5

A cryptographic process calculus

The previous chapter describes how messages exchanged in cryptographic protocols can be represented as *terms*. In this chapter, we discuss how the protocols themselves can be modelled. While the kind of "Alice - Bob" notation used in §2 are convenient for explaining protocols, these notations generally only describe a honest protocol execution, and contain ambiguities. Fundamentally, security protocols are concurrent programs and formalisms for representing such programs do exist. In particular process algebras and calculi have been developed for this purpose. Some "general purpose process algebras", e.g. CSP [Ryan et al., 2000], have indeed been used to reason about security protocols. There also exist dedicated calculi which integrate support for sending messages that use cryptographic primitives. Examples of such dedicated process calculi are CryptoSPA [Focardi and Martinelli, 1999], which extend the CCS process algebra, the spi calculus [Abadi and Gordon, 1999], and the applied pi calculus [Abadi and Fournet, 2001], that both extend the pi calculus. We present here in more details the applied pi calculus. In contrast to the pure pi calculus it is not restricted to communicating names, but processes may output terms that represent messages. One may also note that some people have con-

$P, Q, R :=$ plain processes
0
$P \parallel Q$
$!P$
$\nu n.P$
$\mathsf{if}\ t_1 = t_2\ \mathsf{then}\ P\ \mathsf{else}\ Q$
$\mathsf{in}(u, x).P$
$\mathsf{out}(u, t).P$

Figure 5.1: Syntax: plain processes

sidered the problem of compiling "Alice - Bob" kind notations to more formal models, e.g., Jacquemard et al., 2000, Millen and Denker, 2002, Chevalier and Rusinowitch, 2010, but we will not discuss them here.

5.1 Syntax and informal semantics

We assume a set of names $\mathcal{N}$, a set of variables $\mathcal{X}$, and a signature $\mathcal{F}$ which define the set of terms $T(\mathcal{F}, \mathcal{X}, \mathcal{N})$ equipped with an equational theory E (see §4). The equational theory is left implicit throughout this chapter. Moreover, we rely on a simple sort system that distinguishes channels and basic messages. We distinguish the set $\mathcal{N}_{ch} \subset \mathcal{N}$ of channel names and partition $\mathcal{X} = \mathcal{X}_b \uplus \mathcal{X}_{ch}$ in the set of variables of base sort and variables of channel sort. We suppose that function symbols cannot be applied to variables or names of channel sort, and cannot return terms of that sort. Hence, channels are always atomic: the only terms of channel sort are variables and names of that sort.

The applied pi calculus has two kind of processes: *plain* and *extended* processes. Plain processes are generated by the grammar given in Figure 5.1, where $t, t_1, t_2, \ldots$ range over terms, n over names, x over variables and u is a meta-variable that stands for either a name or a variable of channel sort. The 0 process is the process that does nothing. Parallel composition $P \parallel Q$ models that processes P and Q are executed in parallel. The replication of P, denoted $!P$, allows an unbounded number of copies of P to be spawned. New names can be

A, B, C := extended processes
- P
- $A \parallel B$
- $\nu n.A$
- $\nu x.A$
- $\{^{t}/_{x}\}$

Figure 5.2: Syntax: extended processes

created using the *new* operator νn, which acts as a binder and generates a *restricted* name. The conditional $\mathsf{if}\ t_1 = t_2\ \mathsf{then}\ P\ \mathsf{else}\ Q$ behaves as P whenever $t_1 =_E t_2$ and as Q otherwise. Finally, $\mathsf{in}(u, x).P$ expects an input on channel u that is bound to variable x in P and $\mathsf{out}(u, M).P$ outputs term M on channel u and then behaves as P.

Extended processes are generated by the grammar given in Figure 5.2. They extend plain processes by *active substitutions*, and allow restrictions on both names and variables. An active substitution $\{^{t}/_{x}\}$ allows processes to address a term by a variable. The scope of this access may be restricted using the ν operator on variables. This also allows to define local variables as follows: the construct $\mathsf{let}\ x = t\ \mathsf{in}\ P$ is defined as $\nu x.(P \parallel \{^{t}/_{x}\})$. When the variable x is not restricted, it means that the environment, which represents the attacker, may use x to access the term t. As exemplified in the description of the labelled semantics of the applied pi calculus, these active substitutions are typically used to record the terms output by the processes and represent the attacker knowledge. Given several active substitutions in parallel $\{^{t_1}/_{x_1}\} \parallel \ldots \parallel \{^{t_n}/_{x_n}\}$ we often regroup them in a single substitution $\{^{t_1}/_{x_1}, \ldots, ^{t_n}/_{x_n}\}$. We suppose that these substitutions are cycle free, that there is at most one substitution defined for each variable, and exactly one when this variable is restricted, and substitutions only contain terms of base sort. Given an extended process A we denote by $\phi(A)$ the process obtained by replacing any plain process with 0. $\phi(A)$ is called the frame of the process A. We also note that extended processes must not appear under a replication, an input, an output, or a conditional.

