Condensed Notes for Security Protocol

Matteo Acclavio

November 24, 2021

1 Basic definitions

1.1 Messages

Definition 1 (Message)**.** A **message term** or simply **message** is a term generated by the following syntax

M,N	:=	x	variable
	|	$\mathsf{pk}(\mathsf{M})$	public key of M
	|	$\{\!\|\mathsf{M}\|\!\}_{\mathsf{N}}$	message M encrypted using N
	|	$\mathsf{dec}(\mathsf{M},\mathsf{N})$	message M derypted using N
	|	$\langle \mathsf{M},\mathsf{N}\rangle$	pair containing M and N
	|	$\mathsf{fst}(\mathsf{M})$	first component of a message M
	|	$\mathsf{snd}(\mathsf{M})$	second component of a message M
	|	$\mathsf{sign}(\mathsf{M},\mathsf{N})$	message M signed using N
	|	$\mathsf{check}(\mathsf{M},\mathsf{N})$	removing a signature from a message M using N

Equational theory for messages

$\mathsf{dec}(\{\!\|\mathsf{M}\|\!\}_{\mathsf{K}},\mathsf{pk}(\mathsf{K})) \equiv \mathsf{M}$	decription
$\mathsf{fst}(\langle \mathsf{M},\mathsf{N}\rangle) \equiv \mathsf{M}$	first projection
$\mathsf{snd}(\langle \mathsf{M},\mathsf{N}\rangle) \equiv \mathsf{N}$	second projection
$\mathsf{check}(\mathsf{sign}(\mathsf{M},\mathsf{K}),\mathsf{pk}(\mathsf{K})) \equiv \mathsf{M}$	checking signature

Notation 2. In these notes we use the following colours and typesetting whenever it may help the intuition to consider some messages specifically as agents, roles or channel names:

capital calligraphic	Agents	$\mathcal{A},\mathcal{B},\mathcal{C},\ldots$
small c's	Channels	$\mathsf{c}_1,\mathsf{c}_2,\mathsf{c}_3,\ldots$
capital R's	Roles	$\mathsf{R},\mathsf{R}_1,\mathsf{R}',\ldots$

We may push even further on these intuitions by specifying the "type" of a message, e.g., an index $i \in \mathbb{N}$, a channel c, an agent $\mathcal{A}$ or a role R. The sole purpose of these annotations are to help intuition and they can be skipped while reading the text.

1.2 Protocols

A security protocols is given by a set of behaviours called **roles**. We describe roles by means of **threads** defined as follow

$$
\begin{array}{lcll}
\mathsf{T} & := & \mathsf{0} & \text{end the execution} \\
& | & \mathsf{send}(c, \mathsf{M}); \mathsf{T} & \text{sent the message M on the channel c, then continue with T} \\
& | & \mathsf{fresh}x; \mathsf{T} & \text{bind the variable } x \text{, then continue with T} \\
& | & \mathsf{recv}(c, x); \mathsf{T} & \text{receive a message on the channel c and register it as } x \text{, then continue with T} \\
& | & \mathsf{if\ M = N\ then\ T} & \text{if the message M is equal to N, then continue with T (otherwise end)} \\
& | & \mathsf{C} & \text{claims}
\end{array}
\tag{1}
$$

where (for this note) we limit the syntax for claims to be the following

$$
\begin{array}{lcll}
\mathsf{C} & := & \mathsf{0} & \text{end the execution (no claim)} \\
& | & \mathsf{secret}(\mathsf{M}); \mathsf{C} & \text{the message M is secret} \\
& | & \mathsf{w\text{-}alive}(\mathcal{M}, \mathcal{N}); \mathsf{C} & \mathcal{M} \text{ claims weak-aliveness for } \mathcal{N} \\
& | & \mathsf{r\text{-}alive}(\mathcal{M}, \mathcal{N}); \mathsf{C} & \mathcal{M} \text{ claims recent-aliveness for } \mathcal{N} \\
& | & \mathsf{w\text{-}alive\text{-}role}(\mathcal{M}, \mathcal{N}, \mathsf{R}); \mathsf{C} & \mathcal{M} \text{ claims weak-aliveness in the correct role R for } \mathcal{N} \\
& | & \mathsf{r\text{-}alive\text{-}role}(\mathcal{M}, \mathcal{N}, \mathsf{R}); \mathsf{C} & \mathcal{M} \text{ claims recent-aliveness in the correct role R for } \mathcal{N} \\
& | & \mathsf{ni\text{-}agree}(\mathcal{M}, \mathsf{R}); \mathsf{C} & \mathcal{M} \text{ claims non-injective agreement for the role R} \\
& | & \mathsf{i\text{-}agree}(\mathcal{M}, \mathsf{R}); \mathsf{C} & \mathcal{M} \text{ claims injective agreement for the role R} \\
& | & \mathsf{ni\text{-}synch}(\mathcal{M}, \mathsf{R}); \mathsf{C} & \mathcal{M} \text{ claims non-injective synchronization for the role R} \\
& | & \mathsf{ni\text{-}synch}(\mathcal{M}, \mathsf{R}); \mathsf{C} & \mathcal{M} \text{ claims injective synchronization for the role R}
\end{array}
\tag{2}
$$

