 1, 2$, there is ζ_i such that $n(\zeta_i) \subseteq dom(\sigma)$ and $\widehat{\zeta_i\sigma} = M_i$.*

Finally, we state a crucial compositionality result for $\approx$, which strengthens the congruence rule for parallel composition (C-PAR), under the assumption that the involved processes are "safe" for the appropriate environments, that is, they never increase the knowledge of these environments. Formally, we say that a process R is *σ-safe* if for each s, whenever $\sigma \rhd R \xmapsto[u]{s} \sigma' \rhd R' \xmapsto[\eta(x)]{(\nu\,\widetilde{b})\,\overline{a}\langle M\rangle}$, then $M \in dc(\sigma')$ (hence $\widetilde{b} = \emptyset$).

Theorem 1 (safe composition) *Suppose that $(\sigma_1, \sigma_2) \vdash Q_1 \approx Q_2$, that $(\sigma_1, \sigma_2) \vdash R_1 \approx R_2$, and that, for $i = 1, 2$, Q_i and R_i are σ_i-safe. Then $(\sigma_1, \sigma_2) \vdash Q_1 \mid R_1 \approx Q_2 \mid R_2$.*

4. THE KERBEROS PROTOCOL

Specification

Consider a system where two agents A (the initiator) and B (the responder) share two long-term secret keys, k_{AS} and k_{BS} respectively, with a server S. The Kerberos protocol is designed to set up a new secret session key k_{AB} between A and B. Below, we consider a slightly simplified version of this protocol, akin to that considered in [5]. Informally, the protocol can be described as follows:

$$\begin{array}{lll}
\text{Message 1} & A \longrightarrow S: & A, B \\
\text{Message 2} & S \longrightarrow A: & \{n_S, k_{AB}, B, \{n_S, k_{AB}, A\}_{k_{BS}}\}_{k_{AS}} \\
\text{Message 3} & A \longrightarrow B: & \{n_S, k_{AB}, A\}_{k_{BS}}, \{A, n_A\}_{k_{AB}} \\
\text{Message 4} & B \longrightarrow A: & \{n_A\}_{k_{AB}}.
\end{array}$$

In message 1, A starts the protocol by simply communicating to S his intention to establish a new connection with B. In message 2, S generates a fresh key k_{AB} and embodies it into an appropriate certificate, which is sent to A. The certificate uses a fresh nonce n_S, that acts as a timestamp and is meant to assure A and B about the freshness of the message (this is to prevent attacks based on replays of old messages). In message 3, A extracts B's part of the certificate ($\{\cdots\}_{k_{BS}}$) and forwards it to B, together with some challenge-information containing a new nonce n_A. The fourth message represents a synchronization between A and B: the presence of n_A assures A he is really talking to B.

Our intent here is to verify one run of the protocol, under the hypothesis that an old session key k has been compromised. We shall not consider the case of multiple runs, which requires a more complex analysis. For the sake of simplicity, we also suppose that the protocol is always started by A. In our language, the protocol translates to the process K defined in Table 5. For the sake of readability, we have used here and will be using in the sequel a few obvious notational shorthands. For example, $a(\{M\}_k).P$ stands for $a(x).\,\mathtt{let}\ y = \mathtt{dec}_k(x)\ \mathtt{in}\ [y = M]\,P$ and $[x \notin \{a_1, \ldots, a_n\}]$ stands for $\neg[x = a_1] \wedge \ldots \wedge \neg[x = a_n]$. An expanded version of the protocol in Table 5 is reported below:

