Formal Security Proofs

Hubert Comon-Lundh and Stéphanie Delaune

Abstract.
The goal of the lecture is to present some aspects of formal security proofs of protocols. This is a wide area, and there is another lecture (by B. Banchet) on related topics. The idea is therefore to explain in depth one particular technique, that relies on *deducibility constraints*. We rely mainly on two introductory documents [8,14]. Actually, the current notes are the beginning of [8].

Here is a roadmap:

1. We introduce the problem with examples and touch a little the question of the validity of the security models (section 1).
   We describe then a small process algebra, that will serve as a model for the protocols, as well as a few security properties (section 2).
2. The core of the lecture is here: we introduce the attacker model, as a deduction system, and show how to represent any execution in the hostile environment as *deducibility constraints*. In short, a deducibility constraint is a sequence of proofs, in which some parts are unknown (and formalized with variables) and possibly re-used in other constraints. An instance of such a constraints yields an attacker's strategy.
   We explain how to solve such constraints in a particular setting of a few cryptographic primitives. This is more or less what is described in the first part of [12] and is detailed in the section 3.

Though the lecture aims at being self-contained, it assumes some familiarity with inference rules/ formal proofs (or SOS for programming languages) and terms/ substitutions/ unification. Similarly, a knowledge on concurrency is not required, but will make easier the understanding of the model.

Keywords. Security, formal methods, models of concurrency, symbolic constraints, protocols

1. An introductory example

We start with the well-known example of the so-called "Needham-Schroeder public-key protocol" [23], that has been designed in 1978 and for which an attack was found in 1996 by G. Lowe [18], using formal methods.

1.1. 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 1.

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

Figure 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.

Attack. Actually, an attack was found in 1996 by G. Lowe [18] on the Needham-Schroeder public-key protocol. The attack described in Figure 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 3).

The attack described above cannot be mounted in the corrected version of the protocol. Actually, it is reported in [18] that the technique that permitted to find the Lowe attack on the Needham-Schroeder public key protocol found no attack on this protocol.

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

Figure 2. Attack on the Needham-Schroeder public key protocol

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

Figure 3. Description of the Needham-Schroeder-Lowe protocol

1.2. 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 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 generates a random number N_b
$\mathsf{out}(c, \{y_0, N_b\}_{\mathsf{pk}(a)}).$	b sends a reply 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 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:

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

The Needham-Schroeder-Lowe protocol has been proved secure in several formal models close to the one we have sketched in this section [9,6].

1.3. Further Readings

A survey by Clark and Jacob [10] 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 [15], 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.

1.4. Exercises

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

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

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.

$$A \to B : \{\langle A, K\rangle\}_{\mathsf{pk}(B)}$$
$$B \to A : \{\langle B, K\rangle\}_{\mathsf{pk}(A)}$$

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:

$$A \to B : \{\langle A, \{K\}_{\mathsf{pk}(B)}\rangle\}_{\mathsf{pk}(B)}$$
$$B \to A : \{\langle B, \{K\}_{\mathsf{pk}(A)}\rangle\}_{\mathsf{pk}(A)}$$

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:

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

1. Check that the 'man-in-the-middle' attack described in Figure 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?

1.5. An Attack on the Fixed Version of the Protocol

In this section, we show that we must be cautious with the use of formal methods, and in particular with the assumptions on the implementation of the security primitives. The issue, that is raised here is covered by the *soundness* results that show under which condition the formal model is fully abstract with respect to the computational one. Discussing soundness results would be the subject of another lecture...

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 $\{m\}_{\mathsf{pk}(a)} =$

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 4. Attack on the Needham-Schroeder-Lowe protocol with El-Gamal encryption.

$(\mathsf{pk}(a)^r \times g^{m'}, g^r)$. 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 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 [26] 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.

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. A small process calculus

We now define our cryptographic process calculus for describing protocols. This calculus is inspired by the applied pi calculus [3] which is the calculus used by the PROVERIF tool [9]. The applied pi calculus is a language for describing concurrent processes and their interactions. It is an extension of the pi calculus [22] with cryptographic primitives. It is designed for describing and analysing a variety of security protocols, such as authentication protocols (*e.g.* [17]), key establishment protocols (*e.g.* [5]), e-voting protocols (*e.g.* [16]), ... 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.

2.1. Preliminaries

The applied pi calculus is similar to the spi calculus [2]. 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.

2.1.1. 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. $\mathsf{fn}(M)$) 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 $\mathsf{St}(T)$. 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 2 *In order to model the handshake protocol that we will present later on, we introduce the signature:*

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

together with the term rewriting system $\mathcal{R}_{\mathsf{senc}} = \{\mathit{sdec}(\mathsf{senc}(x, y), y) \to x\}$. *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* $\mathit{sdec}(\mathsf{senc}(n, k), k')$.

