MPRI Lecture Notes

Course 2-30

Cryptographic protocols
Formal and Computational Proofs

Authors :

- Bruno Blanchet
- Hubert Comon-Lundh
- Stéphanie Delaune
- Cédric Fournet
- Steve Kremer
- David Pointcheval

Contents

Part I

Modelling Protocols and Security Properties

Chapter 2

An Introductory Example

We start with the well-known example of the so-called "Needham-Schroeder public-key protocol" [212], that has been designed in 1978 and for which an attack was found in 1996 by G. Lowe [193], using formal methods.

2.1 An Informal Description

The protocol is a so-called "mutual authentication protocol". Two parties A and B wish to agree on some value, *e.g.* they wish to establish a shared secret that they will use later for fast confidential communication. The parties A and B only use a public communication channel (for instance a postal service, Internet or a mobile phone). The transport of the messages on such channels is insecure. Indeed, a malicious agent might intercept the letter (resp. message) look at its content and possibly replace it with another message or even simply destroy it.

In order to secure their communication, the agents use lockers (or encryption). We consider here public-key encryption: the lockers can be reproduced and distributed, but the key to open them is owned by a single person. Encrypting a message m with the public key of A is written $\{m\}_{\mathsf{pk}(A)}$ whereas concatenating two messages m_1 and m_2 is written $\langle m_1, m_2\rangle$. An informal description of the protocol in the so-called Alice-Bob notation is given in Figure 2.1.

$$
\begin{array}{ll}
1.\ A \to B: & \{\langle A, N_A\rangle\}_{\mathsf{pk}(B)} \\
2.\ B \to A: & \{\langle N_A, N_B\rangle\}_{\mathsf{pk}(A)} \\
3.\ A \to B: & \{N_B\}_{\mathsf{pk}(B)}
\end{array}
$$

Figure 2.1: Informal description of the Needham-Schroeder public key protocol

Description. First the agent A encrypts a nonce N_A, *i.e.* a random number freshly generated, and her identity with the public key of B and sends it on the public channel (message 1). Only the agent B, who owes the corresponding private key can open this message. Upon reception, he gets N_A, generates his own nonce N_B and sends back the pair encrypted with the public key of A (message 2). Only the agent A is able to open this message. Furthermore, since only B was able to get N_A, inserting N_A in the plaintext is a witness that it comes from the agent B. Finally, A, after decrypting, checks that the first component is N_A and retrieves the second component N_B. As an acknowledgement, she sends back N_B encrypted by the public key of B (message 3). When B receives this message, he checks that the content is N_B. If this succeeds, it is claimed that, if the agents A and B are honest, then both parties agreed on the nonces N_A and N_B (they share these values). Moreover, these values are secret: they are only known by the agents A and B.

$$
\begin{array}{llll}
1.\ A \to D: & \{\langle A, N_A\rangle\}_{\mathsf{pk}(D)} & & \\
 & & 1'.\ D(A) \to B: & \{\langle A, N_A\rangle\}_{\mathsf{pk}(B)} \\
 & & 2'.\ B \to A: & \{\langle N_A, N_B\rangle\}_{\mathsf{pk}(A)} \\
2.\ B \to A: & \{\langle N_A, N_B\rangle\}_{\mathsf{pk}(A)} & & \\
3.\ A \to D: & \{N_B\}_{\mathsf{pk}(D)} & & \\
 & & 3'.\ D(A) \to B: & \{N_B\}_{\mathsf{pk}(B)}
\end{array}
$$

Figure 2.2: Attack on the Needham-Schroeder public key protocol

Attack. Actually, an attack was found in 1996 by G. Lowe [193] on the Needham-Schroeder public-key protocol. The attack described in Figure 2.2 relies on the fact that the protocol can be used by several parties. Moreover, we have to assume that an honest agent A starts a session of the protocol with a dishonest agent D (message 1). Then D, impersonating A, sends a message to B, starting another instance of the protocol (message $1'$). When B receives this message, supposedly coming from A, he answers (messages $2'$ & 2). The agent A believes this reply comes from C, hence she continues the protocol (message 3). Now, the dishonest agent D decrypts the ciphertext and learn the nonce N_B. Finally, D is able to send the expected reply to B (message $3'$). At this stage, two instances of the protocol have been completed with success. In the second instance B believes that he is communicating with A: contrarily to what is expected, A and B do not agree on N_B. Moreover, N_B is not a secret shared only between A and B.

Fixed version of the protocol. It has been proposed to fix the protocol by including the respondent's identity in the response (see Figure 2.3).

$$
\begin{array}{ll}
1.\ A \to B: & \{\langle A, N_A\rangle\}_{\mathsf{pk}(B)} \\
2.\ B \to A: & \{\langle\langle N_A, N_B\rangle, B\rangle\}_{\mathsf{pk}(A)} \\
3.\ A \to B: & \{N_B\}_{\mathsf{pk}(B)}
\end{array}
$$

Figure 2.3: Description of the Needham-Schroeder-Lowe protocol

The attack described above cannot be mounted in the corrected version of the protocol. Actually, it is reported in [193] that the technique that permitted to find the Lowe attack on the Needham-Schroeder public key protocol found no attack on this protocol.

2.2 A More Formal Analysis

The Alice-Bob notation is a semi-formal notation that specifies the conversation between the agents. We have to make more precise the view of each agent. This amounts specifying the concurrent programs that are executed by each party. One has also to be precise when specifying how a message is processed by an agent. In particular, what parts of a received message are checked by the agent? What are the actions performed by the agent to compute the answer?

A classical way to model protocols is to use a process algebra. However, in order to model the messages that are exchanged, we need a process algebra that allows processes to send first-order terms build over a signature, names and variables. These terms model messages that are exchanged during a protocol.

Example 2.1 *Consider for example the signature* $\Sigma = \{\{\_\}_\_, \mathsf{pk}\_, \mathsf{sk}(\_), \mathsf{dec}, \langle \_, \_\rangle, \mathsf{proj}_1, \mathsf{proj}_2\}$ *which contains three binary function symbols modelling asymmetric encryption, decryption, and pairing, and four unary function symbols modelling projections, public key and private key. The* *signature is equipped with an equational theory and we interpret equality up to this theory. For instance the theory*

$$\mathsf{dec}(\{x\}_{\mathsf{pk}(y)}, \mathsf{sk}(y)) = x,\ \mathsf{proj}_1(\langle x_1, x_2\rangle) = x_1,\ \textit{and}\ \mathsf{proj}_2(\langle x_1, x_2\rangle) = x_2.$$

models that decryption and encryption cancel out whenever suitable keys are used. One can also retrieves the first (resp. second) component of a pair.

Processes $P, Q, R, \ldots$ are constructed as follows. The process $\mathsf{new}\ N.P$ restricts the name N in P and can for instance be used to model that N is a fresh random number. $\mathsf{in}(c, x).P$ models the input of a term on a channel c, which is then substituted for x in process P. $\mathsf{out}(c, t)$ outputs a term t on a channel c. The conditional $\mathsf{if}\ M = N\ \mathsf{then}\ P\ \mathsf{else}\ Q$ behaves as P when M and N are equal modulo the equational theory and behaves as Q otherwise.