For readability we often omit trailing 0 processes and else 0 branches, e.g. we write

$$\mathsf{if}\ t_1 = t_2\ \mathsf{then}\ \mathsf{out}(c, t)$$

instead of

$$\mathsf{if}\ t_1 = t_2\ \mathsf{then}\ \mathsf{out}(c, t).0\ \mathsf{else}\ 0$$

As usual we define free and bound names for processes, denoted $\mathsf{n}(A)$ and $\mathsf{bn}(A)$. Similarly we define free and bound variables, denoted $\mathsf{fv}(A)$ and $\mathsf{bv}(A)$. The set of bound variables contains all variables that have been bound by an input and the ones that are restricted by the ν operator. For an active substitution $\{^t/_x\}$ the set of free variables contains x in addition to the variables occurring in t.

Finally, we define the notion of *context*. A context is a process with a "hole", often denoted _. Given a context C, we write $C[A]$ for the process obtained by replacing _ with the process A. An evaluation context is a context whose hole is neither under replication, nor input, output or conditional.

5.2 Modelling protocols as processes

Before defining the formal semantics of the applied pi calculus we illustrate how the calculus can be used for modelling security protocols. As an example we consider the Needham-Schroeder public key protocol, introduced in §2. The protocol can be informally described as follows.

1. $A \to B: \quad \mathsf{aenc}(\langle a, n_a\rangle, \mathsf{pk}_b)$
2. $B \to A: \quad \mathsf{aenc}(\langle n_a, n_b\rangle, \mathsf{pk}_a)$
3. $A \to B: \quad \mathsf{aenc}(n_b, \mathsf{pk}_b)$

We assume the previously defined equational theory E_{enc} on signature $\mathcal{F}_{\mathsf{dec}} \cup \mathcal{F}_{\mathsf{std}}$ and model each of the roles A and B by a process. It is important to distinguish between the *role* of the initiator A, modelled by a process, and the agent (a in the above informal description) who is executing the role. To make this distinction explicit we parametrize the processes representing the initiator and responder with the keys of the agents who execute the role.

$$\begin{array}{rcl} P_A(sk_i, pk_r) & \hat{=} & \nu n_a.\ \mathsf{out}(c, \mathsf{aenc}(\langle \mathsf{pk}(sk_i), n_a\rangle, \mathsf{pk}_r)).\\ & & \mathsf{in}(c, x).\\ & & \mathsf{if}\ \mathsf{fst}(\mathsf{adec}(x, sk_i)) = n_a\ \mathsf{then}\\ & & \mathsf{let}\ x_{nb} = \mathsf{snd}(\mathsf{adec}(x, sk_i))\ \mathsf{in}\\ & & \mathsf{out}(c, \mathsf{aenc}(x_{nb}, \mathsf{pk}_r)) \end{array}$$

The process first generates a fresh nonce n_a and then outputs the first message on channel c. Note that for simplicity we assume that the agent's identity is his public key. Next, the initiator is waiting for a message which is going to be bound to the variable x. Using a conditional the initiator checks that the message contains the previously sent nonce n_a. For readability we then create a local variable x_{nb} and store in this variable what is expected to be the nonce generated by the responder.

We can model similarly the responder process P_B.

$$\begin{array}{rcl} P_B(sk_r) & \hat{=} & \mathsf{in}(c, y).\\ & & \mathsf{let}\ pk_i = \mathsf{fst}(\mathsf{adec}(y, sk_r))\ \mathsf{in}\\ & & \mathsf{let}\ y_{na} = \mathsf{snd}(\mathsf{adec}(y, sk_r))\ \mathsf{in}\\ & & \nu n_b.\mathsf{out}(c, \mathsf{aenc}(\langle y_{na}, n_b\rangle, pk_i))\\ & & \mathsf{in}(c, z).\\ & & \mathsf{if}\ \mathsf{adec}(z, sk_r)) = n_b\ \mathsf{then}\ Q \end{array}$$

One may note that P_B only takes a single argument, the responder's private key. The initiator's public key is received during the execution. When the final test succeeds we suppose that the responder continues to execute some task modelled by the process Q.