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

Example 3 *In order to model the Needham-Schroeder protocol, we will consider the following signature:*

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

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

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

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.*

2.1.2. 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 [3] 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 $\{x_1, \ldots, x_\ell\}$.

2.1.3. 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 2.1 (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 $\mathsf{fn}(N) \cap \overline{n} = \emptyset$ 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 4 *Consider the theory $\mathcal{R}_{\mathsf{senc}}$ given in Example 2 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:

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

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

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

2.1.4. 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 2.2 (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 $\phi =_\alpha \nu\overline{n}.\sigma$, $\overline{n} \cap (\mathsf{fn}(M) \cup \mathsf{fn}(N)) = \emptyset$, 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 $\mathsf{fn}(M) \cap (\overline{n_1} \cup \overline{n_2}) = \emptyset$, 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 5 *Consider the rewriting system $\mathcal{R}_{\mathsf{senc}}$ provided in Example 2. 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 6 *Consider again the rewriting system $\mathcal{R}_{\mathsf{senc}}$ provided in Example 2. We have that:*

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

2.2. 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.

2.2.1. 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

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

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

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

We rely on the signature given in Example 2. 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*

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

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 8 *Coming back to the Needham-Schroeder public key protocol described in Section 1 and considering the signature given in Example 3, we have:*

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

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 9 (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.*

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

Example 10 (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.*

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

Example 11 (Scenario 3) *In Example 9 and Example 10, 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).*

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

Example 12 (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:*

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

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:

$$A, B, C := P \;\mid\; A \parallel B \;\mid\; \mathsf{new}\ n.A \;\mid\; \{^{M}/_{x}\}$$

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 $fv(A)$, $bv(A)$, $\mathsf{fn}(A)$, $\mathsf{bn}(A)$ for the sets of free and bound variables (resp. names). Moreover, we require processes to be *name and variable distinct*, meaning that $\mathsf{bn}(A) \cap \mathsf{fn}(A) = \emptyset$, $bv(A) \cap fv(A) = \emptyset$, 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 2.1.2). Given an extended process A we denote by $\phi(A)$ the frame obtained by replacing any embedded plain processes in it with 0.

Example 13 *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 $\phi(A) = \mathsf{new}\ s, k_1.(0 \parallel \{^{\mathsf{senc}(s,k_1)}/_{x}\} \parallel \mathsf{new}\ k_2.0)$.

2.2.2. 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:

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

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 2.1.2. 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:

REPL $!P \xrightarrow{\tau} P' \parallel !P$ where P' is a fresh renaming of P

THEN $\mathsf{if}\ M = N\ \mathsf{then}\ P\ \mathsf{else}\ Q \xrightarrow{\tau} P$ where $M{\downarrow} = N{\downarrow}$ and $M{\downarrow}$ is a message

ELSE $\mathsf{if}\ M = N\ \mathsf{then}\ P\ \mathsf{else}\ Q \xrightarrow{\tau} Q$ where $M{\downarrow} \neq N{\downarrow}$ and $M{\downarrow}, N{\downarrow}$ are messages

COMM $\mathsf{out}(c, M_1).P_1 \parallel \mathsf{in}(c, M_2).P_2 \xrightarrow{\tau} P_1 \parallel P_2\theta$ where θ is such that $\mathsf{Dom}(\theta) = \mathcal{V}(M_2)$, $M_2\theta{\downarrow} = M_1{\downarrow}$, and $M_1{\downarrow}$ is a message.

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:

IN $\mathsf{in}(c, x).P \xrightarrow{\mathsf{in}(c,M)}_{\ell} P\{^{M}/_{x}\}$ where M is a message

OUT $\mathsf{out}(c, M).P \xrightarrow{\mathsf{out}(c,M{\downarrow})}_{\ell} P \parallel \{^{M{\downarrow}}/_{x}\}$ where x is a fresh variable and $M{\downarrow}$ is a message

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

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

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 14 *Going back to the handshake protocol described in Example 7, the derivation described below represents a normal execution of the protocol. For simplicity of this example we suppose that $x \notin fv(P)$.*

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

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

Example 15 *Continuing Example 8 we develop some transitions from*

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

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*

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

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

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

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 3 on bounded process verification.

2.3. Security Properties

This section presents mainly through examples how to formalise definitions of the most standard security properties. To prove that security properties hold for all adversaries, quantification over all contexts is required. However, in order to consider realistic adversary, we have to consider processes that are built using public function symbols only and we have to ensure that these processes are executable.