$$\begin{array}{l}
A \stackrel{\mathrm{def}}{=} \overline{c_{AS}}\langle A, B\rangle.\, c_{AS}(x_1). \\
\quad\quad \mathtt{let}\ x_{n_S} = \pi_1(\mathtt{dec}_{k_{AS}}(x_1)), x_{k_{AB}} = \pi_2(\mathtt{dec}_{k_{AS}}(x_1)), \\
\quad\quad\quad x_B = \pi_3(\mathtt{dec}_{k_{AS}}(x_1)), x'_{n_S} = \pi_1(\mathtt{dec}_{k_{BS}}(\pi_4(\mathtt{dec}_{k_{AS}}(x_1)))), \\
\quad\quad\quad x'_{k_{AB}} = \pi_2(\mathtt{dec}_{k_{BS}}(\pi_4(\mathtt{dec}_{k_{AS}}(x_1)))), x_A = \pi_3(\mathtt{dec}_{k_{BS}}(\pi_4(\mathtt{dec}_{k_{AS}}(x_1)))) \\
\quad\quad \mathtt{in}\ [x_B = B, x_{n_S} = x'_{n_S}, x_{k_{AB}} = x'_{k_{AB}}, x_A = A] \\
\quad\quad\quad \overline{c_{AB}}\langle\{x_{n_S}, x_{k_{AB}}, A\}_{k_{BS}}, \{A, n_A\}_{x_{k_{AB}}}\rangle. \\
\quad\quad\quad c_{AB}(x_2).\, [x_2 = \{n_A\}_{x_{k_{AB}}}].\, \mathbf{0}
\end{array}$$

Table 5 The Kerberos Protocol: spi-calculus specification

$$
\begin{array}{rcl}
A & \stackrel{\text{def}}{=} & \overline{c_{AS}}\langle A, B\rangle.\, c_{AS}(\{x_{n_S}, x_{k_{AB}}, B, x_{cert_B}\}_{k_{AS}}).\\
 & & \overline{c_{AB}}\langle x_{cert_B}, \{A, n_A\}_{x_{k_{AB}}}\rangle.\, c_{AB}(\{n_A\}_{x_{k_{AB}}}).\,\mathbf{0}\\
B^V & \stackrel{\text{def}}{=} & c_{AB}(\{y_{n_S}, y_{k_{AB}}, A\}_{k_{BS}}, \{A, y_{n_A}\}_{y_{k_{AB}}}).\\
 & & [y_{n_S} \notin V]\, \overline{c_{AB}}\langle\{y_{n_A}\}_{y_{k_{AB}}}\rangle.\,\mathbf{0}\\
S & \stackrel{\text{def}}{=} & c_{AS}(A, B).\, \overline{c_{AS}}\langle\{n_S, k_{AB}, B, \{n_S, k_{AB}, A\}_{k_{BS}}\}_{k_{AS}}\rangle.\,\mathbf{0}\\
K & \stackrel{\text{def}}{=} & (\nu\, k_{AS}, k_{BS})\, \big(\, \overline{lost}\langle\{n_{old}, k, A\}_{k_{BS}}, k\rangle.\,\mathbf{0}\\
 & & \qquad \mid ((\nu\, n_A)\, A) \mid B^V \mid ((\nu\, n_S, k_{AB})\, S)\,\big)
\end{array}
$$

$$
\begin{array}{rcl}
B^V & \stackrel{\text{def}}{=} & c_{AB}(y).\\
 & & \texttt{let}\ y_{n_S} = \pi_1(\texttt{dec}_{k_{BS}}(\pi_1(x_1))), y_{k_{AB}} = \pi_2(\texttt{dec}_{k_{BS}}(\pi_1(x_1))),\\
 & & \quad y_A = \pi_3(\texttt{dec}_{k_{BS}}(\pi_1(x_1))), y'_A = \pi_1(\texttt{dec}_{y_{k_{AB}}}(\pi_2(x_1))),\\
 & & \quad y_{n_A} = \pi_1(\texttt{dec}_{y_{k_{AB}}}(\pi_2(x_1)))\\
 & & \ \texttt{in}\ [y_A = A, y'_A = y_A]\, [y_{n_S} \notin V]\, \overline{c_{AB}}\langle\{y_{n_A}\}_{y_{k_{AB}}}\rangle.\,\mathbf{0}\\
S & \stackrel{\text{def}}{=} & c_{AS}(z).\, \texttt{let}\ z_A = \pi_1(z), z_B = \pi_2(z)\\
 & & \qquad \texttt{in}\ [z_A = A, z_B = B]\, \overline{c_{AS}}\langle\{n_S, k_{AB}, z_B, \{n_S, k_{AB}, z_A\}_{k_{BS}}\}_{k_{AS}}\rangle.\,\mathbf{0}\\
K & \stackrel{\text{def}}{=} & (\nu\, k_{AS}, k_{BS})\, \big(\, \overline{lost}\langle\{n_{old}, k, A\}_{k_{BS}}, k\rangle.\,\mathbf{0}\\
 & & \qquad \mid ((\nu\, n_A)\, A) \mid B^V \mid ((\nu\, n_S, k_{AB})\, S)\,\big)
\end{array}
$$