We can now put the processes together into a process that models the Needham Schroeder public key protocol as a whole.

$$\begin{array}{rl} P^1_{\mathsf{nspk}} \hat{=} \nu sk_a, sk_b.(P_A(sk_a, \mathsf{pk}(sk_b)) \parallel P_B(sk_b) \parallel & \\ \mathsf{out}(c, \mathsf{pk}(sk_a)) \parallel \mathsf{out}(c, \mathsf{pk}(sk_b))) & \end{array}$$

This first version models that a (or more precisely the agent identified by $\mathsf{pk}(sk_a)$) is executing an instance of the role P_A with b (identified by $\mathsf{pk}(sk_b)$). We also output the public keys of a and b to make these available to the adversary.

However, one may notice that the above modeling would miss Lowe's man in the middle attack since this setting does not involve any dishonest agent c. To capture this attack one could explicitly include that a is willing to start a session with the intruder. We suppose that the intruder possesses a secret key sk_c which formally is just a free name.

$$P^2_{\mathsf{nspk}} \hat{=} \nu sk_a, sk_b.(P_A(sk_a, \mathsf{pk}(sk_b)) \parallel P_A(sk_a, \mathsf{pk}(sk_c)) \parallel P_B(sk_b) \parallel \mathsf{out}(c, \mathsf{pk}(sk_a)) \parallel \mathsf{out}(c, \mathsf{pk}(sk_b)))$$

This second version explicitly includes a session started by a with the intruder and indeed captures Lowe's man in the middle attack. However, this situation is not satisfactory, as one does not know a priori with whom agents should start a session. One trick is to leave this choice to the attacker: we add an input that is used to define the public key given to the initiator role.

$$P^3_{\mathsf{nspk}} \hat{=} \nu sk_a, sk_b.(\mathsf{in}(c, x_{pk}).P_A(sk_a, x_{pk}) \parallel P_B(sk_b) \parallel \mathsf{out}(c, \mathsf{pk}(sk_a)) \parallel \mathsf{out}(c, \mathsf{pk}(sk_b)))$$

Now the attacker can just input the public key that suits him best to create an attack. Note that he may input one of the two regular public keys $\mathsf{pk}(sk_a)$ and $\mathsf{pk}(sk_b)$, or any other term, including in particular his own key $\mathsf{pk}(sk_c)$. Note that the attacker may also trigger the agent a to execute a protocol "with himself", i.e., with the public key $\mathsf{pk}(sk_a)$. There exist indeed attacks, sometimes called *reflection attacks*, that rely on this behavior.

This version has still a shortcoming. We only consider one session for each role. Many attacks do however require several parallel sessions of the same role. Millen [1999] has even shown that there is a priori no upper bound on the number of parallel sessions that would avoid all attacks. We therefore add replication.

$$P^4_{\mathsf{nspk}} \hat{=} \nu sk_a, sk_b.(!\mathsf{in}(c, x_{pk}).P_A(sk_a, x_{pk}) \parallel !P_B(sk_b) \parallel \mathsf{out}(c, \mathsf{pk}(sk_a)) \parallel \mathsf{out}(c, \mathsf{pk}(sk_b)))$$

This new version allows a and b to execute an arbitrary number of sessions. Note that a may execute several sessions with the same as well

as different responders. However, this modeling still misses that both initiator and responder may be executed by the same agent. We therefore include explicitly that a and b may execute both roles. Moreover, the above process only allows two honest agents while an attack may a priori require the presence of more honest agents. Therefore we add an additional replication that allows the creation of an arbitrary number of honest private keys, each of which can be used in an arbitrary number of sessions.

$$\begin{aligned} P^5_{\mathsf{nspk}} \hat{=} !\nu sk_a, sk_b.(&!\mathsf{in}(c, x_{pk}).P_A(sk_a, x_{pk}) \parallel !P_B(sk_a) \parallel \\ &!\mathsf{in}(c, x_{pk}).P_A(sk_b, x_{pk}) \parallel !P_B(sk_b) \parallel \\ &\mathsf{out}(c, \mathsf{pk}(sk_a)) \parallel \mathsf{out}(c, \mathsf{pk}(sk_b))) \end{aligned}$$

Observing the symmetric roles of a and b this process can be written more succinctly and elegantly as

$$P^6_{\mathsf{nspk}} \hat{=} !\nu sk.(!\mathsf{in}(c, x_{pk}).P_A(sk, x_{pk}) \parallel !P_B(sk) \parallel \mathsf{out}(c, \mathsf{pk}(sk)))$$

This final modeling allows the adversary to spawn an arbitrary number of instances of P_A and P_B with either the same or different private keys.