We may write $\mathsf{claim}(\mathsf{M}_1, \ldots, \mathsf{M}_n)$ to denote a single claim secrecy, aliveness, agreement or synchronization claim with n arguments $\mathsf{M}_1, \ldots, \mathsf{M}_n$.

Remark 3. During the course we defined recent aliveness property by parametrizing them with an additional value representing which intuitively is a channel specifying a run of the role. Here I only translate the ones presented in the book *"Operational Semantics of Security Protocols"*.

A system running a protocol may contain any number of *agents* executing instances of any role. Such an instance of a role made by an agent is called **run**. Agents may run multiple roles in parallel. An **honest** agent shows no behaviour other than the one described by the role they are running. In contrast, an attacker has complete control over the communication network: they can intercept any message and learn their content, as well as insert messages produced using its knowledge.

The **role specification** $\mathsf{Spec}(\mathsf{R})$ of a role R is given by its initial knowledge (set of messages) together with a thread describing its behaviour, that is,

$$\mathsf{Spec}(\mathsf{R}) := \langle \mathsf{K}_\mathsf{R}, \mathsf{T}_\mathsf{R} \rangle \quad \text{for a set of messages } \mathsf{K}_\mathsf{R} = \{\mathsf{M}_1, \ldots, \mathsf{M}_n\} \text{ and a thread } \mathsf{T}_\mathsf{R}$$

We call each occurrence of a send or recv in the thread of a role specification an **event** and we denote by $\mathfrak{E}(\mathsf{R}) = \{\mathfrak{e}_1, \ldots, \mathfrak{e}_n\}$ the set of events of a role R. In particular, we denote $\mathfrak{E}_{\mathsf{recv}}(\mathsf{R})$ and $\mathfrak{E}_{\mathsf{send}}(\mathsf{R})$ the subset of send and receive event of R respectively.

We assume the existence of an order $\triangleleft$ between the events defined by their mutual order in the thread role. We denote $\mathfrak{e} \triangleleft \mathfrak{e}'$ if the event $\mathfrak{e}$ occurs before $\mathfrak{e}'$ in the thread T_R, that is,

$$\mathfrak{e} \triangleleft \mathfrak{e}' \quad \text{if} \quad \mathsf{T}_\mathsf{R} = \mathsf{T}_1; \mathfrak{e}; \mathsf{T}_2; \mathfrak{e}'; \mathsf{T}_3 \quad \text{or} \quad \mathsf{T}_\mathsf{R} = \mathsf{T}_1; \mathfrak{e}; \mathfrak{e}'; \mathsf{T}_2 \quad \text{for some threads } \mathsf{T}_1, \mathsf{T}_2, \mathsf{T}_3$$

To the study of a protocol it is useful to represent multiple threads running in parallel. For this purpose we define **network** which we denote as follows:

$$\begin{array}{lcll} \mathsf{P}, \mathsf{Q} & := & \mathsf{T} & \text{a thread } \mathsf{T} \\ & | & \mathsf{fresh}x; \mathsf{P} & \text{bind the variable } x, \text{ then continue with } \mathsf{T} \\ & | & \mathsf{P} \parallel \mathsf{Q} & \text{execute } \mathsf{P} \text{ in parallel with } \mathsf{Q} \\ & | & !\mathsf{P} & \text{the process } \mathsf{P} \text{ a be executed many times in parallel} \\ & | & \mathsf{C} & \text{claim certain properties} \end{array} \tag{3}$$