As usual, all bound names in K are assumed to be distinct from one another and from the free names. Subscripts should help reminding the expected values of each variable. For instance, the expected value for x_{certB} is B's certificate, i.e. $\{n_S, k_{AB}, A\}_{k_{BS}}$. The $\overline{lost}$-output action accounts for the accidental loss of an old session key k and of the corresponding certificate for B, $\{n_{old}, k, A\}_{k_{BS}}$. Set V contains all the past nonces generated by S: thus it is assumed that $n_{old} \in V$. Note that the freshness check on n_S is performed by B in $[y_{n_S} \notin V]$ and then propagated to A, via the last synchronization on channel c_{AS}.

There several properties that one might want to check of K. Here, we will just consider two of them. We formalize each property as a process and then check barbed equivalence ($\cong$) between this process and K. In doing so, we partially follow Abadi and Gordon [2], but deviate from their presentation due to our use of an $\overline{error}\langle\rangle$-action to explicitly signal violation of a property. Indeed, the notions of authentication and secrecy considered here are in some sense less demanding than those in [2], as we are not considering implicit information flow – which does not arise in this specific example anyway.

Authentication: *"B and A only accept the key k_{AB} generated by S"*.
Formally, it holds that $K \cong K_{aut}$, where K_{aut} is defined below:

$$\begin{array}{rcl}
A_{aut} & \stackrel{\text{def}}{=} & \overline{c_{AS}}\langle A, B\rangle .\, c_{AS}(\{x_{n_S}, x_{k_{AB}}, B, x_{cert_B}\}_{k_{AS}}). \\
& & \overline{c_{AB}}\langle x_{cert_B}, \{A, n_A\}_{x_{k_{AB}}}\rangle .\, c_{AB}(\{n_A\}_{x_{k_{AB}}}). \\
& & [x_{k_{AB}} \neq k_{AB}]\, \overline{error}\langle\rangle .\, \mathbf{0} \\
B^V_{aut} & \stackrel{\text{def}}{=} & c_{AB}(\{y_{n_S}, y_{k_{AB}}, A\}_{k_{BS}}, \{A, y_{n_A}\}_{y_{k_{AB}}}). \\
& & [y_{n_S} \notin V]\, \overline{c_{AB}}\langle\{y_{n_A}\}_{y_{k_{AB}}}\rangle . \\
& & [y_{k_{AB}} \neq k_{AB}]\, \overline{error}\langle\rangle .\, \mathbf{0} \\
S & \stackrel{\text{def}}{=} & c_{AS}(A, B).\, \overline{c_{AS}}\langle\{n_S, k_{AB}, B, \{n_S, k_{AB}, A\}_{k_{BS}}\}_{k_{AS}}\rangle .\, \mathbf{0} \\
K_{aut} & \stackrel{\text{def}}{=} & (\nu\, k_{AS}, k_{BS}, k_{AB}) \Big(\overline{lost}\langle\{n_{old}, k, A\}_{k_{BS}}, k\rangle .\, \mathbf{0} \\
& & \qquad \mid ((\nu\, n_A)\, A_{aut}) \mid B^V_{aut} \mid ((\nu\, n_S)\, S) \Big) .
\end{array}$$

Output at channel $error$ is used to explicitly signal that a key different from k_{AB} has been received by A or B. The fact that K and K_{aut} are equivalent means that action $\overline{error}\langle\rangle$, hence violation of authentication, never occurs.

Secrecy: *"The key k_{AB} is never revealed"*.
Formally, it holds that $K \mid E \cong K_{secr}$, where K_{secr} is defined below:

$$\begin{array}{rcl}
E & \stackrel{\text{def}}{=} & e(w).\, \mathbf{0} \\
E_{secr} & \stackrel{\text{def}}{=} & e(w).[w = k_{AB}]\, \overline{error}\langle\rangle .\, \mathbf{0} \\
K_{secr} & \stackrel{\text{def}}{=} & (\nu\, k_{AS}, k_{BS}) \Big(\overline{lost}\langle\{n_{old}, k, A\}_{k_{BS}}, k\rangle .\, \mathbf{0} \\
& & \qquad \mid ((\nu\, n_A)\, A) \mid B^V \mid (\nu\, n_S, k_{AB})\, (S \mid E_{secr}) \Big)
\end{array}$$

Here E and E_{secr} represent "eavesdroppers": each of them may at any time receive at e any message known to the environment. After doing so, E just remains silent, but E_{secr} signals an $\overline{error}\langle\rangle$ if the received message happens to be k_{AB}. The fact that $K \mid E$ and K_{secr} are equivalent means that action $\overline{error}\langle\rangle$, hence violation of secrecy, never occurs.