5.3 Formal semantics

As the goal is to prove security properties of protocols modelled as processes, we need to define the semantics of the calculus in order to have a precise definition on how a process can be executed.

5.3.1 Operational semantics

We first define the notion of *structural equivalence*. Intuitively, structural equivalence relates identical processes that are simply written in a different way.

Formally, structural equivalence $\equiv$ is the smallest equivalence relation closed under α-conversion of bound names and variables and application of evaluation contexts and such that:

Par-0	$A \parallel 0$	$\equiv$	A	
Par-C	$A \parallel B$	$\equiv$	$B \parallel A$	
Par-A	$(A \parallel B) \parallel C$	$\equiv$	$A \parallel (B \parallel C)$	
Repl	$!P$	$\equiv$	$P \parallel !P$	
New-0	$\nu\, n.\, 0$	$\equiv$	0	
New-Par	$A \parallel \nu\, u.B$	$\equiv$	$\nu\, u.(A \parallel B)$	when $u \notin \mathsf{fv}(A) \cup \mathsf{n}(A)$
New-C	$\nu\, u.\nu\, v.A$	$\equiv$	$\nu\, v.\nu\, u.A$	
Alias	$\nu\, x.\{^{t}/_{x}\}$	$\equiv$	0	
Subst	$\{^{t}/_{x}\} \parallel A$	$\equiv$	$\{^{t}/_{x}\} \parallel A\{^{t}/_{x}\}$	
Rewrite	$\{^{t_1}/_{x}\}$	$\equiv$	$\{^{t_2}/_{x}\}$	when $t_1 =_E t_2$

While most of the above rules are standard the last three rules may require some explanation. Alias allows the creation of a new local variable. Subst allows the application of an active substitution to a process and Rewrite allows to relate two active substitutions modulo the equational theory.

Example 5.1. Let us illustrate these rules by showing that $\mathsf{out}(c, t_1) \equiv \mathsf{out}(c, t_2)$ when $t_1 =_E t_2$.

$\mathsf{out}(c, t_1)$	$\equiv$	$\mathsf{out}(c, t_1) \parallel 0$	by Par-0
	$\equiv$	$\mathsf{out}(c, t_1) \parallel \nu\, x.\{^{t_1}/_{x}\}$	by Alias
	$\equiv$	$\nu\, x.(\mathsf{out}(c, t_1) \parallel \{^{t_1}/_{x}\})$	by New-Par
	$\equiv$	$\nu\, x.(\{^{t_1}/_{x}\} \parallel \mathsf{out}(c, t_1))$	by Par-C
	$\equiv$	$\nu\, x.(\{^{t_1}/_{x}\} \parallel \mathsf{out}(c, x))$	by Subst
	$\equiv$	$\nu\, x.(\{^{t_2}/_{x}\} \parallel \mathsf{out}(c, x))$	by Rewrite
	$\equiv$	$\nu\, x.(\{^{t_2}/_{x}\} \parallel \mathsf{out}(c, t_2))$	by Subst
	$\equiv$	$\nu\, x.(\mathsf{out}(c, t_2) \parallel \{^{t_2}/_{x}\})$	by Par-C
	$\equiv$	$\mathsf{out}(c, t_2) \parallel \nu\, x.\{^{t_2}/_{x}\}$	by New-Par
	$\equiv$	$\mathsf{out}(c, t_2) \parallel 0$	by Alias
	$\equiv$	$\mathsf{out}(c, t_2)$	by Par-0

Note that we also implicitly used the fact that structural equivalence is closed under application of evaluation contexts as we applied some of the rules directly under a context.

One may also note that for any extended process A, we have that $\phi(A) \equiv \nu\tilde{n}.\sigma$ for some sequence of names $\tilde{n}$ and substitution σ. Therefore we can lift static equivalence to processes and we write $A \sim_E B$ whenever $\phi(A) \equiv \nu\tilde{n}_A.\sigma_A$, $\phi(B) \equiv \nu\tilde{n}_B.\sigma_B$, and $\tilde{n}_A.\sigma_A \sim \nu\tilde{n}_B.\sigma_B$.