The specification of a protocol $\mathfrak{P}$ is given by describing the behaviours of the roles in the protocol. We denote by $\mathfrak{R}(\mathfrak{P})$ the set of roles of the protocol $\mathfrak{P}$ and by $\mathfrak{E}(\mathfrak{P})$ the set of events of all its roles, that is

$$\mathfrak{E}(\mathfrak{P}) = \bigcup_{\mathsf{R} \in \mathfrak{R}(\mathsf{P})} \mathfrak{E}(\mathsf{R})$$

Even if in the specification of the protocol $\mathfrak{P}$ we split the communications between the parties into two distinct events (one send-event and one recv-event), we assume the existence of a one-to-one correspondence between $\mathfrak{E}_{\mathsf{recv}}(\mathfrak{P})$ and $\mathfrak{E}_{\mathsf{send}}(\mathfrak{P})$. We write $\mathfrak{e} \dashrightarrow \mathfrak{e}'$ to denote that $\mathfrak{e}$ is a send-event in $\mathfrak{E}_{\mathsf{recv}}(\mathfrak{P})$ and $\mathfrak{e}'$ is its corresponding recv-event in $\mathfrak{E}_{\mathsf{send}}(\mathfrak{P})$.

1.3 Freshness

In order to define the notion of freshness, we first introduce the notion of free variable.

Definition 4. The set of **free variables** in a message is defined by induction as follows:

$$\begin{array}{ll} \mathbf{fv}(x) & = \{x\} \\ \mathbf{fv}(\mathsf{pk}()) & = \mathbf{fv}(\mathsf{M}) \\ \mathbf{fv}(\{\!|\mathsf{M}|\!\}_K) & = \mathbf{fv}(\mathsf{M}) \cup \mathbf{fv}(\mathsf{N}) \\ \mathbf{fv}(\mathsf{dec}(\mathsf{M}, K)) & = \mathbf{fv}(\mathsf{M}) \cup \mathbf{fv}(\mathsf{N}) \\ \mathbf{fv}(\mathsf{fst}(\mathsf{M})) & = \mathbf{fv}(\mathsf{M}) \\ \mathbf{fv}(\mathsf{snd}(\mathsf{M})) & = \mathbf{fv}(\mathsf{M}) \\ \mathbf{fv}(\langle \mathsf{M}, \mathsf{N} \rangle) & = \mathbf{fv}(\mathsf{M}) \cup \mathbf{fv}(\mathsf{N}) \\ \mathbf{fv}(\mathsf{sign}(\mathsf{M}, \mathsf{N})) & = \mathbf{fv}(\mathsf{M}) \cup \mathbf{fv}(\mathsf{N}) \\ \mathbf{fv}(\mathsf{check}(\mathsf{M}, \mathsf{N})) & = \mathbf{fv}(\mathsf{M}) \cup \mathbf{fv}(\mathsf{N}) \end{array}$$

Definition 5 (Fresh)**.** We say that a variable x is **fresh** in a message M if $x \notin \mathbf{fv}(\mathsf{M})$. In this case we write $\mathsf{Fresh}_\mathsf{M}(x)$ or simply $\mathsf{Fresh}(x)$ if M is clear from the context. We also write $\mathsf{Fresh}_{\mathsf{M}_1,\ldots,\mathsf{M}_n}(\vec{x})$ if $\vec{x} = \{x_1, \ldots, x_n\}$ contains only fresh variables for all $\mathsf{M}_1, \ldots, \mathsf{M}_n$.

The set of free variables in threads or a network is defined similarly as follows:

$$
\begin{array}{ll}
\mathbf{fv}(0) & = \emptyset \\
\mathbf{fv}(\mathsf{fresh}x; \mathsf{T}) & = \mathbf{fv}(\mathsf{T}) \setminus \{x\} \\
\mathbf{fv}(\mathsf{recv}(c, x)) & = \mathbf{fv}(\mathsf{T}) \setminus \{x\} \\
\mathbf{fv}(\mathsf{send}(\mathsf{M}, \mathsf{N})) & = \mathbf{fv}(\mathsf{M}) \cup \mathbf{fv}(\mathsf{N}) \\
\mathbf{fv}(\mathsf{if\ M = N\ then\ T}) & = \mathbf{fv}(\mathsf{M}) \cup \mathbf{fv}(\mathsf{N}) \cup \mathbf{fv}(\mathsf{T}) \\
\mathbf{fv}(\mathsf{claim}(\mathsf{M}_1, \ldots, \mathsf{M}_n)) & = \bigcup_{i=1}^{n} \mathbf{fv}(\mathsf{M}_i) \\
\hline
\mathbf{fv}(\mathsf{P} \parallel \mathsf{Q}) & = \mathbf{fv}(\mathsf{P}) \cup \mathbf{fv}(\mathsf{Q}) \\
\mathbf{fv}(!\mathsf{P}) & = \mathbf{fv}(\mathsf{P}) \\
\mathbf{fv}(\mathsf{fresh}x\mathsf{P}) & = \mathbf{fv}(\mathsf{P}) \setminus \{x\}
\end{array}
$$