In practise, it may be difficult to reason with the quantification over all contexts. The labelled transition semantics aim to eliminate universal quantification of the context and is easier to manipulate. In this section, we rely on this semantics. Since our small process calculus does not allow us to model private channels, we do not have to consider the rule COMM. The attacker controls the entire network and can eavesdrop, block, intercept, and inject messages.

2.4. Secrecy

Intuitively, a protocol preserves the secrecy of some message M if an adversary cannot obtain M by constructing it from the outputs of the protocol. We can formalise the adversary as a process running in parallel with the protocol, that after constructing M outputs it on a public channel. The adversary process does not have any of the secrets of the protocol. As explained in introduction of this section, another possibility is to rely on the labelled semantics and to simply ask that in any reachable extended process, M can not be deduced from the frame. Below, you illustrate this property through several examples based on the Needham-Schroeder protocol.

Example 16 (Scenario 1) *Consider again the following process defined in Example 8 :*

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

We typically wish to ensure the secrecy of the nonce N_b. For this, we have to show that, for any extended process B such that $P_1 \rightarrow_{\ell}^{*} B$, we have that $\phi(B) \not\vdash N_b$. Actually, this secrecy property does not hold because of the attack described in Chapter 1. Note that, in this scenario, it is not reasonable to require the secrecy of N_a since N_a is generated by an honest agent for a dishonest one.

We may also want to express the secrecy of the nonce N_a received by $P_B(a,b)$. This means that we want that the value of y (this is the variable that represents the nonce N_a in the process $P_B(a,b)$ is not known by the attacker in any possible executions. For this,

we have to show that for each process B such that $P_1 \rightarrow_\ell^* B$ and in which the variable y has been instantiated by some message M, we have that $\phi(B) \not\vdash M$.

Example 17 (Scenario 2) *Consider now*

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

*In such a situation, neither N_a nor N_b can be required to remain secret: this depends on the inputs x_a and x_b. In this case, to express the secrecy of N_b, we can ask that for each process B such that $P_1 \rightarrow_\ell^* B$ and in which the variable x_a has been instantiated by an honest agent (*i.e. *not d), we have that $\phi(B) \not\vdash N_b$.*

To express secrecy of a nonce in the scope of a replication, we need extra material. Consider the following scenario

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

Intuitively, we wish that, in any copy of the process, in which $x_b \neq d$, then N_a is secret. Be careful that x_b is actually a local variable of the process and should actually be renamed in each copy. Similarly, N_a and N_b are renamed in each instance.

There are again several ways of specifying the desired properties For instance, we may split the processes in those for which x_b is bound to a honest party and those in which $x_b = d$ and then forget about the different copies in the specification. We may also enrich the calculus with status events. These status events are also very useful to express correspondence properties explained in the following section.

2.5. Correspondence Properties

Correspondence properties are used to capture relationships between events that can be expressed in the form "if an event e has been executed then event e has been previously executed." Moreover, these events may contain arguments. This will allows one to express agreement properties. To reason with correspondence properties, we have to annotate processes with events. These events will mark the different control points reached by the protocol.

We say that an extended process A has reached an event $\mathsf{event}(M_1, \ldots, M_n)$ if, and only if, there exist an evaluation context C, a plain process P and an extended process B such that $A \equiv C[\mathsf{event}(M_1, \ldots, M_n).P \parallel B]$.

Aliveness. This property is the weakest form of authentication in Lowe's hierarchy [19].

> A protocol satisfies *aliveness* if, whenever an honest agent completes a run of the protocol, apparently with another honest agent B, then B has previously run the protocol.

Note that B may not necessarily believe that he was running the protocol with A. Also, the agent B may not have run the protocol *recently*. The aliveness of principal B to initiator A can be specified in our formalism. First, we have to consider two status

events start and end. We insert them at the beginning and at the end of each role respectively. For instance, in $P_A(a, d)$, we insert $\mathsf{start}(a)$ at the beginning and $\mathsf{end}(a, d)$ at the end. This expresses the fact that the role is executed by a with d. We insert $\mathsf{start}(b)$ and $\mathsf{end}(b, a)$ in $P_B(a, b)$. Now, the aliveness property from the point of view of b can be specified as follows:

For any trace execution such that $P_1 \rightarrow_\ell A_1 \rightarrow_\ell \ldots \rightarrow_\ell A_n$ such that A_n has reached $\mathsf{end}(M_1, M_2)$ with $M_1 \neq d$ and $M_2 \neq d$, there exists i such that A_i has reached $\mathsf{start}(M_2)$. This corresponds to the fact that the property "if $\mathsf{end}(x, y)$ has been executed then $\mathsf{start}(y)$ has been previously executed when x and y are both honest agents." For the Needham-Schroeder public key protocol (*e.g.* Scenario 1) the aliveness property is satisfied.