We can now define how processes interact together. *Internal reduction* is the smallest relation on processes closed under structural equivalence and application of evaluation contexts such that

$$\begin{array}{ll} \textsc{Comm} & \mathsf{out}(c,t).P_1 \parallel \mathsf{in}(c,x).P_2 \rightarrow P_1 \parallel P_2\{^t/_x\} \\ \textsc{Then} & \mathsf{if}\ t = t\ \mathsf{then}\ P\ \mathsf{else}\ Q \rightarrow P \\ \textsc{Else} & \mathsf{if}\ t_1 = t_2\ \mathsf{then}\ P\ \mathsf{else}\ Q \rightarrow Q \\ & \qquad \text{where } t_1, t_2 \text{ are ground and } t_1 \neq_E t_2 \end{array}$$

The first rule (Comm) models communication: whenever a process is ready to output a term t on channel c and another process, running in parallel, is ready to input on channel c, i.e., it starts with $\mathsf{in}(c,x)$ then a communication can take place and x is replaced by t. Rules Then and Else model a conditional. One may note that the Then rule requires syntactic equality of terms ($\mathsf{if}\ t = t$). However as internal reduction is closed under structural equivalence this rule is equivalent to the rule

$$\begin{array}{ll} \textsc{Then'} & \mathsf{if}\ t_1 = t_2\ \mathsf{then}\ P\ \mathsf{else}\ Q \rightarrow P \\ & \qquad \text{where } t_1 =_E t_2 \end{array}$$

using structural equivalence in a similar way as in Example 5.1. One may also note that in the Else rule, contrary to the Then rule we require t_1, t_2 to be ground. This is due to the fact that equality is closed under substitution while disequality is not, e.g. even though $x \neq y$ we have that $x\sigma = y\sigma$ for $\sigma = \{^x/_y\}$.

Example 5.2. We illustrate internal reduction by modelling the honest execution of the Needham Schroeder public key protocol. For simplicity,

we consider a naive model that only considers two honest participants:

$$
\begin{array}{lll}
 & \nu sk_a, sk_b.\ P_A(sk_a, \mathsf{pk}(sk_b)) \parallel P_B(sk_b) & \\
\rightarrow & \nu sk_a, sk_b, n_a, n_b. & \mathsf{in}(c, x). \\
 & & \text{if } \mathsf{fst}(\mathsf{adec}(x, sk_i)) = n_a \text{ then} \\
 & & \text{let } x_{nb} = \mathsf{snd}(\mathsf{adec}(x, sk_i)) \text{ in} \\
 & & \mathsf{out}(c, \mathsf{aenc}(x_{nb}, \mathsf{pk}_r)) \\
 & & \parallel \ \mathsf{out}(c, \mathsf{aenc}(\langle n_a, n_b \rangle, \mathsf{pk}(sk_a))) \\
 & & \mathsf{in}(c, z). \\
 & & \text{if } \mathsf{adec}(z, sk_b)) = n_b \text{ then } Q \\
\rightarrow & \nu sk_a, sk_b, n_a, n_b. & \text{if } n_a = n_a \text{ then} \\
 & & \mathsf{out}(c, \mathsf{aenc}(n_b, \mathsf{pk}_r)) \\
 & & \parallel \ \mathsf{in}(c, z). \\
 & & \text{if } \mathsf{adec}(z, sk_b)) = n_b \text{ then } Q \\
\rightarrow & \nu sk_a, sk_b, n_a, n_b. & \mathsf{out}(c, \mathsf{aenc}(n_b, \mathsf{pk}_r)) \\
 & & \parallel \ \mathsf{in}(c, z). \\
 & & \text{if } \mathsf{adec}(z, sk_b)) = n_b \text{ then } Q \\
\rightarrow & \nu sk_a, sk_b, n_a, n_b. & \text{if } n_b = n_b \text{ then } Q \\
\rightarrow & \nu sk_a, sk_b, n_a, n_b. & Q
\end{array}
$$

5.3.2 Observational equivalence

In order to model security properties we often rely on the notion of *observational equivalence*. Examples of security properties relying on observational equivalence are provided in §6. Intuitively, two processes are observationally equivalent if they cannot be distinguished by an attacker. Here the attacker may be an arbitrary process written in the applied pi calculus. The formal definition of observational equivalence uses the concept of *barbs*: we write $A \Downarrow a$ if process A is able to send a message on channel a, i.e. $A \rightarrow^* C[\mathsf{out}(a, t).P]$ for some evaluation context $C[\_]$ that does not bind a, some process P and term t.