The program (or process) that is executed by an agent, say a, who wants to initiate a session of the Needham-Schroeder protocol with another agent b is as follows:

$A(a,b)$	$\hat{=}$	$\mathsf{new}\ N_a.$	`a generates a fresh message` N_a
		$\mathsf{out}(c, \{a, N_a\}_{\mathsf{pk}(b)}).$	`the message is sent on the channel` c
		$\mathsf{in}(c, x).$	`a is waiting for an input on` c
		$\mathsf{let}\ x_0 = \mathsf{dec}(x, \mathsf{sk}(a))\ \mathsf{in}$	`a tries to decrypt the message`
		$\mathsf{if}\ \mathsf{proj}_1(x_0) = N_a\ \mathsf{then}$	`a checks that the first component is` N_a
		$\mathsf{let}\ x_1 = \mathsf{proj}_2(x_0)\ \mathsf{in}$	`a retrieves the second component`
		$\mathsf{out}(c, \{x_1\}_{\mathsf{pk}(b)})$	`a sends her answer on` c

Note that we use variables for the unknown components of messages. These variables can be (a priori) replaced by any message, provided that the attacker can build it and that it is accepted by the agent. In the program described above, if the decryption fails or if the first component of the message received by a is not equal to N_a, then a will abort the protocol.

Similarly, we have to write the program that is executed by an agent, say b, who has to answer to the messages sent by the initiator of the protocol. This program may look like this:

$B(a,b)$	$\hat{=}$	$\mathsf{in}(c, y).$	`b is waiting for an input on` c
		$\mathsf{let}\ (a, y_0) = \mathsf{dec}(y, \mathsf{sk}(b))\ \mathsf{in}$	`b tries to decrypt it and then retrieves the second component of the plaintext`
		$\mathsf{new}\ N_b.$	`b generated a fresh random number` N_b
		$\mathsf{out}(c, \{y_0, N_b\}_{\mathsf{pk}(a)}).$	`b sends his answer on the channel` c
		$\mathsf{in}(c, y').$	`b is waiting for an input on` c
		$\mathsf{if}\ \mathsf{dec}(y', \mathsf{sk}(b)) = N_b\ \mathsf{then}\ \mathsf{Ok}.$	`b tries to decrypt the message and he checks whether its content is` N_b `or not`

The (weak) secrecy property states for instance that, if a, b are honest (their secret keys are unknown to the environment), then, when the process $B(a, b)$ reaches the Ok state, N_b is unknown to the environment. We will also see later how to formalise agreement properties. The "environment knowledge" is actually a component of the description of the global state of the network. Basically, all messages that can be built from the public data and the messages that have been sent are in the knowledge of the environment.

Any number of copies of A and B (with any parameter values) are running concurrently in a hostile environment. Such a hostile environment is modelled by any process that may receive and emit on public channels. We also assume that such an environment owes as many public/private key pairs as it wishes (compromised agents), an agent may also generate new values when needed. The only restrictions on the environment is on the way it may construct new messages: the encryption and decryption functions, as well as public keys are assumed to be known from the environment. However no private key (besides those that it generates) is known. We exhibit now a process that will yield the attack, assuming that the agent d is a dishonest (or compromised) agent who leaked his secret key:

$P \mathrel{\hat{=}}$	$\mathsf{in}(c, z_1)$.	d `receives a message (from` a`)`
	$\mathsf{let}\ \langle a, z_1' \rangle = \mathsf{dec}(z_1, \mathsf{sk}(d))\ \mathsf{in}$	d `decrypts it`
	$\mathsf{out}(c, \{\langle a, z_1' \rangle\}_{\mathsf{pk}(b)})$.	d `sends the plaintext encrypted with` $\mathsf{pk}(b)$
	$\mathsf{in}(c, z_2).\mathsf{out}(c, z_2)$.	d `forwards to` a `the answer he obtained from` b
	$\mathsf{in}(c, z_3)$.	d `receives the answer from` a
	$\mathsf{let}\ z_3' = \mathsf{dec}(z_3, \mathsf{sk}(d))\ \mathsf{in}$	d `decrypts it and learn` N_b
	$\mathsf{out}(c, \{z_3'\}_{\mathsf{pk}(b)})$.	d `sends the expected message` $\{N_b\}_{\mathsf{pk}(b)}$ `to` b.

The Needham-Schroeder-Lowe protocol has been proved secure in several formal models close to the one we have sketched in this section [72, 24].

2.3 An Attack on the Fixed Version of the Protocol

Up to now, the encryption is a black-box: nothing can be learnt on a plaintext from a ciphertext and two ciphertexts are unrelated.

Consider however a simple El-Gamal encryption scheme. Roughly (we skip here the group choice for instance), the encryption scheme is given by a cyclic group G of order q and generator g; these parameters are public. Each agent a may choose randomly a secret key $\mathsf{sk}(a)$ and publish the corresponding public key $\mathsf{pk}(a) = g^{\mathsf{sk}(a)}$. Given a message m (assume for simplicity that it is an element $g^{m'}$ of the group), encrypting m with the public key $\mathsf{pk}(a)$ consists in drawing a random number r and letting $\{m\}_{\mathsf{pk}(a)} = (\mathsf{pk}(a)^r \times g^{m'}, g^r)$. Decrypting the message consists in raising g^r to the power $\mathsf{sk}(a)$ and dividing the first component of the pair by $g^{r \times \mathsf{sk}(a)}$. We have that:

$$[\mathsf{pk}(a)^r \times g^{m'}]/(g^r)^{\mathsf{sk}(a)} = [(g^{\mathsf{sk}(a)})^r \times g^{m'}]/(g^r)^{\mathsf{sk}(a)} = g^{m'} = m.$$

This means that this encryption scheme satisfies the equation $\mathsf{dec}(\{x\}_{\mathsf{pk}(y)}, \mathsf{sk}(y)) = x$. However, as we will see, this encryption scheme also satisfies some other properties that are not taken into account in our previous formal analysis.

Attack. Assume now that we are using such an encryption scheme in the Needham-Schroeder-Lowe protocol and that pairing two group elements $m_1 = g^{m_1'}$ and $m_2 = g^{m_2'}$ is performed in a naive way: $\langle m_1, m_2 \rangle$ is mapped to $g^{m_1' + 2^{|m_1'|} \times m_2'}$ (*i.e.* concatenating the binary representations of the messages m_1' and m_2'). In such a case, an attack can be mounted on the protocol (see Figure 2.4).

Actually, the attack starts as before. We assume that the honest agent a is starting a session with a dishonest party d. Then d decrypts the message and re-encrypt it with the public key of b. The honest party b replies sending the expected message $\{\langle\langle N_a, N_b \rangle, b \rangle\}_{\mathsf{pk}(a)}$. The attacker intercepts this message. Note that the attacker can not simply forward it to a since it does not have the expected form. The attacker intercepts $\{\langle\langle N_a, N_b \rangle, b \rangle\}_{\mathsf{pk}(a)}$, *i.e.* $(\mathsf{pk}(a)^r \times g^{N_a + 2^\alpha \times N_b + 2^{2\alpha} \times b}, g^r)$ where α is the length of a nonce. The attacker knows g, α, b, hence he can compute $g^{-2^{2\alpha} \times b} \times g^{2^{2\alpha} \times d}$ and multiply the first component, yielding $\{\langle\langle N_a, N_b \rangle, d \rangle\}_{\mathsf{pk}(a)}$. Then the attack can go on as before: a replies by sending $\{N_b\}_{\mathsf{pk}(d)}$ and the attacker sends $\{N_b\}_{\mathsf{pk}(b)}$ to b, impersonating a.

