Formal Proofs of Security Protocols

David Baelde, ENS Paris-Saclay

December 12, 2018

Contents

We present in section 1 a formal model for representing security protocols, and illustrate how it can be used, e.g. to model secrecy. Secrecy verification is developped in more detail in sections 2 and 3 respectively for bounded and unbounded executions: in the former case we obtain a decidability result by using deducibility constraints; in the latter we describe the semi-decision procedure behind Proverif, using a Horn-clause abstraction of protocols. We go back to the semantics of our processes in section 4 to define and study various behavioral equivalences, and show how they are useful to model more advanced security properties such as strong secrecy, anonymity, unlinkability, etc. Section 5 is dedicated to the automated verification of equivalences. We will conclude in section 6 with yet undetermined advanced topics.

1 Model

Before anything else, we must define a formal model for security protocols. As is common, we shall use a variant of the (applied) π-calculus. More specifically, the calculus defined below is very close to the one used in Proverif.

The first step is to define a *term language* to represent messages and computations over them (section 1.1). Then, a *process calculus* will be used to represent protocols (section 1.2).

1.1 Terms

Terms are formal representations of messages and computations over them. We start by assuming several disjoint and countable sets of basic objects:

- a set $\mathcal{X}$ of variables, which will be denoted by x, y, z;
- a set $\mathcal{N}$ of names, which will be denoted by n, m, k.

Names will be used to represent the secrets of honest participants: nonces, keys, identities, etc. The attacker will not know them *a priori*. Function symbols will represent specific terms, terms constructions or computations over terms. Variables, as usual, will be used as placeholders for unknown terms.

Then, we assume a *signature* Σ, that is a set of *function symbols* together with an arity $\mathsf{ar}_\Sigma : \Sigma \to \mathbb{N}$. Given a set of basic terms B, the set $\mathcal{T}(B)$ of *terms* generated from B using Σ is defined as the least set containing B and closed by application of function symbols respecting their arities. Terms will be denoted by s, t, u, v.

Example 1. *One possible signature is* $\Sigma = \{\ \mathbf{senc},\ \mathbf{sdec},\ \mathbf{pair},\ \mathbf{proj}_1,\ \mathbf{proj}_2,\ \mathbf{ok}\ \}$*. The symbols* **senc** *and* **sdec***, of arity 2, represent symmetric encryption and decryption. Pairing is modeled using* **pair** *of arity 2 and projection functions* $\mathbf{proj}_1$ *and* $\mathbf{proj}_2$*, both of arity 1. Finally, the symbol* **ok** *is of arity* 0*, i.e. it is a constant. With this signature we have* $\mathbf{pair}(\mathbf{ok}, \mathbf{ok}) \in \mathcal{T}(\emptyset)$*,* $\mathbf{senc}(\mathbf{pair}(\mathbf{ok}, \mathbf{ok}), k) \in \mathcal{T}(\mathcal{N})$ *and* $\mathbf{sdec}(\mathbf{ok}, x) \in \mathcal{T}(\mathcal{X})$*. However,* $\mathbf{senc}(\mathbf{ok})$ *and* $\mathbf{ok}(\mathbf{ok})$ *are not terms.*

When using this signature, we will often write $\langle s, t\rangle$ *rather than* $\mathbf{pair}(s, t)$*, and* $\{m\}_k$ *for* $\mathbf{senc}(m, k)$*.*

Given a term t, we define $\mathsf{fv}(t)$ as the set of variables that occur in t. Similarly, $\mathsf{fn}(t)$ is the set of names occurring in t. A term is said to be *closed* when it contains no variable. A *substitution* is a finite domain map from $\mathcal{X}$ to $\mathcal{T}(B)$ for some B. Substitutions will be denoted by θ or σ, their domain will be noted $\mathsf{dom}(\cdot)$. A substitution $\{x_i \mapsto t_i\}_{i\in[1;n]}$ may also be written $[t_i/x_i]_{i\in[1;n]}$. In particular, $[t/x]$ is the substitution of domain $\{x\}$ that maps x to t. The application of a substitution θ to a term t is defined as usual and noted $t\theta$. In particular, when $\mathsf{dom}(\theta) \cap \mathsf{fv}(t) = \emptyset$, $t\theta = t$.

We will introduce two mechanisms for giving a meaning to function symbols. First, we will introduce an *equations* to model when two terms should be considered as being equal, i.e. when they represent computations that yield the same result. For example, we may equate $\mathbf{proj}_1(\mathbf{pair}(s,t))$ and s. Second, we provide our term algebra with a means to describe *computations* that may fail. For example, we may have that $\mathbf{sdec}(\mathbf{senc}(s,k),k)$ reduces to s but $\mathbf{sdec}(\mathbf{ok},k)$ fails, indicating an encryption scheme where it is possible to distinguish random messages from actual ciphertexts.

Equations and reductions will be separate mechanisms, each one taking place on a specific kind of function symbol. Before introducing them, we thus assume that our signature is split between *constructor* and *destructor* symbols, i.e. $\Sigma = \Sigma_c \uplus \Sigma_d$. In the following we write $\mathcal{T}_c(B)$ for terms built from B using only constructor symbols, i.e., elements of Σ_c. Elements of $\mathcal{T}_c(\mathcal{N})$ are called *messages*.