Verification

In this section, we will prove the above authentication and secrecy by reasoning compositionally on processes. Let $I \stackrel{\text{def}}{=} fn(K, K_{aut})$ (that is, $I = \{lost, A, B, c_{AS}, c_{BS}, c_{AB}, k, error\} \cup V$) and $I' \stackrel{\text{def}}{=} fn(K \mid E, K_{secr})$ (that is, $I' = \{e\} \cup I$). By virtue of soundness of $\approx$ w.r.t. $\cong$, to establish authentication it will be sufficient to prove that

$$(\epsilon_I, \epsilon_I) \vdash K \approx K_{aut} \tag{1}$$

while, for secrecy, it will be sufficient to establish that

$$(\epsilon_{I'}, \epsilon_{I'}) \vdash K \mid E \approx K_{secr}. \tag{2}$$

Let M_0 be the message issued in the $\overline{lost}$-action, and M_i, $i = 1, \dots, 4$ be the four messages exchanged in the protocol, that is, let us define:

$$\begin{array}{lcl} M_0 & = & \langle \{n_{old}, k, A\}_{BS}, k \rangle \\ M_1 & = & \langle A, B \rangle \\ M_2 & = & \{n_S, k_{AB}, B, \{n_S, k_{AB}, A\}_{k_{BS}}\}_{k_{AS}} \\ M_3 & = & \langle \{n_S, k_{AB}, A\}_{k_{BS}}, \{A, n_A\}_{k_{AB}} \rangle \\ M_4 & = & \{n_A\}_{k_{AB}}. \end{array}$$

We also define $\sigma \stackrel{\mathrm{def}}{=} \epsilon_I[{}^{M_i}\!/\!{}_{w_i}]_{i \in \{0,\dots,4\}}$ and $\sigma' \stackrel{\mathrm{def}}{=} \epsilon_{I'}[{}^{M_i}\!/\!{}_{w_i}]_{i \in \{0,\dots,4\}}$, where the w_i's are fresh names.

Authentication

Let us start proving (1). Suppose we can prove that:

$$(\sigma, \sigma) \vdash A \;\approx\; A_{aut} \tag{3}$$

and

$$(\sigma, \sigma) \vdash B^V \;\approx\; B^V_{aut} \tag{4}$$

and that each of A, A_{aut}, B^V, B^V_{aut} and S is σ-safe. Then, since $(\sigma, \sigma) \vdash S \approx S$ and $(\sigma, \sigma) \vdash \overline{lost}\langle M_0 \rangle. \mathbf{0} \approx \overline{lost}\langle M_0 \rangle. \mathbf{0}$ trivially hold (Reflexivity), using three times Theorem 1, we get that:

$$(\sigma, \sigma) \vdash \overline{lost}\langle M_0 \rangle. \mathbf{0} \mid A \mid B^V \mid S \;\approx\; \overline{lost}\langle M_0 \rangle. \mathbf{0} \mid A_{aut} \mid B^V_{aut} \mid S\,.$$

We can now use (Weakening) so as to discard the w_i-entries of σ, and then use (C-Res), with $(\nu\, k_{AS}, k_{BS}, k_{AB}, n_A, n_S)$, so as to get:

$$\begin{array}{ll} (\epsilon_I, \epsilon_I) \vdash & (\nu\, k_{AS}, k_{BS}, k_{AB}, n_A, n_S) \left(\overline{lost}\langle M_0 \rangle. \mathbf{0} \mid A \mid B^V \mid S \right) \;\approx \\ & (\nu\, k_{AS}, k_{BS}, k_{AB}, n_A, n_S) \left(\overline{lost}\langle M_0 \rangle. \mathbf{0} \mid A_{aut} \mid B^V_{aut} \mid S \right). \end{array}$$

Applying repeatedly the structural equivalence axiom $(\nu\, a)\,(P \mid Q) \equiv P \mid ((\nu\, a)\, Q)$ if $a \notin fn(P)$, we can push each restriction onto the appropriate position, both in the RHS and LHS. By (Struct-eq) we get thus the wanted equality (1).