This example is however a toy example since pairing could be implemented in another way. In [251] there is a real attack that is only based on weaknesses of the El Gamal encryption scheme. In particular, the attack does not dependent on how pairing is implemented.

1. $a \to d:\ \{\langle a, N_a\rangle\}_{\mathsf{pk}(d)}$
   $1'.\ d(a) \to b:\ \{\langle a, N_a\rangle\}_{\mathsf{pk}(b)}$
   $2'.\ b \to a:\ \{\langle\langle N_a, N_b\rangle, b\rangle\}_{\mathsf{pk}(a)} = (g^{N_a+2^\alpha\times N_b+2^{2\alpha}\times b} \times \mathsf{pk}(a)^r, g^r)$

d intercepts this message, and computes
$$[g^{N_a+2^\alpha\times N_b+2^{2\alpha}\times b} \times \mathsf{pk}(a)^r] \times g^{-2^{2\alpha}\times b} \times g^{2^{2\alpha}\times d} = g^{N_a+2^\alpha\times N_b+2^{2\alpha}\times d} \times \mathsf{pk}(a)^r$$

2. $d \to a:\ \{\langle\langle N_a, N_b\rangle, d\rangle\}_{\mathsf{pk}(a)} = (g^{N_a+2^\alpha\times N_b+2^{2\alpha}\times d} \times \mathsf{pk}(a)^r, g^r)$
3. $a \to d:\ \{N_b\}_{\mathsf{pk}(d)}$
   $3'.\ d \to b:\ \{N_b\}_{\mathsf{pk}(d)}$

Figure 2.4: Attack on the Needham-Schroeder-Lowe protocol with El-Gamal encryption.

This shows that the formal analysis only proves the security in a formal model, that might not be faithful. Here, the formal analysis assumed a model in which it is not possible to forge a ciphertext from another ciphertext, without decrypting/encrypting. This property is known as *non-malleability*, which is not satisfied by the El Gamal encryption scheme.

2.4 Further Readings

A survey by Clark and Jacob [106] describes several authentication protocols and outlines also the methods that have been used to analyse them. In addition, it provides a summary of the ways in which protocols have been found to fail. The purpose of the SPORE web page [1] is to continue on-line the seminal work of Clark and Jacob, updating their base of security protocols.

As you have seen, some protocols (or some attacks) rely on some algebraic properties of cryptographic primitives. In [122], a list of some relevant algebraic properties of cryptographic operators is given, and for each of them, some examples of protocols or attacks using these properties are provided. The survey also gives an overview of the existing methods in formal approaches for analysing cryptographic protocols.

2.5 Exercises

Exercice 1 ($\star$)
Consider the following protocol:

$$\begin{array}{ll} A \to B: & \langle A, \{K\}_{\mathsf{pk}(B)}\rangle \\ B \to A: & \langle B, \{K\}_{\mathsf{pk}(A)}\rangle \end{array}$$

First, A generates a fresh key K and sends it encrypted with the public key of B. Only B will be able to decrypt this message. In this way, B learns K and B also knows that this message comes from A as indicated in the first part of the message he received. Hence, B answers to A by sending again the key, this time encrypted with the public key of A.

Show that an attacker can learn the key K generated by an honest agent A to another honest agent B.

Exercice 2 ($\star$)
The previous protocol is corrected as in the Needham-Schroeder protocol, *i.e.* we add the identity of the agent inside each encryption.

$$\begin{array}{ll} A \to B: & \{\langle A, K\rangle\}_{\mathsf{pk}(B)} \\ B \to A: & \{\langle B, K\rangle\}_{\mathsf{pk}(A)} \end{array}$$

1. Check that the previous attack does not exist anymore. Do you think that the secrecy property stated in Exercise 1 holds?

2. Two agents want to use this protocol to establish a session key. Show that there is an attack.

Exercice 3 ($\star\star$)
For double security, all messages in the previous protocol are encrypted twice:

$$\begin{array}{rl} A \rightarrow B: & \{\langle A, \{K\}_{\mathsf{pk}(B)}\rangle\}_{\mathsf{pk}(B)} \\ B \rightarrow A: & \{\langle B, \{K\}_{\mathsf{pk}(A)}\rangle\}_{\mathsf{pk}(A)} \end{array}$$

Show that the protocol then becomes insecure in the sense that an attacker can learn the key K generated by an honest agent A to another honest agent B.

Exercice 4 ($\star\star\star$)
We consider a variant of the Needham-Schroeder-Lowe protocol. This protocol is as follows:

$$\begin{array}{l} 1.\ A \rightarrow B:\ \{\langle A, N_A\rangle\}_{\mathsf{pk}(B)} \\ 2.\ B \rightarrow A:\ \{\langle N_A, \langle N_B, B\rangle\rangle\}_{\mathsf{pk}(A)} \\ 3.\ A \rightarrow B:\ \{N_B\}_{\mathsf{pk}(B)} \end{array}$$

1. Check that the 'man-in-the-middle' attack described in Figure 2.2 does not exist.

2. Show that there is an attack on the secrecy of the nonce N_b.
   hint: type confusion

3. Do you think that this attack is realistic? Why?

Chapter 3

A Small Process Calculus

We now define our cryptographic process calculus for describing protocols. This calculus is inspired by the applied pi calculus [7] which is the calculus used by the ProVerif tool [72]. The applied pi calculus is a language for describing concurrent processes and their interactions. It is an extension of the pi calculus [207] with cryptographic primitives. It is designed for describing and analysing a variety of security protocols, such as authentication protocols (*e.g.* [149]), key establishment protocols (*e.g.* [5]), e-voting protocols (*e.g.* [138]), ... These protocols try to achieve various security goals, such as secrecy, authentication, privacy, ...

In this chapter, we present a simplified version that is sufficient for our purpose and we explain how to formalise security properties in such a calculus.

3.1 Preliminaries

The applied pi calculus is similar to the spi calculus [10]. The key difference between the two formalisms concerns the way that cryptographic primitives are handled. The spi calculus has a fixed set of primitives built-in (symmetric and public key encryption), while the applied pi calculus allows one to define less usual primitives by means of an equational theory. This flexibility is particularly useful to model the new protocols that are emerging and which rely on new cryptographic primitives.

3.1.1 Messages

To describe processes, one starts with an infinite set of *names* $\mathcal{N}$ (which are used to represent atomic data, such as keys, nonces, ...), an infinite set of *variables* $\mathcal{X}$, and a *signature* $\mathcal{F}$ which consists of the *function symbols* which will be used to define *terms*. Each function symbol has an associated integer, its *arity*. In the case of security protocols, typical function symbols will include a binary function symbol senc for symmetric encryption, which takes plaintext and a key and returns the corresponding ciphertext, and a binary function symbol sdec for decryption, taking ciphertext and a key and returning the plaintext. Variables are used to consider messages containing unknown (unspecified) pieces.