1.1.1 Equational theory

Our equational theory is going to be generated from equations between terms that may contain variables but no names. We thus assume a set of equations $\mathsf{E} \subseteq \mathcal{T}_c(\mathcal{X})^2$; we will use an infix notation for it, writing $s \mathrel{\mathsf{E}} t$ rather than $(s,t) \in \mathsf{E}$. Then, for any set of basic terms B we define the binary relation $=^B_{\mathsf{E}}$ over $\mathcal{T}(B \cup \mathcal{X})$ as the least equivalence relation that contains E and is closed under substitution and context closure. In other words, we impose that:

- for all $s \mathrel{\mathsf{E}} t$, we have $s =^B_{\mathsf{E}} t$;
- for all $s =^B_{\mathsf{E}} t$ and for any substitution $\theta : \mathcal{X} \to \mathcal{T}(B \cup \mathcal{X})$, we have $s\theta =^B_{\mathsf{E}} t\theta$;
- for all $f \in \Sigma$ with $\mathsf{ar}(f) = n$,
  for all $s_1, \ldots, s_n$ and $t_1, \ldots, t_n$ such that $s_i =^B_{\mathsf{E}} t_i$ for all $i \in [1;n]$,
  we have $f(s_1, \ldots, s_n) =^B_{\mathsf{E}} f(t_1, \ldots, t_n)$.

Equations should not be able to distinguish specific names, because names represent values that are generated at random.

For practical applications, equations are often oriented into *rewrite rules*, and good properties (e.g. confluence, termination) are required to obtain e.g. computable equality and unification modulo E.

Example 2. *With the signature of ex. 1, and assuming that $\Sigma = \Sigma_c$, consider* E *made of three equations:*

$$\mathbf{sdec}(\mathbf{senc}(x,y),y) \mathrel{\mathsf{E}} x, \quad \textit{and} \quad \mathbf{proj}_i(\mathbf{pair}(x_1,x_2)) \mathrel{\mathsf{E}} x_i \quad \textit{for } i \in \{1,2\}.$$

We then have $\mathbf{proj}_1(\mathbf{sdec}(\mathbf{senc}(\mathbf{pair}(\mathbf{ok},n),k),k)) =^{\mathcal{N}}_{\mathsf{E}} \mathbf{ok}$ *but* $\mathbf{ok} \neq^{\mathcal{N}}_{\mathsf{E}} \mathbf{pair}(\mathbf{ok},n)$.

Exercise 1. *Assume $B \subseteq C$. Using the induction principle associated to $=^B_{\mathsf{E}}$, show that, for any $s,t \in \mathcal{T}(B \cup \mathcal{X})$, $s =^B_{\mathsf{E}} t$ implies $s =^C_{\mathsf{E}} t$.*

The previous exercise justifies that we do not need to specify B explicitly when considering an equality: we will simply write $s =_{\mathsf{E}} t$, meaning that $s =^B_{\mathsf{E}} t$ holds for any B such that $s,t \in \mathcal{T}(B \cup \mathcal{X})$.

Exercise 2. *Let u and v be terms such that $u =_{\mathsf{E}} v$, and let x be a variable. Show that $t[u/x] =_{\mathsf{E}} t[v/x]$ for any term t. Conclude that the* substitution principle *holds: for any terms s,t,u,v and variable x, $s =_{\mathsf{E}} t$ and $u =_{\mathsf{E}} v$ imply $s[u/x] =_{\mathsf{E}} t[v/x]$.*

1.1.2 Rewrite rules

We assume, for each destructor symbol $f \in \Sigma_d$ of arity n, a set of *reduction rules* of the form $f(u_1, \ldots, u_n) \to u$ where $u, u_1, \ldots, u_n \in \mathcal{T}_c(\mathcal{X})$. From this we define a *computation relation* $\Downarrow \subseteq \mathcal{T}(\mathcal{N}) \times \mathcal{T}_c(\mathcal{N})$ between terms and messages as the least relation satisfying the following conditions:

- for all $n \in \mathcal{N}$, $n \Downarrow n$;
- for all $f \in \Sigma_c$ with $\mathsf{ar}(f) = n$,
  for all $t_1, \ldots, t_n$ and $u_1, \ldots, u_n$ such that $t_i \Downarrow u_i$ for all $i \in [1;n]$,
  $f(t_1, \ldots, t_n) \Downarrow f(u_1, \ldots, u_n)$;
- for all $f \in \Sigma_d$ with reduction rule $f(u_1, \ldots, u_n) \to u$,
  for all t_1, …, t_n and $\theta : \mathcal{X} \to \mathcal{T}_c(\mathcal{N})$ such that $t_i \Downarrow u_i\theta$ for each $i \in [1;n]$,
  $f(t_1, \ldots, t_n) \Downarrow u\theta$;

As for the equational theory, specific applications of our model may call for extra assumptions. For instance, one may impose that the computation relation is deterministic (up to $=_{\mathsf{E}}$) or computable.