Weak agreement. Weak agreement is slightly stronger than aliveness.

> A protocol guarantees *weak agreement* if, whenever an honest agent completes a run of the protocol, apparently with another honest agent B, then B has previously been running the protocol, apparently with A.

The weak agreement property can also been expressed in our formalism. We have again to add status events start and end in our specification. However, the predicate start will have also two parameters: $\mathsf{start}(a, d)$ expresses the fact that a has started a session with d. Now, the weak agreement property can be specified as follows:

For any trace execution such that $P_1 \rightarrow_\ell A_1 \rightarrow_\ell \ldots \rightarrow_\ell A_n$ such that A_n has reached $\mathsf{end}(M_1, M_2)$ with $M_1 \neq d$ and $M_2 \neq d$, there exists i such that A_i has reached $\mathsf{start}(M_2, M_1)$. For the Needham-Schroeder public key protocol, it is well-known that this property is not satisfied: b can complete a session apparently with a whereas a has never started a session with b. The property is already falsified on Scenario 1.

We can also express some refinements of these properties by distinguishing the case where an agent starts a session as an initiator or as a responder. Moreover, we can also express the fact that the two agents agreed on some message M, *e.g.* the value of a nonce or a key. This allows us to express the non-injective agreement security property. There are also stronger agreement properties, that would require the mapping from end to start to be injective.

2.6. Guessing Attacks

Guessing attacks are a kind of dictionary attack in which the password is supposed to be weak, *i.e.* part of a dictionary for which a brute force attack is feasible. A guessing attack works in two phases. In a first phase the attacker eavesdrops and interacts with one or several protocol sessions. In a second *offline* phase, the attacker tries each of the possible passwords on the data collected during the first phase. To resist against a guessing attack, the protocol must be designed such that the attacker cannot discover on the basis of the data collected whether his current guess of the password is the actual password or not.

The idea behind the definition is the following. Suppose the frame ϕ represents the information gained by the attacker by eavesdropping one or more sessions and let w

be the weak password. Then, we can represent resistance against guessing attacks by checking whether the attacker can distinguish a situation in which he guesses the correct password w and a situation in which he guesses an incorrect one, say w'. We model these two situations by adding $\{^{w}/_{x}\}$ (resp. $\{^{w'}/_{x}\}$) to the frame. We use static equivalence to capture the notion of indistinguishability. This definition is due to M. Baudet [7], inspired from the one of [13]. In our definition, we allow multiple shared secrets, and write $\overline{w}$ for a sequence of such secrets.

Definition 2.3 *Let $\phi \equiv \mathsf{new}\ \overline{w}.\phi'$ be a frame. We say that the frame ϕ is* resistant to guessing attacks *against $\overline{w}$ if*

$$\mathsf{new}\ \overline{w}.(\phi' \parallel \{^{\overline{w}}/_{\overline{x}}\}) \sim \mathsf{new}\ \overline{w}'.\mathsf{new}\ \overline{w}.(\phi' \parallel \{^{\overline{w}'}/_{\overline{x}}\})$$

where $\overline{w}'$ is a sequence of fresh names and $\overline{x}$ a sequence of variables such that $\overline{x} \cap \mathsf{Dom}(\phi) = \emptyset$.

Note that this definition is general w.r.t. to the equational theory and the number of guessable data items. Now, we can define what it means for a protocol to be resistant against guessing attacks.

Definition 2.4 *Let A be a process and $\overline{w} \subseteq \mathsf{bn}(A)$. We say that A is* resistant to guessing attacks *against $\overline{w}$ if, for every process B such that $A \rightarrow_{\ell}^{*} B$, we have that the frame $\phi(B)$ is resistant to guessing attacks against $\overline{w}$.*

Example 18 *Consider the handshake protocol described in Example 7. An interesting problem arises if the shared key w is a weak secret,* i.e. *vulnerable to brute-force off-line testing. In such a case, the protocol has a guessing attack against w. Indeed, we have that*

$$\mathsf{new}\ w.(P_A(w) \parallel P_B(w)) \rightarrow_{\ell}^{*} D$$

with $\phi(D) = \mathsf{new}\ w.\mathsf{new}\ n.(\{^{\mathsf{senc}(n,w)}/_{x_1}\} \parallel \{^{\mathsf{senc}(\mathsf{f}(n),w)}/_{x_2}\})$. The frame $\phi(D)$ is not resistant to guessing attacks against w. The test $\mathsf{f}(\mathsf{sdec}(x_1, x)) \stackrel{?}{=} \mathsf{sdec}(x_2, x)$ allows us to distinguish the two associated frames:

- $\mathsf{new}\ w.\mathsf{new}\ n.(\{^{\mathsf{senc}(n,w)}/_{x_1}\} \parallel \{^{\mathsf{senc}(\mathsf{f}(n),w)}/_{x_2}\} \parallel \{^{w}/_{x}\})$, *and*
- $\mathsf{new}\ w'.\mathsf{new}\ w.\mathsf{new}\ n.(\{^{\mathsf{senc}(n,w)}/_{x_1}\} \parallel \{^{\mathsf{senc}(\mathsf{f}(n),w)}/_{x_2}\} \parallel \{^{w'}/_{x}\})$.

Hence, the process $\mathsf{new}\ w.(P_A \parallel P_B)$ is not resistant to guessing attacks against w.

2.7. Equivalence Properties

The notion of indistinguishability is a powerful concept which allows us to reason about complex properties that cannot be expressed as secrecy or correspondence properties. Intuitively, two processes are said to be equivalent if an observer has no way to tell them apart. While static equivalence models indistinguishability of sequences of terms, it is also possible to lift it to an observational equivalence, *i.e.* indistinguishability of processes that interact with an arbitrary adversary. We define this observational equivalence by the means of a labelled bisimulation. The processes may perform different computa-

tions, but they have to look the same to an external observer. This notion allows us to define strong notions of secrecy and also privacy properties.

Before we formalise this notion of equivalence, we have to adapt the labelled semantics provided in Section 2.2.2. Indeed, we will now assume that the attacker can observe the interactions with the environment and we have to capture the fact that the attacker performs the same experiment on both processes. Intuitively, we want that, for any experiment s (sequence of labels) such that $A \xrightarrow{s}_\ell{}^* A'$, there exists B' such that $B \xrightarrow{s}_\ell{}^* B'$ and $\phi(A') \sim \phi(B')$. However, our labels are too fine grained.

Let $A = \mathsf{new}\ n.\mathsf{out}(c, n)$ and $B = \mathsf{new}\ n, k.\mathsf{out}(c, \mathsf{senc}(n, k))$. The only transitions that can be performed by A and B are as follows:

- $A \xrightarrow{\mathsf{out}(c,n)}_\ell \mathsf{new}\ n.\{^n/_x\}$, and
- $B \xrightarrow{\mathsf{out}(c,\mathsf{senc}(n,k))}_\ell \mathsf{new}\ n, k.\{^{\mathsf{senc}(n,k)}/_x\}$.

However, in reality an attacker has no way to distinguish these two processes since he will not see any difference between a fresh nonce and an encryption (he does not know the key). The same situation also occurs with the two processes $A = \mathsf{new}\ n.\mathsf{in}(c, y)$ and $B = \mathsf{new}\ n'.\mathsf{in}(c, y)$. We have that $A \xrightarrow{\mathsf{in}(c,n')} \mathsf{0}$ and B can not mimic this step. B is not allowed to use the name n' since it is restricted. Our labels contains too much information. We modify the IN and OUT rules as follows:

IN $\quad \mathsf{in}(c, x).P \xrightarrow{\mathsf{in}(c,M)}_\ell P\{^M/_x\}$ $\qquad$ where M is a message

OUT $\mathsf{out}(c, M).P \xrightarrow{\mathsf{out}(c,x)}_\ell P \parallel \{^{M\downarrow}/_x\}$ where x is a fresh variable and $M\downarrow$ is a message

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

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

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

Moreover, we now consider that structural equivalence is closed under α-renaming.

Example 19 *Going back to the handshake protocol described in Example 7, the derivation described below represents a normal execution of the protocol in the new labelled semantics.*

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

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

For every closed extended process A we define its set of traces, each trace consisting in a sequence of visible actions (*i.e.* different from τ) together with the sequence of sent messages:

$$\mathsf{trace}(A) = \{(s,\phi(B)) \mid A \xrightarrow{s}_\ell B \text{ for some } B\}.$$

Note that, in the new versions of our labelled semantics, the sent messages are exclusively stored in the frame and not in the sequence s (the outputs are made by "reference").