Terms are defined as names, variables, and function symbols applied to other terms. Terms and function symbols may be sorted, and in such a case, function symbol application must respect sorts and arities. We denote by $\mathcal{T}(\Sigma)$ the set of terms built on the symbols in Σ. We denote by $fv(M)$ (resp. $fn(M)$) the set of variables (resp. names) that occur in M. A term M that does not contain any variable is a *ground term.* The set of positions of a term T is written $pos(T) \subseteq \mathbb{N}^*$, and its set of subterms $st(T)$. The subterm of T at position $p \in pos(T)$ is written $T|_p$. The term obtained by replacing $T|_p$ with a term U in T is denoted $T[U]_p$.

We split the function symbols between *private* and *public* symbols, *i.e.* $\mathcal{F} = \mathcal{F}_{\mathsf{pub}} \uplus \mathcal{F}_{\mathsf{priv}}$. Private function symbols are used to model algorithms or data that are not available to the attacker. Moreover, sometimes, we also split the function symbols into *constructors* and *destructors*, *i.e.* $\mathcal{F} = \mathcal{D} \uplus \mathcal{C}$. Destructors are used to model the fact that some operations fail. A typical destructor symbol could be the symbol sdec if we want to model a decryption algorithm that fails when we try to decrypt a ciphertext with a wrong key. A *constructor term* is a term in $\mathcal{T}(\mathcal{C} \cup \mathcal{N} \cup \mathcal{X})$.

By the means of a convergent term rewriting system $\mathcal{R}$, we describe the equations which hold on terms built from the signature. A *term rewriting system* (TRS) is a set of *rewrite rules* $l \to r$ where $l \in \mathcal{T}(\mathcal{F} \cup \mathcal{X})$ and $r \in \mathcal{T}(\mathcal{F} \cup fv(l))$. A term $S \in \mathcal{T}(\mathcal{F} \cup \mathcal{N} \cup \mathcal{X})$ rewrites to T by $\mathcal{R}$, denoted $S \to_{\mathcal{R}} T$, if there is $l \to r$ in $\mathcal{R}$, $p \in pos(S)$ and a substitution σ such that $S|_p = l\sigma$ and $T = S[r\sigma]_p$. Moreover, we assume that $\{x\sigma \mid x \in \mathsf{Dom}(\sigma)\}$ are constructor terms. We denote by $\to^*_{\mathcal{R}}$ the reflexive and transitive closure of $\to_{\mathcal{R}}$, and by $=_{\mathcal{R}}$ the symmetric, reflexive and transitive closure of $\to_{\mathcal{R}}$. A TRS $\mathcal{R}$ is *convergent* if it is:

- *terminating*, *i.e.* there is no infinite chain $T_1 \to_{\mathcal{R}} T_2 \to_{\mathcal{R}} \ldots$; and
- *confluent*, *i.e.* for all terms S, T such that $S =_{\mathcal{R}} T$, there exists U such that $S \to^*_{\mathcal{R}} U$ and $T \to^*_{\mathcal{R}} U$.

A term T is *$\mathcal{R}$-reduced* if there is no term S such that $T \to_{\mathcal{R}} S$. If $T \to^*_{\mathcal{R}} S$ and S is $\mathcal{R}$-reduced then S is *a $\mathcal{R}$-reduced form of* T. When this reduced form is unique (in particular if $\mathcal{R}$ is convergent), we write $S = T{\downarrow_{\mathcal{R}}}$ (or simply $T{\downarrow}$ when $\mathcal{R}$ is clear from the context). In the following, we will only consider convergent rewriting system. Hence, we have that $M =_{\mathcal{R}} N$, if and only if, $M{\downarrow} = N{\downarrow}$. A ground constructor term in normal form is also called a *message*.

Example 3.1 *In order to model the handshake protocol that we will present later on, we introduce the signature:*

$$\mathcal{F}_{\mathsf{senc}} = \{\mathsf{senc}/2,\ \mathsf{sdec}/2,\ \mathsf{f}/1\}$$

together with the term rewriting system $\mathcal{R}_{\mathsf{senc}} = \{\mathsf{sdec}(\mathsf{senc}(x, y), y) \to x\}$. *We will assume that* $\mathcal{F}_{\mathsf{senc}}$ *only contains constructor symbols. This represents a decryption algorithm that always succeeds. If we decrypt the ciphertext* $\mathsf{senc}(n, k)$ *with a key* $k' \neq k$*, the decryption algorithm will return the message* $\mathsf{sdec}(\mathsf{senc}(n, k), k')$.

Here, we have that $\mathsf{sdec}(\mathsf{senc}(n', \mathsf{sdec}(n, n)), \mathsf{sdec}(n, n)) =_{\mathcal{R}} n'$. *Indeed, we have that* $\mathsf{sdec}(\mathsf{senc}(n', \mathsf{sdec}(n, n)),$ *rewrites in one step to* n' *(with* $p = \epsilon$*, and* $\sigma = \{x \mapsto n',\ y \mapsto \mathsf{sdec}(n, n)\}$*).*

Example 3.2 *In order to model the Needham-Schroeder protocol, we will consider the following signature:*

$$\mathcal{F}_{\mathsf{aenc}} = \{\langle \_, \_\rangle,\ \mathsf{proj}_1/1,\ \mathsf{proj}_2/1,\ \mathsf{aenc}/2,\ \mathsf{pk}/1,\ \mathsf{sk}/1,\ \mathsf{adec}/2\}$$

together with the term rewriting system $\mathcal{R}_{\mathsf{aenc}}$*:*

$$\mathsf{proj}_1(\langle x, y\rangle) \ \to\ x \qquad \mathsf{proj}_2(\langle x, y\rangle) \ \to\ y \qquad \mathsf{adec}(\mathsf{aenc}(x, \mathsf{pk}(y)), \mathsf{sk}(y)) \ \to\ x$$

This will allow us to model asymmetric encryption and pairing. We will assume that proj_1*,* proj_2*, and* adec *are destructors symbols. The only private non-constant symbol is the symbol* sk*. Note that* $\mathsf{proj}_1(\langle n, \mathsf{adec}(n, n)\rangle) \neq_{\mathcal{R}} n$*. Indeed, the terms* $\mathsf{proj}_1(\langle n, \mathsf{adec}(n, n)\rangle$ *and* n *are both irreducible and not syntactically equal.*

3.1.2 Assembling Terms into Frames

At some moment, while engaging in one or more sessions of one or more protocols, an attacker may have observed a sequence of messages $M_1, \ldots, M_\ell$, *i.e.* a set of ground constructor terms in normal form. We want to represent this knowledge of the attacker. It is not enough for us to say that the attacker knows the *set* of terms $\{M_1, \ldots, M_\ell\}$ since he also knows the order that he observed them in. Furthermore, we should distinguish those names that the attacker knows from those that were freshly generated by others and which remain secret from the attacker; both kinds of names may appear in the terms. We use the concept of *frame* from the applied pi calculus [7] to represent the knowledge of the attacker. A *frame* $\phi = \mathsf{new}\ \overline{n}.\sigma$ consists of a finite set $\overline{n} \subseteq \mathcal{N}$ of *restricted* names (those that the attacker does not know), and a substitution σ of the form:

$$\{^{M_1}/_{x_1}, \ldots, {}^{M_\ell}/_{x_\ell}\}.$$

The variables enable us to refer to each message M_i. We always assume that the terms M_i are ground term in normal form that do not contain destructor symbols. The names $\overline{n}$ are bound and can be renamed. We denote by $=_\alpha$ the α-renaming relation on frames. The *domain* of the frame ϕ, written $\mathsf{Dom}(\phi)$, is defined as $\{x_1, \ldots, x_\ell\}$.