- for all t, u and v such that $t \Downarrow u$ and $u =_{\mathsf{E}} v$, $t \Downarrow v$.

We write $t \not\Downarrow$ when there is no message u such that $t \Downarrow u$.

Example 3. *Consider again Σ from ex. 1 but assuming now that $\Sigma_d = \{ \textbf{proj}_1, \textbf{proj}_2 \}$. Assume that* E *contains only the equation* $\textbf{sdec}(\textbf{senc}(x, y), y) \ \mathsf{E} \ x$*, and take the reduction rules*

$$\textbf{proj}_1(\textbf{pair}(x, y)) \to x \ \textit{ and } \ \textbf{proj}_2(\textbf{pair}(x, y)) \to y.$$

As an analogue of what we obtained in the previous example, we have

$$\textbf{proj}_1(\textbf{sdec}(\textbf{senc}(\textbf{pair}(\textbf{ok}, n), k), k)) \Downarrow \textbf{ok}.$$

Observe that the fourth item of the definition of $\Downarrow$ is crucial to obtain this computation, as it is needed to have $\textbf{sdec}(\textbf{senc}(\textbf{pair}(\textbf{ok}, n), k), k) \Downarrow \textbf{pair}(\textbf{ok}, n)$*. We also have* $\textbf{pair}(\textbf{ok}, n) \not\Downarrow \textbf{ok}$*: in fact,* $\textbf{pair}(\textbf{ok}, n) \Downarrow u$ *iff* $u =_{\mathsf{E}} \textbf{pair}(\textbf{ok}, n)$*. Finally, there are terms that cannot be computed, e.g.* $\textbf{proj}_1(\textbf{ok}) \not\Downarrow$ *and* $\textbf{proj}_1(\langle \textbf{ok}, \textbf{proj}_1(\textbf{ok}) \rangle) \not\Downarrow$*.*

Depending on the problem that is considered, it may be more practical to consider only equations and reduction rules. Sometimes, e.g. in Proverif, both are available. In such cases, there is often a choice between equations and reductions, as illustrated in the previous examples for pairing. This choice may involve performance issues but it also affects the adequacy of the modelling of cryptographic primitives. The key difference to keep in mind is that, when t is a term featuring destructors, it is sometimes impossible to obtain a message u such that $t \Downarrow u$: this failure to compute (or failure to eliminate destructors) will lead to different behaviours of the protocol (and attacker). We will see below examples where it makes a difference.

ϟ Note that computation failures can often be detected through equations: for instance, under the equational theory of ex. 2, if t is a message, $\langle \textbf{proj}_1(t), \textbf{proj}_2(t) \rangle =_{\mathsf{E}} t$ holds iff t is of the form $\langle t_1, t_2 \rangle$, i.e. iff $\textbf{proj}_1(t)$ computes successfully.

Example 4. *Asymmetric encryption is generally better represented as a destructor, using binary encryption and decryption symbols as well as a unary public-key symbol* **pub***, and the following reduction rule:*

$$\textbf{adec}(\textbf{aenc}(x, \textbf{pub}(y)), y) \to x$$

Example 5. *More expressive computations can be expressed by ordering the reduction rules associated to a destructor, and requiring that a rule may only be used if the previous ones do not apply. For instance, if* $\textbf{eq} \in \Sigma_d$ *we may consider the following* list *of rules for it:*

$$\begin{array}{lcl} \textbf{eq}(x, x) & \to & \textbf{true} \\ \textbf{eq}(x, y) & \to & \textbf{false} \end{array}$$

Consider two different names n and m, and assume that $\textbf{true} \neq_{\mathsf{E}} \textbf{false}$*. When taking the ordering into account we have* $\textbf{eq}(n, n) \Downarrow b$ *iff* $b =_{\mathsf{E}} \textbf{true}$*. Without the ordering, this is not true since the second rule applies, and thus* $\textbf{eq}(n, n) \Downarrow \textbf{false}$*.*

1.1.3 Renaming

A *renaming* is total applicatio from names to names. Renamings will be noted in the same way as substitutions, implicitly assuming that they behave as the identity where they are not explicitly defined. Their application is also defined analogously. For example, if $\theta = \{n \mapsto m, m \mapsto n, p \mapsto n\}$ and $t = \textbf{pair}(m, p)$, then $t\theta = \textbf{pair}(n, n)$. As shown in that example, a renaming may not be bijective.

Exercise 3. *Assume that* $s =_{\mathsf{E}} t$ *for some* $s, t \in \mathcal{T}(\mathcal{N})$*. Show that* $s\sigma =_{\mathsf{E}} t\sigma$ *for any renaming* σ*.*

Exercise 4. *Consider the variant of computations where reductions rules are ordered. Is it true that* $t \Downarrow u$ *implies* $t\sigma \Downarrow u\sigma$ *for any renaming* σ*? If not, propose an extra assumption under which the claim holds.*

1.2 Processes

Protocols will be modelled using a process algebra in the style of the applied pi-calculus, which itself elaborates on Milner's pi-calculus. Although our presentation differs from its specific description, our calculus is compatible with that of Proverif, the main difference being that we do not treat private channels.