So the proof reduces to establishing equations (3,4) above, and to proving that each of A, A_{aut}, B^V, B^V_{aut}, S and $\overline{lost}\langle M_0 \rangle. \mathbf{0}$ is σ-safe. This we do in detail below.

In the sequel, we shall write $R(w)$ to indicate that name w may occur free in R, and, for any message M, we write $R(M)$ in place of $R[{}^{M}\!/\!{}_{w}]$.

Equation (3). Let $A'(x_1)$ and $A'_{aut}(x_1)$ be the parts of A and A_{aut}, respectively, underneath the input prefix $c_{AS}(x_1)$., that is let $A = \overline{c_{AS}}\langle A, B\rangle.\, c_{AS}(x_1).A'(x_1)$ and $A_{aut} = \overline{c_{AS}}\langle A, B\rangle.\, c_{AS}(x_1).A'_{aut}(x_1)$. By virtue of (C-OUT), it suffices to show that

$$(\sigma, \sigma) \vdash c_{AS}(x_1).A'(x_1) \approx c_{AS}(x_1).A'_{aut}(x_1)\,. \tag{5}$$

We prove (5) using (C-INP). Consider any $M \stackrel{\text{def}}{=} \widehat{\zeta\sigma}$ s.t. $\widetilde{b} \stackrel{\text{def}}{=} n(\zeta) - dom(\sigma)$ are fresh names; we have to check that

$$(\sigma[\widetilde{b}/\widetilde{b}], \sigma[\widetilde{b}/\widetilde{b}]) \vdash A'(M) \approx A'_{aut}(M)\,. \tag{6}$$

By definition, if $A'(M)$ is stuck so is $A'_{aut}(M)$ – and vice-versa, thus in these cases (6) follows trivially. Suppose now that $A'(M)$ and $A'_{aut}(M)$ are not stuck. By definition, this implies that $M = \{M'\}_{k_{AS}}$, for some M' (this also implies that $\widetilde{b} = \emptyset$). Now, since $M = \widehat{\zeta\sigma[\widetilde{b}/\widetilde{b}]}$ and $k_{AS} \notin dc(\sigma[\widetilde{b}/\widetilde{b}])$, Proposition 3.2 implies that $M \in dc(\sigma[\widetilde{b}/\widetilde{b}])$: the only possibility is therefore that $M = M_2$. Now, using the structural equivalence rules for `let` and matching, $A'(M)$ and $A'_{aut}(M)$ get simplified as follows:

$$\begin{aligned} A'(M) &\equiv \overline{c_{AB}}\langle M_3\rangle.\, c_{AB}(x_2).[x_2 = \{n_A\}_{k_{AB}}]\mathbf{0} \\ A'_{aut}(M) &\equiv \overline{c_{AB}}\langle M_3\rangle.\, c_{AB}(x_2).[x_2 = \{n_A\}_{k_{AB}}][k_{AB} \neq k_{AB}]\overline{error}\langle\rangle.\,\mathbf{0}. \end{aligned} \tag{7}$$

The RHS of these two equalities are clearly bisimilar under $\sigma[\widetilde{b}/\widetilde{b}]$, because the mismatch-guarded output in the second expression ($[k_{AB} \neq k_{AB}]\overline{error}\langle\rangle.\,\mathbf{0}$) is actually stuck. By (STRUCT-EQ), we therefore get (6) for this case.

A and A_{aut} are σ-safe. Consider A (the proof for A_{aut} is the same). The first output action clearly does not violate σ-safety. Consider now the second output, that is consider the case when

$$(\sigma \rhd A) \xmapsto[\eta(t)]{\overline{c_{AS}}\langle M_1\rangle} \xmapsto[(\nu\widetilde{b})\,\overline{\eta'}\langle\zeta\rangle]{c_{AS}\,\langle M\rangle} (\sigma_1 \rhd A'(M)) \xmapsto[\eta''(t')]{\overline{c_{AB}}\langle M'\rangle}\,.$$

(for some η, η', η'') where $M = \widehat{\zeta\sigma}$. Since $A'(M)$ is not stuck, we know from the above analysis that (7) holds, hence it must be that $M' = M_3 \in dc(\sigma) \subseteq dc(\sigma_1)$. Thus A is σ-safe.