3.1.3 Deduction

Given a frame ϕ that represents the information available to an attacker, we may ask whether a given ground constructor term M may be deduced from ϕ. Given a convergent rewriting system $\mathcal{R}$ on $\mathcal{F}$, this relation is written $\phi \vdash_\mathcal{R} M$ and is formally defined below.

Definition 3.1 (Deduction) *Let M be a ground term and $\phi = \mathsf{new}\ \overline{n}.\sigma$ be a frame. We have that $\mathsf{new}\ \overline{n}.\sigma \vdash_\mathcal{R} M$ if, and only if, there exists a term $N \in \mathcal{T}(\mathcal{F}_{\mathsf{pub}} \cup \mathcal{N} \cup \mathsf{Dom}(\phi))$ such that $fn(N) \cap \overline{n} = \emptyset$ and $N\sigma =_\mathcal{R} M$. Such a term N is a* recipe *of the term M.*

Intuitively, the deducible messages are the messages of ϕ and the names that are not protected in ϕ, closed by rewriting with $\mathcal{R}$ and closed by application of public function symbols. When $\mathsf{new}\ \overline{n}.\sigma \vdash_\mathcal{R} M$, any occurrence of names from $\overline{n}$ in M is bound by $\mathsf{new}\ \overline{n}$. So $\mathsf{new}\ \overline{n}.\sigma \vdash_\mathcal{R} M$ could be formally written $\mathsf{new}\ \overline{n}.(\sigma \vdash_\mathcal{R} M)$.

Example 3.3 *Consider the theory $\mathcal{R}_{\mathsf{senc}}$ given in Example 3.1 and the following frame:*

$$\phi = \mathsf{new}\ k, s_1.\{^{\mathsf{senc}(\langle s_1, s_2\rangle, k)}/_{x_1},\ {}^{k}/_{x_2}\}.$$

We have that $\phi \vdash_{\mathcal{R}_{\mathsf{senc}}} k$, $\phi \vdash_{\mathcal{R}_{\mathsf{senc}}} s_1$ and $\phi \vdash_{\mathcal{R}_{\mathsf{senc}}} s_2$. Indeed x_2, $\mathsf{proj}_1(\mathsf{sdec}(x_1, x_2))$ and s_2 are recipes of the terms k, s_1 and s_2 respectively.

The relation $\mathsf{new}\ \overline{n}.\sigma \vdash_\mathcal{R} M$ can be axiomatized by the following rules:

$$\frac{}{\mathsf{new}\ \overline{n}.\sigma \vdash_\mathcal{R} M}\ \text{if } \exists x \in dom(\sigma) \text{ such that } x\sigma = M \qquad \frac{}{\mathsf{new}\ \overline{n}.\sigma \vdash_\mathcal{R} s}\ s \in \mathcal{N} \smallsetminus \overline{n}$$

$$\frac{\phi \vdash_\mathcal{R} M_1 \ \ \ldots \ \ \phi \vdash_\mathcal{R} M_\ell}{\phi \vdash_\mathcal{R} \mathsf{f}(M_1, \ldots, M_\ell)}\ \mathsf{f} \in \mathcal{F}_{\mathsf{pub}} \qquad \frac{\phi \vdash_\mathcal{R} M}{\phi \vdash_\mathcal{R} M'}\ M =_\mathcal{R} M'$$

Since we only consider convergent rewriting systems, it is easy to prove that the two definitions coincide.

3.1.4 Static Equivalence

The frames we have introduced are too fine-grained as representations of the attacker's knowledge. For example, $\nu k.\{^{\mathsf{senc}(s_0,k)}/_x\}$ and $\nu k.\{^{\mathsf{senc}(s_1,k)}/_x\}$ represent a situation in which the encryption of the public name s_0 (resp. s_1) by a randomly-chosen key has been observed. Since the attacker cannot detect the difference between these two situations, the frames should be considered equivalent. To formalise this, we note that if two recipes M, N on the frame ϕ produce the same constructor term, we say they are equal in the frame, and write $(M =_\mathcal{R} N)\phi$. Thus, the knowledge of the attacker can be thought of as his ability to distinguish such recipes. If two frames have identical distinguishing power, then we say that they are *statically equivalent.*

Definition 3.2 (static equivalence) *We say that two terms M and N in $\mathcal{T}(\mathcal{F}_{\mathsf{pub}} \cup \mathcal{N} \cup \mathcal{X})$ are* equal in the frame ϕ, *and write $(M =_{\mathcal{R}} N)\phi$, if there exists $\overline{n}$ and a substitution σ such that $\phi =_\alpha \nu\overline{n}.\sigma$, $\overline{n} \cap (fn(M) \cup fn(N)) = \emptyset$, and $M\sigma\!\downarrow$ and $N\sigma\!\downarrow$ are both constructor terms that are equal,* i.e. *$M\sigma\!\downarrow = N\sigma\!\downarrow$.*

We say that two frames $\phi_1 = \overline{n_1}.\sigma_1$ and $\phi_2 = \overline{n_2}.\sigma_2$ are statically equivalent, *and write $\phi_1 \sim_{\mathcal{R}} \phi_2$, when:*

- $\mathsf{Dom}(\phi_1) = \mathsf{Dom}(\phi_2)$,
- *for all term $M \in \mathcal{T}(\mathcal{F}_{\mathsf{pub}} \cup \mathcal{N} \cup \mathcal{X})$ such that $fn(M) \cap (\overline{n_1} \cup \overline{n_2}) = \emptyset$, we have that: $M\sigma_1\!\downarrow$ is constructor term $\Leftrightarrow$ $M\sigma_2\!\downarrow$ is a constructor term.*
- *for all terms M, N in $\mathcal{T}(\mathcal{F}_{\mathsf{pub}} \cup \mathcal{N} \cup \mathcal{X})$ we have that: $(M =_{\mathcal{R}} N)\phi_1 \Leftrightarrow (M =_{\mathcal{R}} N)\phi_2$.*

Note that by definition of $\sim$, we have that $\phi_1 \sim \phi_2$ when $\phi_1 =_\alpha \phi_2$ and we have also that $\mathsf{new}\ n.\phi \sim \phi$ when n does not occur in ϕ.