We assume a countably infinite set $\mathcal{C}$ of channels, whose elements will be denoted by c, d, etc. Processes are generated from the following grammar:

$$\begin{array}{rcl|l|l} P, Q & ::= & 0 & (P \mid Q) & !P \\ & \mid & \mathbf{in}(c, x).P & \mathbf{out}(c, u).P & \mathbf{new}\ n.P \\ & \mid & \mathbf{let}\ x = t\ \mathbf{in}\ P\ \mathbf{else}\ Q & & \end{array}$$

where $c \in \mathcal{C}$, $x \in \mathcal{X}$, $n \in \mathcal{N}$, $u \in \mathcal{T}_c(\mathcal{N} \cup \mathcal{X})$ is a constructor term and $t \in \mathcal{T}(\mathcal{N} \cup \mathcal{X})$ is an arbitrary term. Before providing a formal semantics for this language, we describe intuitively what each construct should mean:

- 0 is the process that does nothing.
- $(P \mid Q)$ is the parallel composition of processes P and Q.
- $!P$ is the replication of P, which can be thought of as an infinite parallel composition $(P \mid P \mid P \mid \ldots)$.
- $\mathbf{in}(c, x).P$ is a process that waits for an input on channel c and then behaves as P with x bound to the received message.
- $\mathbf{out}(c, u).P$ outputs a message u on c and then behaves as P.
- **new** $n.P$ creates a new (previously unused) name m and then behaves as P with n replaced by m.
- **let** $x = y$ **in** P **else** Q attempts to evaluate t: upon success, it binds x to the resulting message and continues with P; otherwise, it continues with Q.

This syntax is close but not identical to that of Proverif. We refer the reader to the user manual of the tool for the concrete Proverif syntax.

We will consider terms up to associativity and commutativity of parallel composition, and up to the identification of $P \mid 0$ and P. This means, for instance, that $(P \mid Q) \mid (R \mid 0)$ and $Q \mid (P \mid R)$ are the same process, which we would write more simply (and unambiguously) as $P \mid Q \mid R$.

Notations. We will usually omit the null process, writing e.g. $\mathbf{out}(c, u)$ instead of $\mathbf{out}(c, u).0$ or (**let** $x = t$ **in** P) instead of (**let** $x = t$ **in** P **else** Q). When $u, v \in \mathcal{T}(\mathcal{N} \cup \mathcal{X})$ we write (**if** $u = v$ **then** P **else** Q) for (**let** _ $=$ $\mathbf{eq}(u, v)$ **in** P **else** Q), assuming that **eq** is a destructor defined by the single rule $\mathbf{eq}(x, x) \to x$.

Handling binders. We write $\mathsf{fv}(P)$ for the set of *free variables* of P, i.e. the set of variables that are not bound by an input or a **let** construct. Similarly, we write $\mathsf{fn}(P)$ for the set of *free names* of P, i.e. the set of names that are not bound by a **new** construct. A process P is *closed* if $\mathsf{fv}(P) = \emptyset$ and we will only consider the execution of processes under this condition: this means that when $\mathbf{out}(c, u)$ is emitted, $\mathsf{fv}(u) = \emptyset$, hence u is a message; similarly, when executing **let** $x = t$, we have $t \in \mathcal{T}(\mathcal{N})$, i.e. it is well-defined to ask whether there exists u such that $t \Downarrow u$.

The constructs **in**, **let** and **new** being binders, they induce a notion of α-renaming. We will implicitly consider terms up to it. As is standard in higher-order rewriting, we also assume that substitution is capture avoiding (and hence compatible with α-renaming): this means, for instance, that

$$\begin{aligned} & \big(\textbf{new } n.\ \textbf{out}(c, \textbf{senc}(x,n)).P\big)\{x \mapsto n\} \\ = \quad & \big(\textbf{new } m.\ \textbf{out}(c, \textbf{senc}(x,m)).P\{x \mapsto m\}\big)\{x \mapsto n\} \\ = \quad & \big(\textbf{new } m.\ \textbf{out}(c, \textbf{senc}(n,m)).P\{x \mapsto m\}\{x \mapsto n\}. \end{aligned}$$

1.2.1 Internal reduction

We first endow processes with an operational semantics that expresses how a closed process may execute.