Definition 5.1. Observational equivalence, denoted $\approx$, is the largest symmetric relation $\mathcal{R}$ on closed extended processes with same domain such that if $A\ \mathcal{R}\ B$ then

1. if $A \Downarrow a$ then $B \Downarrow a$;
2. if $A \to^* A'$ then there exists B' such that $B \to^* B'$ and $A' \mathcal{R} B'$;
3. for all closing evaluation context $C[\_]$ we have that $C[A] \mathcal{R} C[B]$.

Intuitively, the aim of the attacker (modelled by the context $C[\_]$) is to output on channel a whenever he believes that he interacts with A and not with B.

Example 5.3. Consider the following two processes A and B.

$$\begin{array}{rcl} A & = & \mathsf{in}(c, x).\ \mathsf{if}\ x = 0\ \mathsf{then}\ \mathsf{out}(c, 1) \\ B & = & \mathsf{in}(c, x).\ \mathsf{if}\ x = 0\ \mathsf{then}\ \mathsf{out}(c, 0) \end{array}$$

where 0 and 1 are constants. These two processes are *not* observationally equivalent, i.e., $A \not\approx B$. A witness of the non-equivalence is for instance provided by the context

$$C[\_] = \mathsf{out}(c, 1).\ \mathsf{in}(c, y).\ \mathsf{if}\ y = 1\ \mathsf{then}\ \mathsf{out}(a, 1) \parallel \_$$

We indeed have that $C[A] \to^* \mathsf{out}(a, 1)$ and therefore $C[A] \Downarrow a$ while $C[B]$ can never emit on a.

Example 5.4. As another example consider the following processes A and B

$$\begin{array}{rcl} A & = & \mathsf{in}(c, x).\ \nu n.\ \mathsf{out}(c, h(n)) \\ B & = & \mathsf{in}(c, x).\ \nu n.\ \mathsf{out}(c, h(\langle x, n \rangle)) \end{array}$$

where h is a free symbol, i.e., not appearing in the equational theory. One may think of h as modeling a hash function. These two processes are observationally equivalent, i.e. $A \approx B$ even though this is not straightforward to prove.

5.3.3 Labelled semantics and labelled bisimulation

In the previous example we presented two processes A and B that are observationally equivalent. Showing observational equivalence formally is however tricky as it requires to show that the processes behave in the same way *for all* context $C[\_]$. This universal quantification over

contexts is generally difficult to reason about and motivates the introduction of a labelled semantics, which allows processes to directly interact with the environment.

The *labelled operational semantics* defines the relation $\xrightarrow{\alpha}$ where α is either $\mathsf{in}(a,t)$ (a is a channel name and t is a term that can contain names and variables), or $\nu x.\mathsf{out}(a,x)$ (x is a variable of base type), or $\mathsf{out}(a,c)$ or $\nu c.\mathsf{out}(a,c)$ (c is a channel name). $\xrightarrow{\alpha}$ extends the internal reduction ($\rightarrow$) by the following rules:

$$\textsc{In} \qquad \mathsf{in}(a,x).P \xrightarrow{in(a,t)} P\{^t/_x\}$$

$$\textsc{Out-Ch} \qquad \mathsf{out}(a,c).P \xrightarrow{out(a,c)} P$$

$$\textsc{Open-Ch} \qquad \frac{A \xrightarrow{out(a,c)} A' \qquad c \neq a}{\nu c.A \xrightarrow{\nu c.\mathsf{out}(a,c)} A'}$$

$$\textsc{Out-T} \qquad \mathsf{out}(a,t).P \xrightarrow{\nu x.out(a,x)} P \mid \{^t/_x\} \qquad x \notin \mathsf{fv}(P) \cup \mathsf{fv}(t)$$

$$\textsc{Scope} \qquad \frac{A \xrightarrow{\alpha} A' \qquad u \text{ does not occur in } \alpha}{\nu u.A \xrightarrow{\alpha} \nu u.A'}$$

$$\textsc{Par} \qquad \frac{A \xrightarrow{\alpha} A' \quad bv(\alpha) \cap fv(B) = bn(\alpha) \cap fn(B) = \emptyset}{A \mid B \xrightarrow{\alpha} A' \mid B}$$

$$\textsc{Struct} \qquad \frac{A \equiv B \quad B \xrightarrow{\alpha} B' \quad A' \equiv B'}{A \xrightarrow{\alpha} A'}$$