And we extend the definition of freshness from Theorem 5 to threads and protocols.

1.4 Substitution

A **substitution** θ is a function mapping a finite number of variables $x_1, \ldots, x_n$ into a finite number of messages $\mathsf{M}_1, \ldots, \mathsf{M}_n$ and we denote it

$$
\theta = \{x_1 \mapsto \mathsf{M}_1, \ldots, x_n \mapsto \mathsf{M}_n\} \quad \text{or} \quad \theta = \left\{ \begin{array}{c} x_1 \mapsto \mathsf{M}_1 \\ \vdots \\ x_n \mapsto \mathsf{M}_n \end{array} \right\}
$$

Each substitution defines a map from messages to messages associating to each message M a message $M\theta$ defined inductively as follows

$$
x\theta = \begin{cases} \mathsf{M}_i & \text{if } x = x_i \text{ for a } i \\ x & \text{otherwise} \end{cases}
$$

$$
\begin{array}{lll}
\langle M, N \rangle \theta = \langle M\theta, N\theta \rangle & \mathsf{fst}(\mathsf{M})\theta = \mathsf{fst}(\mathsf{M}\theta) & \mathsf{snd}(\mathsf{M})\theta = \mathsf{snd}(\mathsf{M}\theta) \\
\{\!|M|\!\}_N \, \theta = \{\!|M\theta|\!\}_{N\theta} & \mathsf{dec}\,(M, N)\,\theta = \mathsf{dec}\,(M\theta, N\theta) & \\
\mathsf{sign}(\mathsf{M}, \mathsf{N})\theta = \mathsf{sign}(\mathsf{M}\theta, \mathsf{N}\theta) & \mathsf{check}(\mathsf{M}, \mathsf{N})\theta = \mathsf{check}(\mathsf{M}\theta, \mathsf{N}\theta) &
\end{array}
$$

Substitutions can also be applied to threads and networks:

$$
\begin{array}{rcll}
0\theta & = & 0 & \\
\mathsf{fresh}x; \mathsf{T}\theta & = & \mathsf{fresh}z; (\mathsf{T}\,\{x \mapsto z\}\,\theta) & \text{for a } z \text{ such that } \mathsf{Fresh}_{\mathsf{T}}(z) \\
(\mathsf{recv}(\mathsf{M}, x); \mathsf{T})\theta & = & \mathsf{recv}(\mathsf{M}\theta, z); (\mathsf{T}\,\{x \mapsto z\}\,\theta) & \text{for a } z \text{ such that } \mathsf{Fresh}_{\mathsf{T}}(z) \\
(\mathsf{send}(\mathsf{M}, \mathsf{N}); \mathsf{T})\theta & = & \mathsf{send}(\mathsf{M}\theta, \mathsf{N}\theta); (\mathsf{T}\theta) & \\
(\mathsf{if\ M = N\ then\ ; T})\theta & = & \mathsf{if\ M}\theta = \mathsf{N}\theta\ \mathsf{then\ ; } (\mathsf{T}\theta) & \\
\mathsf{claim}(\mathsf{M}_1, \ldots, \mathsf{M}_n)\theta; \mathsf{C} & = & \mathsf{claim}(\mathsf{M}_1\theta, \ldots, \mathsf{M}_n\theta); \mathsf{C} & \\
(\mathsf{P} \parallel \mathsf{Q})\theta & = & \mathsf{P}\theta \parallel \mathsf{Q}\theta & \\
(!\mathsf{P})\theta & = & !(\mathsf{P}\theta) &
\end{array}
$$

Definition 6 (State)**.** A **state** (also called **extended process**) is given by an substitution θ (also called the **active substitution**of the state) and a network P.