Example 3.4 *Consider the rewriting system $\mathcal{R}_{\mathsf{senc}}$ provided in Example 3.1. Consider the frames $\phi = \mathsf{new}\ k.\{^{\mathsf{senc}(s_0,k)}/_{x_1}, {}^{k}/_{x_2}\}$, and $\phi' = \mathsf{new}\ k.\{^{\mathsf{senc}(s_1,k)}/_{x_1}, {}^{k}/_{x_2}\}$. Intuitively, s_0 and s_1 could be the two possible (public) values of a vote. We have $(\mathsf{sdec}(x_1, x_2) =_{\mathcal{R}_{\mathsf{senc}}} s_0)\phi$ whereas $(\mathsf{sdec}(x_1, x_2) \neq_{\mathcal{R}_{\mathsf{senc}}} s_0)\phi'$. Therefore we have that $\phi \not\sim \phi'$. However, we have that:*

$$\mathsf{new}\ k.\{^{\mathsf{senc}(s_0,k)}/_{x_1}\} \sim \mathsf{new}\ k.\{^{\mathsf{senc}(s_1,k)}/_{x_1}\}.$$

Example 3.5 *Consider again the rewriting system $\mathcal{R}_{\mathsf{senc}}$ provided in Example 3.1. We have that:*

$$\begin{array}{rcll}
\mathsf{new}\ k.\{^{\mathsf{senc}(0,k)}/_x\} & \sim & \mathsf{new}\ k.\{^{\mathsf{senc}(1,k)}/_x\} & \\
\{^{\mathsf{senc}(0,k)}/_x, {}^{\langle 0,k\rangle}/_y\} & \not\sim & \mathsf{new}\ k.\{^{\mathsf{senc}(1,k)}/_x, {}^{\langle 0,k\rangle}/_y\} & (\mathsf{sdec}(x, \mathsf{proj}_2(y)) \stackrel{?}{=} 0) \\
\mathsf{new}\ a.\{^{a}/_x\} & \sim & \mathsf{new}\ b.\{^{b}/_x\} & \\
\mathsf{new}\ a.\{^{a}/_x\} & \not\sim & \mathsf{new}\ b.\{^{b}/_y\} & (\textit{different domains}) \\
\{^{a}/_x\} & \not\sim & \{^{b}/_x\} & (x \stackrel{?}{=} a)
\end{array}$$

3.2 Protocols

We now described our cryptographic process calculus for describing protocols. For sake of simplicity, we only consider public channels, *i.e.* under the control of the attacker.

3.2.1 Protocol Language

The grammar for *processes* is given below. One has *plain processes* P, Q, R and *extended processes* A, B, C.

Plain processes. Plain processes are formed from the following grammar

$$\begin{array}{lll}
P, Q, R \ \hat{=} & \text{plain processes} & \\
& 0 & \text{null process} \\
& P \parallel Q & \text{parallel composition} \\
& \mathsf{in}(c, M_i).P & \text{message input} \\
& \mathsf{out}(c, M_o).P & \text{message output} \\
& \mathsf{if}\ M = N\ \mathsf{then}\ P\ \mathsf{else}\ Q & \text{conditional} \\
& \mathsf{new}\ n.P & \text{restriction} \\
& !P & \text{replication}
\end{array}$$

such that a variable x appears in a term only if the term is in the scope of an input $\mathsf{in}(c, M_i)$ with $x \in fv(M_i)$. The null process 0 does nothing; $P \parallel Q$ is the parallel composition of P and Q. The replication $!P$ behaves as an infinite number of copies of P running in parallel. The conditional construction $\mathsf{if}\ M = N\ \mathsf{then}\ P\ \mathsf{else}\ Q$ is standard. We omit $\mathsf{else}\ Q$ when Q is 0. The process $\mathsf{in}(c, M_i).P$ is ready to input on the public channel c, then to run P where the variables of M_i are bound by the actual input message. The term M_i is a constructor term with variables. $\mathsf{out}(c, M_o).P$ is ready to output M_o (it may contains some destructors), then to run P. Again, we omit P when P is 0.

In this definition, we consider both pattern inputs and conditionals, which is redundant in some situations: for any executable process, the patterns can be replaced with conditionals. However, we keep both possibilities, in order to keep some flexibility in writing down the protocols.

Example 3.6 *We illustrate our syntax with the well-known handshake protocol that can be informally described as follows:*

$$\begin{array}{lcl} A & \rightarrow & B: \ \mathsf{senc}(n, w) \\ B & \rightarrow & A: \ \mathsf{senc}(\mathsf{f}(n), w) \end{array}$$

We rely on the signature given in Example 3.1. The goal of this protocol is to authenticate B from A's point of view, provided that they share an initial secret w. This is done by a simple challenge-response transaction: A sends a random number (a nonce*) encrypted with the shared secret key w. Then, B decrypts this message, applies a given function (for instance $\mathsf{f}(n) = n+1$) to it, and sends the result back, also encrypted with w. Finally, the agent A checks the validity of the result by decrypting the message and checking the decryption against $\mathsf{f}(n)$. In our calculus, we can model the protocol as $\mathsf{new}\ w.(P_A \parallel P_B)$ where*

- $P_A(w) = \mathsf{new}\ n.\ \mathsf{out}(c, \mathsf{senc}(n, w)).\ \mathsf{in}(c, x).\ \mathsf{if}\ \mathsf{sdec}(x, w) = \mathsf{f}(n)\ \mathsf{then}\ P$
- $P_B(w) = \mathsf{in}(c, y).\ \mathsf{out}(c, \mathsf{senc}(\mathsf{f}(\mathsf{sdec}(y, w)), w)).$

where P models an application that is executed when P_B has been successfully authenticated. Here, we use the formalism with explicit destructors but we could also used pattern inputs.

Example 3.7 *Going back to the Needham-Schroeder public key protocol described in Chapter 2 and considering the signature given in Example 3.2, we have that:*

$$\begin{array}{lcl} P_A(a,b) & \hat{=} & \mathsf{out}(c, \mathsf{aenc}(\langle a, N_a\rangle, \mathsf{pk}(b))). \\ & & \mathsf{in}(c, \mathsf{aenc}(\langle N_a, x\rangle, \mathsf{pk}(a)). \\ & & \mathsf{out}(c, \mathsf{aenc}(x, \mathsf{pk}(b))) \end{array} \qquad \begin{array}{lcl} P_B(a,b) & \hat{=} & \mathsf{in}(c, \mathsf{aenc}(\langle a, y\rangle, \mathsf{pk}(b))). \\ & & \mathsf{out}(c, \mathsf{aenc}(\langle y, N_b\rangle, \mathsf{pk}(a))). \\ & & \mathsf{in}(c, \mathsf{aenc}(N_b, \mathsf{pk}(b))) \end{array}$$

Here, we have used pattern inputs. We could also have used the alternative formalism of explicit destructors. With pattern inputs, we do not need in general to used destructors to describe the outputs.

Note that all the processes that can be written in this syntax (in particular the one with pattern inputs) are not necessary meaningful. Some of them will not be executable.