Equation (4). This is similar to the proof of (3). Let us call $B'(y)$ and $B'_{aut}(y)$ the parts of B^V and B^V_{aut} underneath the input prefix, that is assume that $B^V = c_{AB}(y).B'(y)$ and that $B^V_{aut} = c_{AB}(y).B'_{aut}(y)$. Using (C-INP), it suffices to check that for any $M \stackrel{\mathrm{def}}{=} \widehat{\zeta\sigma}$ s.t. $\widetilde{b} \stackrel{\mathrm{def}}{=} n(\zeta) - dom(\sigma)$ are fresh names, it holds

$$(\sigma[\widetilde{b}/\widetilde{b}], \sigma[\widetilde{b}/\widetilde{b}]) \vdash B'(M) \approx B'_{aut}(M)\,. \tag{8}$$

For $B'(M)$ and $B'_{aut}(M)$ not being stuck, M must be a pair, say $M = \langle N_1, N_2\rangle$, and N_1 must be an encryption under k_{BS}. Since $M = \widehat{\zeta\sigma[\widetilde{b}/\widetilde{b}]}$ and $k_{BS} \notin dc(\sigma[\widetilde{b}/\widetilde{b}])$, Proposition 3.2 implies that $N_1 \in dc(\sigma[\widetilde{b}/\widetilde{b}])$: there are two possibilities, either $N_1 = \{n_{old}, k, A\}_{BS}$ or $N_1 = \{n_S, k_{AB}, A\}_{BS}$. In the first case, both $B'(M)$ and $B'_{aut}(M)$ are stuck, hence $\approx$-equivalent, because the mismatch sequence $[y_{n_S} \notin V]$ gets instantiated to $[n_{old} \notin V]$, which evaluates to false. In the second case, for the two processes not being stuck, it must be that N_2 is a pair encrypted under k_{AB}. Again Proposition 3.2 tells us that the only possibility is $N_2 = \{A, n_A\}_{k_{AB}}$, thus $M = M_3$. Using the structural rules for `let` and matching, it is easily seen that $B'(M)$ and $B'_{aut}(M)$ are $\approx$-equivalent under $\sigma[\widetilde{b}/\widetilde{b}]$.

B^V and B^V_{aut} are σ-safe. Let us call M the message received at c_{AB}. The previous analysis shows that the only case in which $B'(M)$ and $B'_{aut}(M)$ are not stuck is when $M = M_3$, which implies that the message sent along $\overline{c_{AB}}$ is M_4. This easily implies that B^V and B^V_{aut} are σ-safe.

S and $\overline{lost}\langle M_0\rangle.\mathbf{0}$ are σ-safe. Trivial.

Secrecy

The crucial step is showing that:

$$(\sigma', \sigma') \vdash E \approx E_{secr} \tag{9}$$

and that E and E_{secr} are σ'-safe. Otherwise, the proof proceeds like in the case of authentication. To prove (9), it will be sufficient to use (C-INP), and check that for any $M \stackrel{\mathrm{def}}{=} \widehat{\zeta\sigma'}$ s.t. $\widetilde{b} \stackrel{\mathrm{def}}{=} n(\zeta) - dom(\sigma')$ are fresh names, it holds that:

$$(\sigma'[\widetilde{b}/\widetilde{b}], \sigma'[\widetilde{b}/\widetilde{b}]) \vdash \mathbf{0} \approx [M = k_{AB}]\, \overline{error}\langle\rangle.\mathbf{0}\,.$$

Since $M = \widehat{\zeta\sigma'[\widetilde{b}/\widetilde{b}]}$ and $k_{AB} \notin dc(\sigma'[\widetilde{b}/\widetilde{b}])$, Proposition 3.2 implies that $M \neq k_{AB}$. Hence, $[M = k_{AB}]\, \overline{error}\langle\rangle.\mathbf{0}$ is stuck and equivalent to $\mathbf{0}$. This analysis also shows that E_{secr} is σ'-safe. Finally, E is trivially σ'-safe.