Definition 2.5 (trace equivalence $\approx_t$) *Let A and B be two closed extended processes, $A \sqsubseteq_t B$ if for every $(s,\varphi) \in \mathit{trace}(A)$, there exists $(s',\varphi') \in \mathit{trace}(B)$ such that $s = s'$ and $\varphi \sim \varphi'$. The extended processes A and B are* trace equivalent*, denoted by $A \approx_t B$, if $A \sqsubseteq_t B$ and $B \sqsubseteq_t A$.*

Example 20 *Consider the equational theory described in Example 2.*

$$\mathsf{new}\ s,k.\mathit{out}(c,\mathsf{senc}(s,k)).\mathit{in}(c,x).\ \mathsf{if}\ x = s\ \mathsf{then}\ \mathit{out}(c,a) \approx_t \mathsf{new}\ s,k.\mathit{out}(c,\mathsf{senc}(s,k)).\mathit{in}(c,x)$$

$$\mathsf{new}\ s.\mathit{out}(c,\mathsf{senc}(s,k)).\mathit{in}(c,x).\ \mathsf{if}\ x = s\ \mathsf{then}\ \mathit{out}(c,a) \not\approx_t \mathsf{new}\ s.\mathit{out}(c,\mathsf{senc}(s,k)).\mathit{in}(c,x)$$

$$\mathit{out}(c,a).\mathit{out}(c,a) \approx_t \mathit{out}(c,a) \parallel \mathit{out}(c,a)$$

$$\mathit{out}(c_1,a).\mathit{out}(c_2,a) \not\approx_t \mathit{out}(c_1,a) \parallel \mathit{out}(c_2,a)$$

$$\mathit{out}(c,a);\mathit{out}(c,b) \not\approx_t \mathit{out}(c,a) \parallel \mathit{out}(c,b)$$

Now, we develop an example to illustrate how this notion of equivalence can be used to formalise anonymity.

Example 21 *We consider a slightly simplified version of a protocol given in [4] designed for transmitting a secret without revealing its identity to other participants. In this protocol, A is willing to engage in communication with B and wants to reveal its identity to B. However, A does not want to compromise its privacy by revealing its identity or the identity of B more broadly. The participants A and B proceed as follows:*

$$A \to B \ : \ \mathsf{aenc}(\langle N_a, \mathit{pk}(A)\rangle, \mathit{pk}(B))$$

$$B \to A \ : \ \mathsf{aenc}(\langle N_a, \langle N_b, \mathit{pk}(B)\rangle\rangle, \mathit{pk}(A))$$

First A sends to B a nonce N_a and its public key encrypted with the public key of B. If the message is of the expected form then B sends to A the nonce N_a, a freshly generated nonce N_b and its public key, all of this being encrypted with the public key of A. Otherwise, B sends out a "decoy" message: $\mathsf{aenc}(N_b, \mathit{pk}(B))$. *This message should basically look like B's other message from the point of view of an outsider. This is important since the protocol is supposed to protect the identity of the participants.*

A session of role A played by agent a with b can be modelled by the following basic process where $M = \mathit{dec}(x, \mathit{sk}(a))$.

$$\begin{array}{l} A(a,b) \mathrel{\hat{=}} \\ \quad \mathit{out}(c, \mathsf{aenc}(\langle N_a, \mathit{pk}(a)\rangle, \mathit{pk}(b))). \\ \quad \mathit{in}(c,x). \\ \quad \mathsf{if}\ \langle \mathsf{proj}_1(M), \mathsf{proj}_2(\mathsf{proj}_2(M))\rangle = \langle N_a, \mathit{pk}(b)\rangle\ \mathsf{then}\ 0 \end{array}$$

Similarly, a session of role B played by agent b with a can be modelled by the basic process $B(b,a)$ where $N = \mathit{dec}(y, \mathsf{sk}(b))$.

$$\begin{array}{l} B(b,a) \mathrel{\hat{=}} \mathit{in}(c,y). \\ \qquad \mathsf{if}\ \mathsf{proj}_2(N) = \mathit{pk}(a)\ \mathsf{then}\ \mathit{out}(c, \mathsf{aenc}(\langle \mathsf{proj}_1(N), \langle N_b, \mathit{pk}(b)\rangle\rangle, \mathit{pk}(a))) \\ \qquad\qquad\qquad\qquad \mathsf{else}\ \mathit{out}(c, \mathsf{aenc}(N_b, \mathit{pk}(b))). \end{array}$$

Intuitively, this protocol preserves anonymity if an attacker cannot distinguish whether b is willing to talk to a (represented by the process $B(b,a)$) or willing to talk to a' (represented by the process $B(b,a')$), provided a, a' and b are honest participants. For illustration purposes, we also consider the process $B'(b,a)$ obtained from $B(b,a)$ by replacing the else *branch by* 0. *We will see that the "decoy" message plays a crucial role to ensure privacy.*

We can ask whether the two following processes P_{ex} and P'_{ex} are in equivalence:

- $P_{\mathsf{ex}} = \mathsf{new}\ N_a.\mathsf{new}\ N_b.[\, A(a,b) \parallel B(b,a) \parallel K(a,a',b)\,]$, *and*
- $P'_{\mathsf{ex}} = \mathsf{new}\ N_a.\mathsf{new}\ N_b.[\, A(a',b) \parallel B(b,a') \parallel K(a,a',b)\,]$.

where $K(a,a',b) = \mathit{out}(c, \mathit{pk}(a)).\mathit{out}(c, \mathit{pk}(a')).\mathit{out}(c, \mathit{pk}(b))$.