Continuing with the Needham-Schroeder protocol, we may define several execution scenarii:

Example 3.8 (Scenario 1) *The following specifies a copy of the role of Alice, played by a, with d and a copy of the role of Bob, played by b, with a, as well as the fact that d is dishonest, hence his secret key is leaked.*

$$P_1 \hat{=} (\mathsf{new}\ N_a.\ P_A(a, d)) \parallel (\mathsf{new}\ N_b.\ P_B(a, b)) \parallel \mathsf{out}(c, \mathsf{sk}(d))$$

Example 3.9 (Scenario 2) *Assume that we wish a to execute the role of the initiator, however with any other party, which is specified here by letting the environment give the identity of such another party: the process first receives x_b, that might be bound to any value. The other role is specified in the same way.*

$$P_2 \mathrel{\hat{=}} (\mathsf{new}\ N_a.\ \mathsf{in}(c, x_b).\ P_A(a, x_b)) \parallel (\mathsf{new}\ N_b.\ \mathsf{in}(c, x_a).\ P_B(x_a, b)) \parallel \mathsf{out}(c, \mathsf{sk}(d))$$

Example 3.10 (Scenario 3) *In Example 3.8 and Example 3.9, a was only able to engage the protocol once (and b was only able to engage once in a response). We may wish a (resp. b) be able to execute any number of instances of the role of the initiator (resp. responder).*

$$P_3 \mathrel{\hat{=}} !(\mathsf{new}\ N_a.\ \mathsf{in}(c, x_b).\ P_A(a, x_b))\ \parallel !(\mathsf{new} N_b.\ \mathsf{in}(c, x_a).\ P_B(x_a, b))\ \parallel\ \mathsf{out}(c, \mathsf{sk}(d))$$

Example 3.11 (Scenario 4) *Finally, in general, the role of the initiator could be executed by any agent, including b and the role of the responder could be executed by any number of agents as well. We specify an unbounded number of parties, engaging in an unbounded number of sessions by:*

$$P_4 \mathrel{\hat{=}} \begin{cases} !(\mathsf{new}\ N_a.\ \mathsf{in}(c, x_a).\ \mathsf{in}(c, x_b).\ P_A(x_a, x_b))\ \parallel \\ !(\mathsf{new}\ N_b.\ \mathsf{in}(c, x_a).\ \mathsf{in}(c, x_b).\ P_B(x_a, x_b))\ \parallel\ \mathsf{out}(c, \mathsf{sk}(d)) \end{cases}$$

We can imagine other scenarios as well. Verifying security will only be relative to a given scenario.

Extended Processes. Further, we extend processes with active substitutions and restrictions:

$$A, B, C := P\ \big|\ A \parallel B\ \big|\ \mathsf{new}\ n.A\ \big|\ \{^M/_x\}$$

where M is a ground constructor term in normal form. As usual, names and variables have scopes, which are delimited by restrictions and by inputs. We write $fv(A)$, $bv(A)$, $fn(A)$, $bn(A)$ for the sets of free and bound variables (resp. names). Moreover, we require processes to be *name and variable distinct*, meaning that $bn(A) \cap fn(A) = \emptyset$, $bv(A) \cap fv(A) = \emptyset$, and also that any name and variable is bound at most once in A. Note that the only free variables are introduced by active substitutions (the x in $\{^M/_x\}$). Lastly, in an extended process, we require that there is at most one substitution for each variable. An *evaluation context* is an extended process with a hole instead of an extended process.

Extended processes built up from the null process, active substitutions using parallel composition and restriction are called *frames* (extending the notion of frame introduced in Section 3.1.2). Given an extended process A we denote by $\phi(A)$ the frame obtained by replacing any embedded plain processes in it with 0.

Example 3.12 *Consider the following process:*

$$A = \mathsf{new}\ s, k_1.(\mathsf{out}(c, a) \parallel \{^{\mathsf{senc}(s,k_1)}/_x\} \parallel \mathsf{new}\ k_2.\mathsf{out}(c, \mathsf{senc}(s, k_2))).$$

We have that $\phi(A) = \mathsf{new}\ s, k_1.(0 \parallel \{^{\mathsf{senc}(s,k_1)}/_x\} \parallel \mathsf{new}\ k_2.0)$.

3.2.2 Operational Semantics

To formally define the operational semantics of our calculus, we have to introduce three relations, namely *structural equivalence*, *internal reduction*, and *labelled transition*.

Structural Equivalence. Informally, two processes are *structurally equivalent* if they model the same thing, even if the grammar permits different encodings. For example, to describe a pair of processes P_A and P_B running in parallel, we have to write either $P_A \parallel P_B$, or $P_B \parallel P_A$. These two processes are said to be structurally equivalent. More formally, structural equivalence is the smallest equivalence relation closed by application of evaluation contexts and such that:

$$\begin{array}{lrcl}
\textsc{Par-0} & A \parallel 0 & \equiv & A \\
\textsc{Par-C} & A \parallel B & \equiv & B \parallel A \\
\textsc{Par-A} & (A \parallel B) \parallel C & \equiv & A \parallel (B \parallel C) \\
\\
\textsc{New-Par} & A \parallel \mathsf{new}\ n.B & \equiv & \mathsf{new}\ n.(A \parallel B) \quad n \notin fn(A) \\
\textsc{New-C} & \mathsf{new}\ n_1.\mathsf{new}\ n_2.A & \equiv & \mathsf{new}\ n_2.\mathsf{new}\ n_1.A
\end{array}$$

Note that the side condition of the rule New-Par is always true on processes that are name and variable distinct. Using structural equivalence, every extended process A can be rewritten to consist of a substitution and a plain process with some restricted names, *i.e.*

$$A \equiv \mathsf{new}\ \overline{n}.(\{^{M_1}/_{x_1}\} \parallel \ldots \parallel \{^{M_k}/_{x_k}\} \parallel P).$$

In particular, any frame can be rewritten as $\mathsf{new}\ \overline{n}.\sigma$ matching the notion of frame introduced in Section 3.1.2. We note that unlike in the original applied pi calculus, active substitutions cannot "interact" with the extended processes. As we will see in the following, active substitutions record the outputs of a process to the environment. The notion of frames will be particularly useful to define equivalence based security properties such as resistance against guessing attacks and privacy type properties.

Internal Reduction. A process can be executed without contact with its environment, *e.g.* execution of conditionals, or internal communications between processes in parallel. Formally, *internal reduction* is the smallest relation on processes closed under structural equivalence and application of evaluation contexts such that:

$$\begin{array}{ll}
\textsc{Repl} & !P \xrightarrow{\tau} P' \parallel !P \quad \text{where } P' \text{ is a fresh renaming of } P \\
\\
\textsc{Then} & \mathsf{if}\ M = N\ \mathsf{then}\ P\ \mathsf{else}\ Q \xrightarrow{\tau} P \text{ where } M{\downarrow} = N{\downarrow} \text{ and } M{\downarrow} \text{ is a message} \\
\\
\textsc{Else} & \mathsf{if}\ M = N\ \mathsf{then}\ P\ \mathsf{else}\ Q \xrightarrow{\tau} Q \text{ where } M{\downarrow} \neq N{\downarrow} \text{ and } M{\downarrow}, N{\downarrow} \text{ are messages} \\
\\
\textsc{Comm} & \mathsf{out}(c, M_1).P_1 \parallel \mathsf{in}(c, M_2).P_2 \xrightarrow{\tau} P_1 \parallel P_2\theta \text{ where } \theta \text{ is such that} \\
& \qquad \mathsf{Dom}(\theta) = fv(M_2),\ M_2\theta{\downarrow} = M_1{\downarrow}, \text{ and } M_1{\downarrow} \text{ is a message.}
\end{array}$$

We write $\rightarrow^*$ for the reflexive and transitive closure of $\xrightarrow{\tau}$. Note that, in some situations, a process of the form $\mathsf{if}\ M = N\ \mathsf{then}\ P\ \mathsf{else}\ Q$ may block. This happens when $M{\downarrow}$ (resp. $N{\downarrow}$) contains some destructors.