5. TRACE ANALYSIS

An alternative approach to the analysis of crypto-protocols is based on analyzing the sequences of actions (*traces*) executable by a given spi-calculus specification. We briefly touch upon some ongoing work by Martin Abadi, Marcelo Fiore and one of us (M.B.) in this direction.

A sensible way of formalizing secrecy of the session key in our version of Kerberos, is requiring that action $\overline{error}\langle\rangle$ is never executed by the specification K_{secr}: formally, for no s, u, σ, P and δ, it is the case that $\epsilon_I \rhd K_{secr} \overset{s}{\underset{u}{\Longmapsto}} \sigma \rhd P \overset{\overline{error}\langle\rangle}{\underset{\delta}{\Longmapsto}}$. Trace analysis is less amenable to compositional reasoning than equivalence-based analysis, but is well-suited for mechanical checking. We discuss this below.

The presence of input actions causes any nontrivial process to be infinite-state, as each input variable can be instantiated to any of infinitely many values (see rules (Inp) and (E-Inp) of Table 2). It is possible to avoid this kind of state-explosion by equipping spi-calculus with a *symbolic* operational semantics. Given a generic input-prefixed process $a(x).R$, the idea is to incrementally constrain the set of possible values x can take on, rather than to immediately instantiate x. Constraints are expressed as unifiers. To see how this can be done, let us consider simplified versions of both configurations and transitions: in fact, the 'environment action' δ is not necessary anymore and the first component of any configuration is assumed to be simply a trace of input/output actions. As an example, consider $P \stackrel{\mathrm{def}}{=} a(x).R$, where $R \stackrel{\mathrm{def}}{=} \mathtt{let}\ x' = \mathtt{dec}_k(x)\ \mathtt{in}\ \overline{a}\langle x'\rangle.\, R'$. The two initial symbolic transitions of $\sigma_0 \rhd P$ are reported below. Here σ_0 is any trace of ouput actions that export names in $fn(P)$, thus encoding the initial knowledge of the environment:

$$\sigma_0 \rhd P \xrightarrow{a\,\langle x\rangle} (\sigma_0 \cdot a\,\langle x\rangle) \rhd R \xrightarrow{\overline{a}\langle x'\rangle} (\sigma_0 \cdot a\,\langle\{x'\}_k\rangle \cdot \overline{a}\langle x'\rangle) \rhd R' \,.$$

In the second step, decryption of x with key k is resolved as unification of x with a term $\{x'\}_k$, for a fresh x'. The resulting substitution $([\{x'\}_k/x])$ is propagated through the whole configuration, and causes the first input action of the trace, $a\,\langle x\rangle$, to be 'refined' into $a\,\langle\{x'\}_k\rangle$. A generic matching $[M_1 = M_2]$ would be similarly resolved as unification between M_1 and M_2.

Let us consider again the process K_{secr} that specifies secrecy of the session key in Kerberos. Suppose $\sigma_0 \rhd K_{secr} \xrightarrow{\sigma} (\sigma_0 \cdot \sigma) \rhd P$, where again σ_0 represents the initial knowledge of the environment (i.e.

$fn(K_{secr})$) and:

$$\begin{array}{rcl}\sigma & = & \overline{lost}\langle M_0\rangle\cdot \\ & & \overline{c_{AS}}\langle A,B\rangle\cdot c_{AB}\,\langle A,B\rangle\cdot \\ & & \overline{c_{AS}}\langle\{n_S,k_{AB},B,\{n_S,k_{AB},A\}_{k_{BS}}\}_{k_{AS}}\rangle\cdot c_{AS}\,\langle\{x_{n_S},x_{k_{AB}},B,x_{cert_B}\}_{k_{AS}}\rangle\cdot \\ & & \overline{c_{AB}}\langle x_{cert_B},\{A,n_A\}_{x_{k_{AB}}}\rangle\cdot c_{AB}\,\langle\{y_{n_S},y_{k_{AB}},A\}_{k_{BS}},\{A,y_{n_A}\}_{y_{k_{AB}}}\rangle\cdot \\ & & \overline{c_{AB}}\langle\{y_{n_A}\}_{y_{k_{AB}}}\rangle\cdot c_{AB}\,\langle\{n_A\}_{x_{k_{AB}}}\rangle\cdot \\ & & e\,\langle k_{AB}\rangle\cdot\overline{error}\langle\rangle\end{array}$$

It can be seen that the final trace $\sigma_0\cdot\sigma$ is not *consistent*: this means that it cannot be instantiated to any *concrete* trace, i.e. a trace generable by the 'concrete' operational semantics of Table 2. In fact, it can be proven that no symbolic trace of K_{secr} leading to $\overline{error}\langle\rangle$ is consistent in this sense. Thus, K_{secr} will never execute $\overline{error}\langle\rangle$. In general, it can be shown that consistency of symbolic traces can be effectively decided.