Actually, the 'decoy' message is crucial to have this equivalence, and thus anonymity. We have that $P_{\mathsf{ex}} \approx_t P'_{\mathsf{ex}}$ whereas:

$$\mathsf{new}\ N_a, N_b.[\, A(a,b) \parallel B'(b,a) \parallel K(a,a',b)\,] \not\approx_t \ \mathsf{new}\ N_a, N_b.[\, A(a',b) \parallel B'(b,a') \parallel K(a,a',b)\,].$$

Another notion of equivalence that has been quite well-studied is the notion of *observationally equivalent*. However, proofs of observational equivalences are difficult because of the universal quantification over all contexts. In the context of the applied pi calculus, it has been shown that observational equivalence coincides with labelled bisimilarity [3]. This should also hold in the calculus presented here.

Definition 2.6 (labelled bisimilarity $\approx_\ell$) Labeled bisimilarity *is the largest symmetric relation $\mathcal{R}$ on closed extended processes such that $A\ \mathcal{R}\ B$ implies*

1. $\phi(A) \sim \phi(B)$,
2. *if* $A \xrightarrow{\tau} A'$, *then* $B \rightarrow^* B'$ *and* $A'\ \mathcal{R}\ B'$ *for some* B',
3. *if* $A \xrightarrow{\alpha}_\ell A'$ *then* $B \rightarrow^* \xrightarrow{\alpha}_\ell \rightarrow^* B'$ *and* $A'\ \mathcal{R}\ B'$ *for some* B'.

It is easy to see that observational equivalence (or labelled bisimilarity) implies trace equivalence while the converse is false in general (see Example 22).

Lemma 2.7 *Let A and B be two extended processes: $A \approx_\ell B$ implies $A \approx_t B$.*

Example 22 *For convenience we introduce for this example a non-deterministic choice operator $+$ and extend the internal reduction by the rule $A + B \rightarrow A$ and structural equivalence by associativity and commutativity of $+$. Consider the two following processes:*

$$A = (\mathsf{out}(c, a).\mathsf{out}(c, b_1)) + (\mathsf{out}(c, a).\mathsf{out}(c, b_2))$$

$$B = \mathsf{out}(c, a).(\mathsf{out}(c, b_1) + \mathsf{out}(c, b_2))$$

We have that $A \approx_t B$ whereas $A \not\approx_\ell B$. Intuitively, after B's first move, B still has the choice of emitting b_1 or b_2, while A, trying to follow B's first move, is forced to choose between two states from which she can only emit one of the two.

The notion of labelled bisimilarity is quite strong but is also used to express privacy type properties. It is more or less a matter of taste to define anonymity w.r.t. trace equivalence or w.r.t. the stronger version with labelled bisimilarity.

2.8. Further Readings

The calculus presented in this chapter is very close to the applied pi calculus [3]. A presentation of this calculus in a tutorial style is also available [25]. Another calculus that is very close to the applied pi calculus and for which there exists a tutorial presentation is the spi-calculus [2].

2.9. Exercises

Exercice 5 ($\star$)
Consider the signature described in Example 2. Let

$$\phi = \mathsf{new}\ s, k_1.\{^{\mathsf{senc}(s,\langle k_1,k_2\rangle)}/_{x_1}, {}^{\mathsf{senc}(k_1,k_2)}/_{x_2}\}.$$

1. Is s deducible fom ϕ?
2. Could you enumerate the subterms of ϕ? Among these subterms, give those that are deducible.
3. Give a term that is deducible from ϕ and that is not a subterm.

Exercice 6 ($\star$)
Give a reasonable formalisation of the following protocol:

$$A \to B : \langle A, \{K\}_{\mathsf{pk}(B)} \rangle$$
$$B \to A : \langle B, \{K\}_{\mathsf{pk}(A)} \rangle$$

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.

Exercice 7 ($\star$)
Consider the formalisation of the Needham-Schroeder protocol as described in Example 8, and the following scenario (see Example 16).