Labelled Transition. Communications are synchronous, but (as long as there is no private channel) we can assume that they occur with the environment. We sketch here a labelled transition semantics. The semantics given previously allow us to reason about protocols with an adversary represented by a context. In order to prove that security properties hold for all adversaries, quantification over all contexts is typically required, which can be difficult in practise. The *labelled semantics* aim to eliminate universal quantification of the context. We have two main rules:

$$\begin{array}{lll}
\textsc{In} & \mathsf{in}(c, x).P \xrightarrow{\mathsf{in}(c,M)}_{\ell} P\{^M/_x\} & \text{where } M \text{ is a message} \\
\\
\textsc{Out} & \mathsf{out}(c, M).P \xrightarrow{\mathsf{out}(c,M\downarrow)}_{\ell} P \parallel \{^{M\downarrow}/_x\} & \text{where } x \text{ is a fresh variable and } M{\downarrow} \text{ is a message}
\end{array}$$

The labelled operational semantics is closed by structural equivalence and under some evaluation contexts. Actually, we have that:

$$\frac{A \equiv A' \quad A' \xrightarrow{\alpha}_\ell B' \quad B' \equiv B}{A \xrightarrow{\alpha}_\ell B} \qquad\qquad \frac{A \xrightarrow{\alpha}_\ell B}{C[A] \xrightarrow{\alpha}_\ell C[B]}$$

where C is an evaluation context, and in case of an input, *i.e.* $\alpha = \mathsf{in}(c, M)$, we have that $\phi(C[A]) \vdash_{\mathcal{R}} M$.

We write $\rightarrow_\ell$ to denote $\xrightarrow{\tau} \cup \xrightarrow{\alpha}_\ell$ and $\rightarrow_\ell^*$ to denote the reflexive and transitive closure of $\rightarrow_\ell$.

Example 3.13 *Going back to the handshake protocol described in Example 3.6, the derivation described below represents a normal execution of the protocol. For simplicity of this example we suppose that $x \notin fv(P)$.*

$$\begin{array}{rl}
& \mathsf{new}\ w.(P_A(w) \parallel P_B(w)) \\
\xrightarrow{\mathsf{out}(c,\mathsf{senc}(n,w))}_\ell & \mathsf{new}\ w, n.(P_B(w) \parallel \{^{\mathsf{senc}(n,w)}/_{x_1}\} \parallel \mathsf{in}(c,x).\ \mathsf{if}\ \mathsf{sdec}(x,w) = \mathsf{f}(n)\ \mathsf{then}\ P) \\
\xrightarrow{\mathsf{in}(c,\mathsf{senc}(n,w))}_\ell & \mathsf{new}\ w, n.(\mathsf{out}(c,M) \parallel \{^{\mathsf{senc}(n,w)}/_{x_1}\} \parallel \mathsf{in}(c,x).\ \mathsf{if}\ \mathsf{sdec}(x,w) = \mathsf{f}(n)\ \mathsf{then}\ P) \\
\xrightarrow{\mathsf{out}(c,M\downarrow)}_\ell & \mathsf{new}\ w, n.(\{^{\mathsf{senc}(n,w)}/_{x_1}\} \parallel \{^{M\downarrow}/_{x_2}\} \parallel \mathsf{in}(c,x).\ \mathsf{if}\ \mathsf{sdec}(x,w) = \mathsf{f}(n)\ \mathsf{then}\ P) \\
\xrightarrow{\mathsf{in}(c,\mathsf{senc}(\mathsf{f}(n),w))}_\ell & \mathsf{new}\ w, n.(\{^{\mathsf{senc}(n,w)}/_{x_1}\} \parallel \{^{M\downarrow}/_{x_2}\} \parallel \mathsf{if}\ \mathsf{sdec}(\mathsf{senc}(\mathsf{f}(n),w),w) = \mathsf{f}(n)\ \mathsf{then}\ P) \\
\xrightarrow{\tau} & \mathsf{new}\ w, n.(\{^{\mathsf{senc}(n,w)}/_{x_1}\} \parallel \{^{M\downarrow}/_{x_2}\} \parallel P)
\end{array}$$

where $M = \mathsf{senc}(\mathsf{f}(\mathsf{sdec}(\mathsf{senc}(n,w),w)),w) \rightarrow_{\mathcal{R}_{\mathsf{senc}}} \mathsf{senc}(\mathsf{f}(n),w)$.

Example 3.14 *Continuing Example 3.7 we develop some transitions from*

$$P_1 = (\mathsf{new}\ N_a.\ P_A(a,d)) \ \parallel\ (\mathsf{new}\ N_b.\ P_B(a,b)) \ \parallel\ \mathsf{out}(c,\mathsf{sk}(d))$$

For convenience, the names N_a and N_b are pushed out. We obtain another process that is structurally equivalent.

Case 1: *The process P_A may move first, yielding*

$$\begin{array}{rl}
P_1 \xrightarrow{\mathsf{out}(c,\mathsf{aenc}(\langle a,N_a\rangle,\mathsf{pk}(d)))}_\ell \mathsf{new}\ N_a.\mathsf{new}\ N_b.\ (& \{^{\mathsf{aenc}(\langle a,N_a\rangle,\mathsf{pk}(d))}/_{x_1}\} \\
& \parallel (\mathsf{in}(c,\mathsf{aenc}(\langle N_a,x\rangle,\mathsf{pk}(a))).\ \mathsf{out}(c,\mathsf{aenc}(x,\mathsf{pk}(b))) \\
& \parallel P_B(a,b) \\
& \parallel \mathsf{out}(c,\mathsf{sk}(d))\ \ )
\end{array}$$

Case 2: *The process P_B may also move first, and the resulting process depends on an input M_1 such that* $\mathsf{new}\ N_a, N_b.(\sigma \vdash \mathsf{aenc}(\langle a, M_1\rangle, \mathsf{pk}(b)))$ *where* $\mathsf{Dom}(\sigma) = \emptyset$.

$$\begin{array}{rl}
P_1 \xrightarrow{\mathsf{in}(c,M_1)}_\ell = \mathsf{new}\ N_a, \mathsf{new}\ N_b.\ (& P_A(a,d) \\
\parallel & \mathsf{out}(c,\mathsf{aenc}(\langle M_1,N_b\rangle,\mathsf{pk}(a))).\mathsf{in}(c,\mathsf{aenc}(N_b,\mathsf{pk}(b))) \\
\parallel & \mathsf{out}(c,\mathsf{sk}(d))\ \ )
\end{array}$$

Case 3: *The last process may also move first, yielding*

$$P_1 \xrightarrow{\mathsf{out}(c,\mathsf{sk}(d))}_\ell \mathsf{new}\ N_a, \mathsf{new}\ N_b.\ (\ \ \{^{\mathsf{sk}(d)}/_{x_1}\} \ \parallel\ P_A(a,d) \ \parallel\ P_B(a,b) \ \ )$$

From the resulting processes, there are again several possible transitions. We do not continue here the full transition sequence, which is too large to be displayed.

In the above example, we see that the transition system might actually be infinite. Indeed, the term M_1 is an arbitrary message that satisfies some deducibility conditions. Such deducibility conditions can be simplified (and decided). This will be the subject of Chapter 4 on bounded process verification.