$$
\begin{array}{rcll}
\mathsf{S} & := & [\theta; \mathsf{P}] & \text{where } \theta \text{ is a substitution and P is a network} \\
& | & \mathsf{fresh}x; \mathsf{S} & \text{binding the variable } x \text{ in S}
\end{array}
$$

$$\text{send}\ \frac{}{[\theta; \mathsf{send}(\mathsf{M}, \mathsf{N}); \mathsf{T}] \xrightarrow{\mathsf{send}(\mathsf{K},x)} [\theta \circ \{x \mapsto \mathsf{N}\}; \mathsf{T}]}\ \mathsf{K}\theta{\equiv}\mathsf{M} \text{ and } \mathsf{Fresh}_{\mathsf{K},\mathsf{N},\mathsf{T},\theta}(x)$$

$$\text{recv}\ \frac{}{[\theta; \mathsf{recv}(\mathsf{M}, x); \mathsf{T}] \xrightarrow{\mathsf{recv}(\mathsf{K},\mathsf{L})} [\theta; \mathsf{T}\{x \mapsto \mathsf{N}\theta\}]}\ \mathsf{K}\theta{\equiv}\mathsf{M} \text{ and } \mathsf{L}\theta{\equiv}\mathsf{N}$$

$$\text{match}\ \frac{[\theta; \mathsf{T}] \xrightarrow{\alpha} \mathsf{S}}{[\theta; \mathsf{if}\ \mathsf{M} = \mathsf{N}\ \mathsf{then}\ ; \mathsf{T}] \xrightarrow{\alpha} \mathsf{S}}\ \mathsf{M}{\equiv}\mathsf{N}$$

$$\text{fresh}\ \frac{\mathsf{S} \xrightarrow{\alpha} \mathsf{S}'}{\mathsf{fresh}x; \mathsf{S} \xrightarrow{\alpha} \mathsf{fresh}x; \mathsf{S}'}\ \mathsf{Fresh}_{\alpha}(x)$$

$$\text{extrude}\ \frac{[\theta; \mathsf{T}] \xrightarrow{\alpha} \mathsf{S}}{[\theta; \mathsf{fresh}x; \mathsf{T}] \xrightarrow{\alpha} \mathsf{fresh}x; \mathsf{S}}\ \mathsf{Fresh}_{\alpha,\theta}(x)$$

$$\text{claim}\ \frac{}{[\theta; \mathsf{claim}(\mathsf{M}_1, \ldots, \mathsf{M}_n); \mathsf{C}] \xrightarrow{\mathsf{claim}(\mathsf{N}_1,\ldots,\mathsf{N}_n)} [\theta; \mathsf{C}]}\ \mathsf{N}_i\theta{\equiv}\mathsf{M}_i \forall i{\in}\{1,\ldots,n\}$$

Figure 1: Labelled transition rules on threads

2 Operational semantics

An **action** α is an occurrence of a send, recv or claim in a protocol specification, that is,

$$\alpha \in \left\{ \begin{array}{l} \mathsf{send}(\mathsf{M}_1, \mathsf{M}_2) \\ \mathsf{recv}(\mathsf{M}_1, \mathsf{M}_2) \\ \mathsf{claim}(\mathsf{M}_1, \ldots, \mathsf{M}_n) \end{array} \middle| \ \mathsf{M}_1, \ldots, \mathsf{M}_n \text{ messages} \right\} \tag{4}$$

A **labelled transition** $\xrightarrow{\alpha}$ is a relation between two states S and S′ labelled by an action α. We say that the state S reach the state S′ after the transition α when $\mathsf{S} \xrightarrow{\alpha} \mathsf{S}'$ holds, that is when this expression is derivable in the sequent system defined by the rules in Figure 1 and Figure 2. More in general, we say that S_0 can reach the state S_{n+1} if there are states $\mathsf{S}_1, \ldots \mathsf{S}_n$ and actions $\alpha_1, \ldots, \alpha_n$ such that $\mathsf{S}_i \xrightarrow{\alpha_i} \mathsf{S}_{i+1}$ for all $i \in \{1, \ldots, n\}$.