The rule In allows the environment to input a term and annotates this transition by the label $in(a,t)$. Intuitively, t is the *recipe* (as in §4.2.1) that allows to deduce the term that is input. The fact that recipes do not contain restricted names is enforced by the rule Scope. We distinguish different ways of outputting terms. The distinction is due to the fact that channel names are handled differently from terms of base type to ensure that they do not appear in the frame. The rule Out-Ch simply outputs a public channel name. Open-Ch allows to output a private channel name: this channel name is *opened* and be-

comes public by removing the "νc" specified in the label $\nu c.out(a, c)$. The side-condition $c \neq a$ simply ensures that one cannot open a private channel, by outputting it on itself. The rule OUT-T allows to output terms of base type. Terms are output *by reference*, i.e., an output creates an entry $\{^t/_x\}$ in the frame, allowing the environment to address t through the variable x. The label $\nu x.out(c, x)$ indicates that x is a fresh variable that has been "opened" and can now be used by the environment. SCOPE and PAR are used to close the labelled reduction under evaluation context, provided these contexts do not interfere with the bound names and variables. We provide examples below illustrating the importance of these side-conditions. Finally, the rule STRUCT states that labelled reduction is closed under structural equivalence.

Remark 5.1. Our definition of labelled semantics slightly differs from the definition given in Abadi and Fournet, 2001. We prefer this presentation as it clarifies the distinction between channel names and terms of base type. Delaune et al. 2010a show that observational equivalence coincides for both semantics.

Example 5.5. Let A be the following process

$$A = \nu s.\ \mathsf{out}(c, \mathsf{enc}(s, k)).\ \mathsf{in}(c, y).\ \mathsf{if}\ y = s\ \mathsf{then}\ P$$

where $x, y \notin \mathsf{fv}(P)$. This models a simple challenge-response protocol: the protocol outputs a secret s encrypted with a key k and only proceeds if it receives s. However, the key k has not been declared private. Therefore an attacker can indeed provide s through the following transition sequence.

$$A \xrightarrow{\nu x.out(c,x)} \nu s.\ A_1 \xrightarrow{in(c,\mathsf{dec}(x,k))} \nu s.\ A_2 \to P$$

where

$$\begin{array}{rcl} A_1 & = & \mathsf{in}(c, y).\ \mathsf{if}\ y = s\ \mathsf{then}\ P \parallel \{^{\mathsf{enc}(s,k)}/_x\} \\ A_2 & = & \mathsf{if}\ \mathsf{dec}(x, k) = s\ \mathsf{then}\ P \parallel \{^{\mathsf{enc}(s,k)}/_x\} \end{array}$$

The step $\nu s.\ A_1 \xrightarrow{in(c,\mathsf{dec}(x,k))} \nu s.\ A_2$ can be shown to be a valid transition as follows.

$$\dfrac{\dfrac{\dfrac{}{\mathsf{in}(c,y).\ \mathsf{if}\ y=s\ \mathsf{then}\ P \xrightarrow{in(c,\mathsf{dec}(x,k))} \mathsf{if}\ \mathsf{dec}(x,k)=s\ \mathsf{then}\ P}\ \textsc{In}}{A_1 \xrightarrow{in(c,\mathsf{dec}(x,k))} A_2}\ \textsc{Par}}{\nu s.\ A_1 \xrightarrow{in(c,\mathsf{dec}(x,k))} \nu s.\ A_2}\ \textsc{Scope}$$

We may now consider the process $\nu k.\ A$, which uses a private key k for the challenge-response. Then we have that

$$\nu k.\ A \xrightarrow{\nu x.out(c,x)} \nu k.\ \nu s.\ A_1$$

However the transition

$$\nu k.\ A_1 \xrightarrow{in(c,\mathsf{dec}(x,k))} \nu k.\ \nu s.\ A_2$$

is *not* valid. In particular, the above proof that $\nu s.\ A_1 \xrightarrow{in(c,\mathsf{dec}(x,k))} \nu s.\ A_2$ cannot be extended using the rule Scope: the rule's side condition is violated because k does appear in the label $in(c,\mathsf{dec}(x,k))$. In this way we ensure that the attacker may not directly use restricted names in the terms that are input. In other words, only deducible terms may be sent by the attacker.

As discussed above proving observational equivalence is particularly tricky due to the universal quantification over all contexts. Therefore we now introduce a *labelled bisimulation*.