6. CONCLUSIONS

We have outlined some recent approaches to the analysis of crypto-protocols, centered around concepts derived from process algebras, such as labelled transition systems and observational semantics.

When compared to more traditional methods – like CSP-based model checking [7, 13] and theorem proving [12] – major benefits of the equivalence-based approach seem to be a fully satisfactory formalization of many important properties (including implicit information flow), and compositional proof rules. Symbolic trace analysis appears to be similar in spirit to model checking, but it does not suffer from the limitation of model checking, which requires the model to be 'cut' to a suitable finite size, even in the presence of only a limited number of participants. On the other hand, both model checking and theorem proving have proven very effective in practice in finding bugs of crypto-protocols. Real-life case studies are necessary to determine whether process algebraic approaches may represent a valid alternative to established techniques.

Acknowledgments

This work has been partially supported by MURST projects SALADIN and TOSCA.

References

[1] M. Abadi, A.D. Gordon. Reasoning about cryptographic protocols in the spi calculus. *CONCUR'97, Proceedings* (A. Mazurkiewicz, J. Winkowsky, Eds.), *LNCS* 1243, pp.59-73, Springer-Verlag, 1997.

[2] M. Abadi, A.D. Gordon. A calculus for cryptographic protocols: The spi calculus. *Information and Computation*, 148(1):1-70, Academic Press, 1999.

[3] M. Boreale, R. De Nicola. Testing equivalence for mobile processes. *Information and Computation*, 120:279-303, Academic Press, 1995.

[4] M. Boreale, R. De Nicola, R. Pugliese. Proof Techniques for Cryptographic Processes. In *Proc. of the 14th IEEE Symposium Logic In Computer Science (LICS'99)*, IEEE Computer Society Press, pp.157-166, 1999. (Full paper available at: `http://music.dsi.unifi.it`)

[5] M. Burrows, M. Abadi, R. Needham. A Logic of Authentication. *ACM Transactions on Computer Systems*, 8(1):18-36, 1990.

[6] R. De Nicola, M.C.B. Hennessy. Testing Equivalence for Processes. *Theoretical Computers Science*, 34:83-133, Elsevier, 1984.

[7] G. Lowe. Breaking and Fixing the Needham-Schroeder Public-Key Protocol Using FDR. *TACAS'96*, *Proceedings* (T. Margaria, B. Steffen, Eds.), *LNCS* 1055, pp. 147-166, Springer-Verlag, 1996.

[8] S.P. Miller, C. Neumann, J.I. Schiller, J.H. Saltzer. Kerberos authentication and authorization system. In Project Athena Technical Plan, Section E.2.1, MIT, 1987.

[9] R. Milner. The Polyadic π-calculus: a Tutorial. In *Logic and Algebra of Specification* (F.L. Hamer, W. Brauer, H. Schwichtenberg, Eds.), Springer-Verlag, 1993.

[10] R. Milner, J. Parrow, D. Walker. A calculus of mobile processes, (Part I and II). *Information and Computation*, 100:1-77, Academic Press, 1992.

[11] R. Milner, D. Sangiorgi. Barbed Bisimulation. *ICALP'92*, *Proceedings* (W. Kuich, Ed.), *LNCS* 623, pp.685-695, Springer-Verlag, 1992.

[12] L.C. Paulson. Provin Security Protocols Correct. In *Proc. of the 14th IEEE Symposium Logic In Computer Science (LICS'99)*, IEEE Computer Society Press, pp.370-381, 1999.

[13] S. Schneider. Verifying Authentication Protocols in CSP. *IEEE Transactions on Software Engineering*, 24(8):743-758, IEEE Computer Society Press, 1998.