A **trace** is a sequence of actions which may be performed during the execution of a protocol. We denote traces as formulas generated by the following syntax

$$\phi \coloneqq \top \mid \langle \alpha \rangle \, \phi \qquad \text{with } \alpha \text{ an action from Equation (4)}$$

Definition 7 (Satisfiability)**.** We define the notion of **satisfiability** for a trace ϕ in a

$$\text{L-parallel}\ \frac{[\theta;\mathsf{P}] \xrightarrow{\alpha} \mathsf{fresh}\vec{x};[\theta';\mathsf{P}' \parallel \mathsf{Q}]}{[\theta;\mathsf{P} \parallel \mathsf{Q}] \xrightarrow{\alpha} \mathsf{fresh}\vec{x};[\theta';\mathsf{P}' \parallel \mathsf{Q}]}\ \mathsf{Fresh}_{\mathsf{Q}}(\mathbf{bn}(\alpha)\cup\vec{x})$$

$$\text{R-parallel}\ \frac{[\theta;\mathsf{Q}] \xrightarrow{\alpha} \mathsf{fresh}\vec{x};[\theta';\mathsf{P} \parallel \mathsf{Q}']}{[\theta;\mathsf{P} \parallel \mathsf{Q}] \xrightarrow{\alpha} \mathsf{fresh}\vec{x};[\theta';\mathsf{P}' \parallel \mathsf{Q}]}\ \mathsf{Fresh}_{\mathsf{P}}(\mathbf{bn}(\alpha)\cup\vec{x})$$

$$\text{!-repetition}\ \frac{[\theta;\mathsf{P}] \xrightarrow{\alpha} \mathsf{fresh}\vec{x};[\theta';\mathsf{Q}]}{[\theta;!\mathsf{P}] \xrightarrow{\alpha} [\theta';\mathsf{Q} \parallel !\mathsf{P}]}\ \mathsf{Fresh}_{\mathsf{P}}(\mathbf{bn}(\alpha)\cup\vec{x})$$

$$\text{where} \qquad \mathbf{bn}(\alpha) = \begin{cases} \{x\} & \text{if } \alpha = \mathsf{recv}(\mathsf{c}, x) \\ \emptyset & \text{otherwise} \end{cases}$$

Figure 2: Labelled transition rules on networks

state S by induction ϕ as follows:

$$\begin{array}{ll} \mathsf{S} \Vdash \top & \text{for any } \mathsf{S} \\ \mathsf{S} \Vdash \langle\alpha\rangle\,\phi & \text{if there is } \mathsf{S}' \text{ such that } \mathsf{S} \xrightarrow{\alpha} \mathsf{S}' \text{ and } \mathsf{S}' \Vdash \phi \end{array}$$

3 Secrecy

Definition 8. A state S reveals a secret M if $\mathsf{S} \Vdash \langle\alpha_1\rangle \cdots \langle\alpha_n\rangle\,\langle\mathsf{secret}(\mathsf{M})\rangle\,\top$.

4 Authentication

In order to define authentication properties, we need to be able to keep track not only of the exchanged messages, but also of the author of a message, its role when the message was sent or received, and in which run of that given role this event is performed.

To simplify our definitions, we convoy part of the information in the channels we use for the send and receive actions. More precisely, we assume that each honest agent will use a unique channel for each run of a role during the execution of the the protocol after declaring it publicly.

For this purpose, we assume that each agent will send a message on a special channel called *channels* to inform all the other parties taking action in the protocol its name and its role. This declaration is done by sending on this dedicated channel *channels* a message containing the name $\mathcal{A}$ of the agent, the role R this agent will be performing, and the name c of the dedicated channel the agent is going to use during its execution of that given role. That is, the agent $\mathcal{A}$ will perform the following action

$$\mathsf{Declare}(\mathcal{A}, \mathsf{R}, \mathsf{c}) := \mathsf{send}(\mathit{channels}, \langle\langle\mathcal{A}, \mathsf{R}\rangle, \mathsf{c}\rangle)$$

Note that we assume that each honest agent will use the communicated channel only for the actions of a single run of the chosen role, that is, there will be a one-to-one

correspondence between the set of channels communicated on *channels* and the set of runs of a role by an agent.

After such a declaration, we would be able to identify the name of the agent, its role while performing the action, and during which run the agent performed the action by only looking at the channel used during a send or recv assuming that the action has been performed by an honest agent.

Remark 9. We are not assuming that the information sent through *channels* is private, that is, the information of the channels used by each agent during a specific run of a role is public and can be exploited by an attacker. Moreover, we are not excluding the possibility for an attacker to send a dishonest message in *channels* in which it claims a fake name or a fake role.