Definition 1 (Internal reduction). *The binary relation $P \rightsquigarrow Q$ is given by the following rules:*

- $\textbf{in}(c,x).P \mid \textbf{out}(c,u).Q \mid R \ \rightsquigarrow\ P\{x \mapsto u\} \mid Q \mid R$
- $(\textbf{let } x = t \textbf{ in } P \textbf{ else } Q) \mid R \ \rightsquigarrow\ P\{x \mapsto u\} \mid R$ *when* $t \Downarrow u$
- $(\textbf{let } x = t \textbf{ in } P \textbf{ else } Q) \mid R \ \rightsquigarrow\ Q \mid R$ *when* $t \not\Downarrow$
- $\textbf{new } n.P \mid R \ \rightsquigarrow\ P\{n \mapsto m\} \mid R$ *when* $m \notin \mathsf{fn}(\textbf{new } n.P, R)$
- $!P \mid R \ \rightsquigarrow\ !P \mid P \mid R$

Remark 1. *The following rules are admissible for the syntactic sugar defined above, when u and v are constructor terms – in practice we will use it when $\Sigma_d = \emptyset$:*

$$\begin{aligned} &\textbf{if } u = v \textbf{ then } P \textbf{ else } Q \mid R \rightsquigarrow P \mid R \qquad \textit{when } u =_{\mathsf{E}} v \\ &\textbf{if } u = v \textbf{ then } P \textbf{ else } Q \mid R \rightsquigarrow Q \mid R \qquad \textit{when } u \neq_{\mathsf{E}} v \end{aligned}$$

The choice of fresh names in the reductions is somewhat arbitrary, as expressed in the following property, where the bijectivity condition is needed even without ordered reduction rules (cf. exercise 4) due to the presence of **else** branches.

Proposition 1. *If $P \rightsquigarrow Q$ then $P\sigma \rightsquigarrow Q\sigma$ for any bijective renaming σ.*

This result will often be used in the case where there is an "undesirable" name $n \in \mathsf{fn}(P) \setminus \mathsf{fn}(Q)$. Then we can swap m with any other name $n \notin \mathsf{fn}(P) \cup \mathsf{fn}(Q)$, using prop. 1 with the permutation $\{n \leftrightarrow m\}$, which gives us

$$P\{n \mapsto m\} = P\{n \leftrightarrow m\} \rightsquigarrow Q\{n \leftrightarrow m\} = Q.$$

We use this notion of computation — which correctly reflects (some) real-world computations — to give a first formal security definition.

Definition 2 (Secrecy). *A process P does not ensure the secrecy of a message s if there exist closed processes A and Q, a channel c and a term s' such that:*

A is the adversary, or attacker. By interacting in a malicious way with P, it attempts to obtain s and emit it on c.

- *terms in A belong to $\mathcal{T}_{\mathsf{pub}}(\mathcal{N})$ and $\mathsf{fn}(P,s) \cap \mathsf{fn}(A) = \emptyset$,*
- *$s =_{\mathsf{E}} s'$ and*
- *$P \mid A \ \rightsquigarrow^*\ \textbf{out}(c,s') \mid Q$ and names chosen in this reduction when reducing* **new** *constructs are never taken in the initial "secret set" $\mathsf{fn}(P,s)$.*

Otherwise, we say that P ensures the secrecy of s.

The condition on free names expresses that the attacker does not know the initial secrets of the protocol. Without it, secrecy would never hold!

Exercise 5. *For each of the following processes, indicate when the secrecy of n is ensured, and exhibit an adversary otherwise:*

- $P_1 = \textbf{new}\ k.\textbf{out}(c, \textbf{senc}(n,k)).\textbf{out}(c,k)$
- $P_2 = \textbf{in}(c,x).\textbf{out}(c, \textbf{senc}(n,x))$
- $P_3 = \textbf{out}(c, \textbf{senc}(n,k)).\textbf{in}(c,x).\textbf{if}\ x = n\ \textbf{then}\ \textbf{out}(c,k)$
- $P_4 = \textbf{in}(c,x).\textbf{let}\ y = \textbf{adec}(x,k)\ \textbf{in}\ \textbf{out}(c,k)\ \textbf{else}\ \textbf{out}(c, \textbf{aenc}(n, \textbf{pub}(k)))$
- $P_5 = !P_4$

Exercise 6. *Show that the condition of def. 2 on the choice of fresh names in the reduction is necessary, but providing an undesirable example that would count as a breach of secrecy without it. Similarly, show the importance of* $\mathsf{fn}(s) \cap \mathsf{fn}(A) = \emptyset$*, i.e. show that the definition would not adequately model secrecy if* $\mathsf{fn}(P)$ *were used instead of* $\mathsf{fn}(P,s)$ *in both places.*

1.2.2 Labelled transitions

In order to analyze the possible interactions of a process with its environment, it is often more convenient to work with labelled transition semantics, as defined next. In the context of security protocols, it will allow us to characterize secrecy without quantifying over all possible adversaries.

We assume another set $\mathcal{W}$ of special variables called *handles* and denoted by w. Handles being variables, they are excluded from closed processes. In the labelled transition system, some terms will represent how the adversary may perform a computation involving messages he obtained from the protocol: these terms, called *recipes*, will belong to $\mathcal{T}_{\mathsf{pub}}(\mathcal{W} \cup \mathcal{N})$ and will be denoted by R, M, N.