Definition 5.2. *Labelled bisimilarity*, denoted $\approx_\ell$, is the largest symmetric relation $\mathcal{R}$ on closed extended processes, such that if $A\mathcal{R}B$ then we have

- $A \sim B$;
- if $A \to A'$ then there exists B' such that $B \to^* B'$ and $A'\mathcal{R}B'$;
- if $A \xrightarrow{\alpha} A'$ and $\mathsf{fv}(\alpha) \subseteq \mathsf{Dom}(A)$ and $\mathsf{bn}(\alpha) \cap \mathsf{n}(B) = \emptyset$ then there exists B' such that $B \to^* \xrightarrow{\alpha} \to^* B$ and $A'\mathcal{R}B'$;

The conditions 2 and 3 correspond to the standard definition of (weak) bisimulation. The first condition requires that the processes are statically equivalent, i.e., the attacker cannot distinguish the sequence of terms output so far by the processes.

Abadi and Fournet [2001] show that labelled bisimulation and observational equivalence coincide.

Theorem 5.1 ([Abadi and Fournet, 2001])**.** Let A and B be two closed extended processes. $A \approx B$ if and only if $A \approx_\ell B$.

We can therefore use labelled bisimulation as a proof technique to show that two processes are (or are not) observationally equivalent. Labelled bisimulation indeed avoids the universal quantification over all possible contexts. One may nevertheless notice that the transition system defined by the labelled reduction relation is infinite branching as the input rule may allow an infinite number of possible terms to be sent by the adversary.

We now revisit Examples 5.3 and 5.4.

Example 5.6. Recall the processes

$$\begin{aligned} A &= \mathsf{in}(c,x).\ \mathsf{if}\ x = 0\ \mathsf{then}\ \mathsf{out}(c,1) \\ B &= \mathsf{in}(c,x).\ \mathsf{if}\ x = 0\ \mathsf{then}\ \mathsf{out}(c,0) \end{aligned}$$

defined in Example 5.3. To show that these processes are not observationally equivalent we provided a context that distinguishes them. We provide an alternate way to prove this result using labelled bisimulation.

We proceed by contradiction. Suppose that $A \approx_\ell B$. We have that $A \xrightarrow{in(c,0)} \to \xrightarrow{\nu x.out(c,x)} A'$ where $A' = \{^1/_x\}$. Then, because $A \approx_\ell B$ there must exist B' such that $B \xrightarrow{in(c,0)} \to^* \xrightarrow{\nu x.out(c,x)} \to^* B'$ and $A' \approx_\ell B'$. In the given example the process B', such that $B \xrightarrow{in(c,0} \to^* \xrightarrow{\nu x.out(c,x)} \to^* B'$ is uniquely defined (up to $\equiv$) as $\{^0/_x\}$. However, $A' \not\sim B'$, violating condition 1 of Definition 5.2, hence contradicting that $A \approx_\ell B$.

Example 5.7. In Example 5.4 we defined processes

$$\begin{aligned} A &= \mathsf{in}(c,x).\ \nu n.\ \mathsf{out}(c,h(n)) \\ B &= \mathsf{in}(c,x).\ \nu n.\ \mathsf{out}(c,h(\langle x,n\rangle)) \end{aligned}$$

which we claimed to be observationally equivalent. We now define the relation $\mathcal{R}$ on closed extended processes as

$$\begin{aligned}\mathcal{R} = (A, B) \cup \{(A', B') \mid\ & A' \equiv \nu n.\ \mathsf{out}(c, h(n)),\\ & B' \equiv \nu n.\ \mathsf{out}(c, h(\langle t, n\rangle)),\\ & t \text{ ground}\}\\ \cup \{(A', B') \mid\ & A' \equiv \nu n.\ \{^{h(n)}/_{x}\}\\ & B' \equiv \nu n.\ \{^{h(\langle t,n\rangle)}/_{x}\}\\ & t \text{ ground}, x \in \mathcal{X}\}\end{aligned}$$

Provided that for any ground term t we have that

$$\nu n.\ \{^{h(n)}/_{x}\} \sim \nu n.\ \{^{h(\langle t,n\rangle)}/_{x}\}$$

it is easy to verify that $\mathcal{R}$ satisfies the 3 conditions of Definition 5.2. Hence, as $A\ \mathcal{R}\ B$ we have that $A \approx_\ell B$. We see that labelled bisimulation can be used to reduce the proof of observational equivalence to a proof of a (generally infinite) family of static equivalences—in this example the static equivalence must hold for all ground terms t.