4.1 Aliveness

Definition 10. A state S satisfies *weak aliveness* whenever for each trace

$$\phi = \langle \alpha_1 \rangle \cdots \langle \alpha_n \rangle \langle \mathsf{w\text{-}alive}(\mathcal{M}, \mathcal{N}) \rangle \top$$

if $\mathsf{S} \Vdash \phi$, then $\mathcal{N}$ is performing an action during one of its run, that is:

1. there is $i \in \{1, \ldots, n\}$ such that $\alpha_i = \mathsf{Declare}(\mathcal{N}, \mathsf{R_x}, \mathsf{c_y})$ for some $\mathsf{R_x}$ and $\mathsf{c_y}$, and
2. there is $j \in \{i+1, \ldots, n\}$ such that $\alpha_j \in \{\mathsf{send}(\mathsf{c_y}, \mathsf{Z}), \mathsf{recv}(\mathsf{c_y}, \mathsf{Z}) \mid \mathsf{Z} \in \mathcal{M}\}$

Definition 11. A state S satisfies *recent aliveness* whenever for each trace

$$\phi = \langle \alpha_1 \rangle \cdots \langle \alpha_n \rangle \langle \mathsf{r\text{-}alive}(\mathcal{M}, \mathcal{N}) \rangle \top$$

if $\mathsf{S} \Vdash \phi$, then $\mathcal{N}$ is performing an action during each of its run, that is,

1. for all $i \in \{1, \ldots, n\}$ such that $\alpha_i = \mathsf{Declare}(\mathcal{N}, \mathsf{R_x}, \mathsf{c_y})$ for some $\mathsf{R_x}$ and $\mathsf{c_y}$,
2. there is $j(i) \in \{i+1, \ldots, n\}$ such that $\alpha_{j(i)} \in \{\mathsf{send}(\mathsf{c_y}, \mathsf{Z}), \mathsf{recv}(\mathsf{c_y}, \mathsf{Z}) \mid \mathsf{Z} \in \mathcal{M}\}$

Definition 12. A state S satisfies *weak aliveness in the correct role* whenever for each trace

$$\phi = \langle \alpha_1 \rangle \cdots \langle \alpha_n \rangle \langle \mathsf{w\text{-}alive\text{-}role}(\mathcal{M}, \mathcal{N}, \mathsf{R}) \rangle \top$$

if $\mathsf{S} \Vdash \phi$, then $\mathcal{N}$ is performing an action in the role R during one of its run, that is,

1. there is $i \in \{1, \ldots, n\}$ such that $\alpha_i = \mathsf{Declare}(\mathcal{N}, \mathsf{R}, \mathsf{c_y})$ for a $\mathsf{c_y}$,
2. there is $j \in \{i+1, \ldots, n\}$ such that $\alpha_j \in \{\mathsf{send}(\mathsf{c_y}, \mathsf{Z}), \mathsf{recv}(\mathsf{c_y}, \mathsf{Z}) \mid \mathsf{Z} \in \mathcal{M}\}$

It satisfies *recent aliveness in the correct role* if for each trace

$$\phi = \langle \alpha_1 \rangle \cdots \langle \alpha_n \rangle \langle \mathsf{r\text{-}alive\text{-}role}(\mathcal{M}, \mathcal{N}, \mathsf{R}) \rangle \top$$

if $\mathsf{S} \Vdash \phi$, then $\mathcal{N}$ is performing an action in the role R during each of its run, that is,

1. for all $i \in \{1, \ldots, n\}$ such that $\alpha_i = \mathsf{Declare}(\mathcal{N}, \mathsf{R_x}, \mathsf{c_y})$ for some $\mathsf{R_x}$ and $\mathsf{c_y}$,
2. there is $j(i) \in \{i+1, \ldots, n\}$ such that $\alpha_{j(i)} \in \{\mathsf{send}(\mathsf{c_y}, \mathsf{Z}), \mathsf{recv}(\mathsf{c_y}, \mathsf{Z}) \mid \mathsf{Z} \in \mathcal{M}\}$

4.2 Agreement and Synchronization

Definition 13. Let S be a state of an execution of the protocol $\mathfrak{P}$. We say that S satisfies *non-injective agreement* whenever for each trace

$$\phi = \langle \alpha_1 \rangle \cdots \langle \alpha_n \rangle \left\langle \mathsf{ni\text{-}agree}(\mathcal{M}, \hat{\mathsf{R}}) \right\rangle \top$$

if $\mathsf{S} \Vdash \phi$, then there exists a run for each role of the protocol such that each pair of matched send and receive events in P are executed in these runs and their content the same. That is, for each $\mathsf{R} \in \mathfrak{R}(\mathfrak{P})$ there is (at least) a channel c_R such that