Definition 3 (Frame). *A* frame $\vec{n}.\sigma$ *is given by a list of names* $\vec{n}$ *and a finite mapping* $\sigma : \mathcal{W} \to \mathcal{T}_c(\mathcal{N})$*. Frames are denoted by* Φ *or* Ψ*. If* $\Phi = \vec{n}.\sigma$ *is a frame we write* $\mathsf{bn}(\Phi)$ *for* $\vec{n}$*;* $\mathsf{dom}(\Phi)$ *for* $\mathsf{dom}(\sigma)$*;* $\Phi \cup \{w \mapsto u\}$ *for* $\vec{n}.(\sigma \cup \{w \mapsto u\})$*; and* $m.\Phi$ *for* $(m, \vec{n}).\sigma$*.*

Definition 4 (Configuration). *A* configuration *is a pair* (P, Φ) *where* P *is a closed process and* Φ *is a frame. Configurations are denoted by* K*. When* K *is a configuration,* $\Phi(K)$ *denotes its frame.*

We introduce a convenient notation for avoiding heavy freshness conditions on names. Given two objects (terms, processes, frames or sequences of such objects) we write $x \sharp y$ when no name occurs free in both x and y, i.e. $\mathsf{fn}(x) \cap \mathsf{fn}(y) = \emptyset$. For instance, when R is a recipe, $R \sharp (P, \Phi)$ means that $R \in \mathcal{T}_{\mathsf{pub}}(\mathcal{W} \cup \mathcal{N} \setminus \mathsf{fn}(P, \Phi))$.

Definition 5 (Labelled transitions). *The labelled transition relation* $K \xrightarrow{\alpha} K'$*, given by the rules of fig. 1, is a relation between two configurations and an* action α *that may be either*

- *the silent action* τ*, or*
- *the input action* $\textbf{in}(c, R)$ *for some* $c \in \mathcal{C}$ *and* $R \in \mathcal{T}_{\mathsf{pub}}(\mathcal{N} \cup \mathcal{W})$*, or*
- *the output action* $\textbf{out}(c, w)$ *for some* $c \in \mathcal{C}$ *and* $w \in \mathcal{W}$*.*

We define the labelled reflexive transitive closure of $\xrightarrow{\alpha}$ *as follows:* $K_0 \xrightarrow{\mathsf{tr}} K_n$ *when* $\mathsf{tr} = \alpha_1 \ldots \alpha_n$ *and* $K_i \xrightarrow{\alpha_i} K_{i+1}$ *for all* $i \in [1; n]$*.*

In an input, the recipe explains how the environment computes the input message from the current frame. In an output, w *is the handle to which the output message will be associated in the updated frame.*

$$(\mathbf{out}(c,u).P \mid Q, \Phi) \xrightarrow{\mathbf{out}(c,w)} (P \mid Q, \Phi \cup \{w \mapsto u\}) \quad \text{when } w \notin \mathsf{dom}(\Phi)$$

$$(\mathbf{in}(c,x).P \mid Q, \Phi) \xrightarrow{\mathbf{in}(c,R)} (P\{x \mapsto u\} \mid Q, \Phi)$$
$$\text{when } R \in \mathcal{T}_{\mathsf{pub}}(\mathcal{N} \cup \mathsf{dom}(\Phi)),\ R \sharp \mathsf{bn}(\Phi) \text{ and } R\Phi \Downarrow u$$

$$(\mathbf{let}\ x = t\ \mathbf{in}\ P\ \mathbf{else}\ Q \mid R, \Phi) \xrightarrow{\tau} (P\{x \mapsto u\}, \Phi) \qquad \text{when } t \Downarrow u$$

$$(\mathbf{let}\ x = t\ \mathbf{in}\ P\ \mathbf{else}\ Q \mid R, \Phi) \xrightarrow{\tau} (Q, \Phi) \qquad \text{when } t \not\Downarrow$$

$$(\mathbf{new}\ n.P \mid Q, \Phi) \xrightarrow{\tau} (P\{n \mapsto m\} \mid Q, m.\Phi) \text{ if } m \sharp (\mathbf{new}\ n.P, Q, \Phi, \mathsf{bn}(\Phi))$$

$$(!P \mid Q, \Phi) \xrightarrow{\tau} (!P \mid P \mid Q, \Phi)$$

Figure 1: Labelled transitions between configurations

The main novelty here is that, when $K = (P, \Phi)$ performs a labelled transition, communication is not taking place between sub-processes of P. Instead, the transition represents a possible interaction with an hypothetical environment (or attacker, in our context) whose knowledge is represented by Φ. This idea is pushed to the extreme here, and sub-processes of P are not even *allowed* to communicate. The intuitive justification is that, since the attacker can eavesdrop and inject messages, he can in particular mediate internal communications, and we might as well assume that he mediates them all. Formally, adding the internal communication rule would not change the next result.

Proposition 2. *If $K \xrightarrow{\alpha} K'$ and σ is a bijection on names, then $K\sigma \xrightarrow{\alpha\sigma} K'\sigma$.*

We now formulate the analogue of secrecy in the framework of frames and labelled transitions, before establishing that it captures the same idea.