1. $\alpha_{i(\mathsf{R})} = \mathsf{Declare}(\mathcal{A}_\mathsf{R}, \mathsf{R}, \mathsf{c}_\mathsf{R})$ for an index $i(\mathsf{R}) \in \{1, \ldots, n\}$ and an agent $\mathcal{A}_\mathsf{R}$
2. for each event $\mathfrak{e} \in \mathfrak{E}(\mathsf{R})$ there is a index $i(\mathfrak{e}) \in \{1, \ldots, n\}$ such that

$$\alpha_{i(\mathfrak{e})} = \begin{cases} \mathsf{send}(\mathsf{c}_\mathsf{R}, \mathsf{M}) & \text{if } \mathfrak{e} \in \mathfrak{E}_{\mathsf{recv}}(\mathsf{R}) \\ \mathsf{recv}(\mathsf{c}_\mathsf{R}, \mathsf{M}) & \text{if } \mathfrak{e} \in \mathfrak{E}_{\mathsf{send}}(\mathsf{R}) \end{cases}$$

and such that if $\mathfrak{e} \dashrightarrow_{\mathfrak{P}} \mathfrak{e}'$, then

$$\alpha_{i(\mathfrak{e})} = \mathsf{send}(\mathsf{c}_\mathsf{R}, \mathsf{N}) \quad \text{and} \quad \alpha_{i(\mathfrak{e}')} = \mathsf{recv}(\mathsf{c}_{\mathsf{R}'}, \mathsf{M}) \qquad \text{with } \mathsf{c}_\mathsf{R} = \mathsf{c}_{\hat{\mathsf{R}}} \text{ or } \mathsf{c}_{\mathsf{R}'} = \mathsf{c}_{\hat{\mathsf{R}}}$$

and $\mathsf{M}\theta_{i(\mathfrak{e})} = \mathsf{N}\theta_{i(\mathfrak{e}')}$, where θ_k is the active substitution of the state S_k such that

$$\mathsf{S} = \mathsf{S}_0 \xrightarrow{\alpha_1} \cdots \xrightarrow{\alpha_k} \mathsf{S}_k = [\theta_k; \mathsf{P}_k]$$

We say that S satisfies *non-injective synchronization* if we additionally ask that

3. for all $\mathfrak{e}, \mathfrak{e}' \in \mathfrak{E}(\mathfrak{P})$, if $\mathfrak{e} \vartriangleleft \mathfrak{e}'$ or $\mathfrak{e} \dashrightarrow_{\mathfrak{P}} \mathfrak{e}'$, then $i(\mathfrak{e}) < i(\mathfrak{e}')$.

We say that S satisfies *injective agreement* (respectively *injective synchronization*) if we add to the definition of non-injective agreement (non-injective synchronization) the additional constrain that the channel c_R is unique for each $\mathsf{R} \in \mathfrak{R}(\mathfrak{P})$, that is, there is an *injective* map from $\mathfrak{R}(\mathfrak{P})$ to $\{\mathsf{c}_\mathsf{R} \mid \exists i_\mathsf{R} \in \{1, \ldots, n\} \text{ s.t. } \alpha_{i(\mathsf{R})} = \mathsf{Declare}(\mathcal{A}_\mathsf{R}, \mathsf{R}, \mathsf{c}_\mathsf{R})\}$.

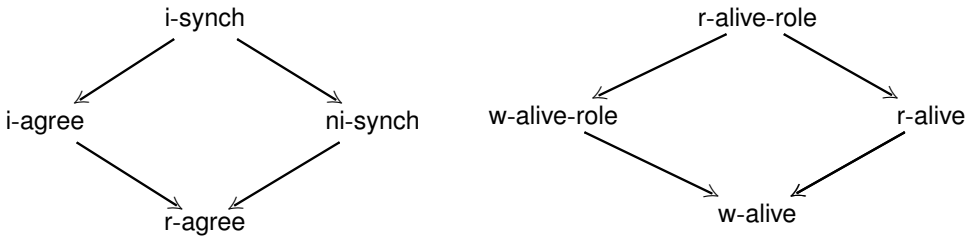

Figure 3: Relation between authentication statements.