Definition 6 ($\Phi \vdash u$). *A frame Φ allows to* deduce *a message s, written $\Phi \vdash s$, if there exists a recipe R such that $R \sharp \mathsf{bn}(\Phi)$ and $R\Phi \Downarrow s$.*

Proposition 3 (Secrecy). *A process P does not ensure the secrecy of s iff there exist* tr*, P', Φ' such that $(P, \mathsf{fn}(P,s).\emptyset) \xrightarrow{\mathsf{tr}} (P', \Phi')$ and $\Phi' \vdash s$.*

Proof. We first prove that, if there exists an execution $(P, \Phi) \xrightarrow{\mathsf{tr}} (P', \Phi')$ and a recipe $R \sharp \mathsf{bn}(\Phi')$ such that $R\Phi' \Downarrow s$, then there exists a process A containing terms in $\mathcal{T}_{\mathsf{pub}}(\mathsf{dom}(\Phi) \cup \mathcal{N} \setminus \mathsf{bn}(\Phi))$ such that $P \mid A\Phi \rightsquigarrow^* \mathbf{out}(\_, s) \mid \ldots$ with a reduction that does not pick fresh names in $\mathsf{bn}(\Phi)$.

We proceed by induction on tr, essentially translating (tr, R) into an attacker A. If $\mathsf{tr} = \epsilon$ we have $R \sharp \mathsf{bn}(\Phi)$ and $R\Phi \Downarrow s$. We conclude with $A := \mathbf{let}\ x = R\ \mathbf{in}\ \mathbf{out}(c, x)$ since $A\Phi \rightsquigarrow \mathbf{out}(c, s)$.

Assume now that $\mathsf{tr} = \alpha.\mathsf{tr}'$. We have $(P, \Phi) \xrightarrow{\alpha} (P_1, \Phi_1) \xrightarrow{\mathsf{tr}'} (P', \Phi')$ and we proceed by case analysis on the first transition.

- If $\alpha = \mathbf{out}(c, w)$, P is of the form $\mathbf{out}(c, v).Q \mid R$, $P_1 = Q \mid R$ and $\Phi_1 = \Phi \cup \{w \mapsto v\}$ for some $w \notin \mathsf{dom}(\Phi)$. By induction hypothesis on tr' we have an adversary A_1 against P_1 such that $A_1 \sharp \mathsf{bn}(\Phi_1)$. We conclude with $A := \mathbf{in}(c, w).A_1$: we check that $A \sharp \mathsf{bn}(\Phi) = \mathsf{bn}(\Phi_1)$; that terms of A are in $\mathcal{T}_{\mathsf{pub}}(\mathcal{N} \cup \mathsf{dom}(\Phi))$, because $\mathsf{dom}(\Phi) = \mathsf{dom}(\Phi_1) \setminus \{w\}$; and that the expected reduction is possible, because the input of A and the output of P, both on channel c, can be used in a communication rule so that $P \mid A\Phi \rightsquigarrow P_1 \mid A_1\Phi\{w \mapsto v\} = P_1 \mid A_1\Phi_1$.

- If $\alpha = \mathbf{in}(c, R)$ we have $R \sharp \mathsf{bn}(\Phi)$, $R\Phi \Downarrow u$ and

$$(P, \Phi) = (\mathbf{in}(c, x).Q \mid R, \Phi) \xrightarrow{\alpha} (Q\{x \mapsto u\} \mid R, \Phi) = (P_1, \Phi_1).$$

  We obtain by induction hypothesis an adversary A_1 against P_1. The attacker $A := \mathbf{let}\ x = R\ \mathbf{in}\ \mathbf{out}(c, x).A_1$ (for some $x \notin \mathsf{fv}(A_1)$) allows us to conclude. We easily check that it executes well, and that it contains the same free variables as A_1. It also satifies $A \sharp \mathsf{bn}(\Phi)$ since $R \sharp \mathsf{bn}(\Phi)$.

- If the first transition is a name creation, P is of the form $\mathbf{new}\ n.Q \mid R$ and $P_1 = Q\{n \mapsto m\} \mid R$ for some $m \sharp (P, \Phi, \mathsf{bn}(\Phi))$. By induction hypothesis we obtain an adversary A_1 against P_1, satisfying $A_1 \sharp \mathsf{bn}(\Phi_1)$ and hence $A_1 \sharp \mathsf{bn}(\Phi)$ since $\Phi_1 = m.\Phi$. Since $m \notin \mathsf{fn}(A_1)$, we can reduce $P \mid A_1\Phi \rightsquigarrow P_1 \mid A_1\Phi_1$, which allows us to conclude with $A := A_1$.

- The other cases, i.e. **let** evaluations and replication, are similar.

For the other direction, we need to find an inductive characterization of secrecy (def. 2). The problem is the notion of adversary: the condition $A \sharp P$ is not preserved through the reductions of $P \mid A$. Frames get us closer to a solution: in general, when considering a process P and a current frame Φ, it would seem reasonable to consider adversaries of the form $A\Phi$ where $A \sharp P$ and all terms in A belong to $\mathcal{T}_{\mathsf{pub}}(\mathcal{N} \cup \mathcal{W})$. But this form of adversaries is not stable by reduction of **let** constructs, so we need to introduce a slightly more complex notion.

In the proof below, we call adversary a closed process in which some (sub)terms may be decorated by a recipe, which is noted u^R. We say that an adversary A is a Φ-adversary when $R\Phi \Downarrow u$ for all decorated subterms u^R occurring in A. Given a term t with possibly decorated subterms, $R(t)$ is obtained from t by replacing any u^R by R. By extension, if A is an adversary, $R(A)$ is obtained by replacing each u^R by R — note that the resulting object might not be a well-formed process, e.g. since destructors may occur in output terms. For adversaries, freshness conditions are always wrt. $R(A)$, i.e. $A \sharp X$ means $R(A) \sharp X$. Intuitively, we do not consider the computed terms but only the recipes that have been used to obtain them. In the same spirit, we require that any Φ-adversary A is such that $R(A)$ only contains terms in $\mathcal{T}_{\mathsf{pub}}(\mathcal{N} \cup \mathsf{dom}(\Phi))$.

In particular, a Φ-adversary where all decorated terms are of the form $\Phi(w)^w$ is simply a process of the form $A\Phi$.

As an exercise, one can check that the existence of an adversary in the sense of def. 2 is equivalent to the existence of an $\emptyset$-adversary. Thus, it suffices to establish that there exists a trace $(P, \Phi) \xrightarrow{\mathsf{tr}} (P', \Phi')$ such that $\Phi' \vdash s$ whenever there exists a Φ-adversary A such that $A \sharp \mathsf{bn}(\Phi)$ and $P \mid A \rightsquigarrow^* \mathbf{out}(\_, s') \mid \ldots$ for some $s' =_{\mathsf{E}} s$, using a reduction that does not pick fresh names in $\mathsf{bn}(\Phi)$.

We proceed by induction on (the length of) the execution of $P \mid A$.

- If the reduction sequence is empty then $\mathbf{out}(c, s')$ is an immediate parallel sub-process of $P \mid A$: if it belongs to P we conclude with $\mathsf{tr} = \mathbf{out}(c, w)$ for some fresh w since $\Phi \cup \{w \mapsto s'\} \vdash s$ by taking $R = w$; if it belongs to A we have $R(s') \Downarrow s$, hence $\Phi \vdash s$, so we can conclude with $\mathsf{tr} = \epsilon$ because $R(s') \sharp \mathsf{bn}(\Phi)$ in inherited from $A \sharp \mathsf{bn}(\Phi)$.

If t occurs in a Φ-adversary and $t \Downarrow u$, then $R(t)\Phi \Downarrow u$.

- If the first reduction step is internal to A, i.e. $P \mid A \rightsquigarrow P \mid A'$ with $A \rightsquigarrow A'$, we conclude by induction hypothesis on the rest of the reduction involving A', keeping tr unchanged. If the reduction is a name creation, we have assumed that the chosen name satisfies $m \notin \mathsf{bn}(\Phi)$, hence $A' \sharp \mathsf{bn}(\Phi)$. If the reduction is the computation of some **let** $x = t$, we also need to check that A' is still a Φ-adversary: after computing $t \Downarrow u$, we actually replace x by $u^{R(t)}$ to justify the subterm u – indeed, we have $R(t)\Phi \Downarrow u$.

- Assume now that the first step is a communication occurring inside P, exchanging u on channel c. We have

$$(P, \Phi) \xrightarrow{\mathsf{out}(c,w).\mathsf{in}(c,w)} (P', \Phi \cup \{w \mapsto u\})$$

which allows us to conclude by induction hypothesis with A, which is also a $(\Phi \cup \{w \mapsto u\})$-adversary against P'.

- For other reductions internal to P we conclude by induction hypothesis with $\mathsf{tr} = \tau.\mathsf{tr}'$. If P creates a name m, we need A to be a $(m.\Phi)$-adversary satisfying $A \sharp \mathsf{bn}(m.\Phi)$: the former is immediate, the latter comes from the side condition on the reduction of $P \mid A$.

- Assume that, in the first reduction step, a message u is sent by A and received by P on some channel c. Since $A \sharp \mathsf{bn}(\Phi)$ we have $R(u) \sharp \mathsf{bn}(\Phi)$ and thus

$$(P, \Phi) \xrightarrow{\mathsf{in}(c,R(u))} (P'\{x \mapsto u\}, \Phi).$$

We conclude by induction hypothesis using $P'\{x \mapsto u\}$ and A': one easily checks that A' is still a Φ-adversary, and $A' \sharp \mathsf{bn}(\Phi)$.

- Consider finally the case where P sends some message u to A. Then

$$(P, \Phi) \xrightarrow{\mathsf{out}(c,w)} (P', \Phi \cup \{w \mapsto u\})$$

and we conclude by induction hypothesis using P' and $A'\{x \mapsto u^w\}$. $\square$

Exercise 7. *For each process P of exercise 5, exhibit (if it exists) an execution $(P, \emptyset) \xrightarrow{\mathsf{tr}} (Q, \Phi)$ and a recipe $R \sharp \mathsf{bn}(\Phi)$ such that $R\Phi \Downarrow n$.*

2 Verifying secrecy for bounded executions

3 Verifying secrecy for unbounded executions

4 Equivalences

5 Verifying equivalences

6 Advanced topics