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

Give the complete transition sequence that yields the attack on the secrecy of the nonce N_b.

Exercice 8 ($\star\star$)
Give a reasonable formalisation of the handshake protocol without using the conditional (if then else). Give a trace that exists in the model presented in 7 and that does not exist in this new formalisation.

3. Deducibility Constraints

In this chapter, we present the NP-complete decision procedure for a bounded number of sessions by H. Comon-Lundh *et al.* [12]. In this setting (*i.e.* finite number of sessions), deducibility constraint systems have become the standard model for verifying security properties, with a special focus on secrecy. Starting with a paper by J. Millen and V. Shmatikov [21], many results (*e.g.* [11,7]) have been obtained within this framework.

Here, we consider only symmetric/asymmetric encryptions, and pairing. We show that any deducibility constraint system can be transformed in (possibly several) much simpler deducibility constraint systems that are called solved forms, preserving all solutions of the original system, and not only its satisfiability. In other words, the deducibility constraint system represents in a symbolic way all the possible sequences of messages that are produced, following the protocol rules, whatever are the intruder's actions. This set of symbolic traces is infinite in general. Solved forms are a simple (and finite) representation of such traces. The procedure preserves all solutions. Hence, we can represent for instance, all attacks on the secrecy and not only decide if there exists one. Moreover, presenting the decision procedure using a small set of simplification rules yields more flexibility for further extensions and modifications.

3.1. Intruder Deduction problem

3.1.1. Preliminaries

An *inference rule* is a rule of the form $\dfrac{u_1 \ \ldots \ u_n}{u_0}$ where $u_0, u_1, \ldots, u_n$ are terms (with variables). An *inference system* is a set of inference rules.

Example 23 *The following inference system $\mathcal{I}_{\mathsf{DY}}$ represents the deduction capabilities of an attacker. We consider the signature $\mathcal{F} = \{\mathsf{senc}, \mathsf{aenc}, \langle \_, \_ \rangle, \mathsf{sk}\}$ and the underlying rewriting system $\mathcal{R}$ is empty. There are several possible ways of defining the intruder capabilities, we choose here the "implicit destructors" formulation, in which the destructors do not appear. This leads to an inference system that is slightly different from the one proposed in Section 2.1.3. For sake of simplicity, we make a confusion between the identity of an agent and his public key.*

$$\frac{x \quad y}{\langle x, y \rangle}\ \mathsf{P} \qquad \frac{x \quad y}{\mathsf{aenc}(x, y)}\ \mathsf{PKE} \qquad \frac{x \quad y}{\mathsf{senc}(x, y)}\ \mathsf{SE}$$

$$\frac{\langle x, y \rangle}{x}\ \mathsf{Left} \quad \frac{\langle x, y \rangle}{y}\ \mathsf{Right} \qquad \frac{\mathsf{aenc}(x, y) \quad \mathsf{sk}(y)}{x}\ \mathsf{PKD} \qquad \frac{\mathsf{senc}(x, y) \quad y}{x}\ \mathsf{SD}$$

The rules P, SE, *and* PKE *are* composition rules *whereas the rules* Left, Right, SD, *and* PKD *are* decomposition rules.

Definition 3.1 (proof) *Let $\mathcal{I}$ be an inference system. A proof Π of $T \vdash u$ in $\mathcal{I}$ is a tree such that:*

- *every leaf of Π is labelled with a term v such that $v \in T$,*

- *for every node labelled with v_0 having n sons labelled with $v_1, \ldots, v_n$, there is an instance of an inference rule with conclusion v_0 and hypotheses $v_1, \ldots, v_n$. We say that* Π *ends with this instance if the node is the root of* Π*,*
- *the root is labelled with u.*

We denote by $\mathsf{Hyp}(\Pi)$ the set of labels of the leaves of a proof Π and $\mathsf{Conc}(\Pi)$ is the label of the root of Π. $\mathsf{Steps}(\Pi)$ is the set of labels of all nodes of Π. The *size* of a proof Π is the number of nodes in it. A proof Π of $T \vdash u$ is *minimal* if it does not exist any proof Π' of $T \vdash u$ having a size strictly smaller than the size of Π.

Example 24 *Let* $\phi = \mathsf{new}\ a, b, s.\ \{^{\langle \mathsf{senc}(s,\langle a,b\rangle),a\rangle}/_{x_1},\ ^{\mathsf{senc}(b,a)}/_{x_2}\}$. *We may ask whether s is deducible from ϕ,* i.e. *does there exist a proof of*

$$\langle \mathsf{senc}(s, \langle a, b\rangle), a\rangle,\ \mathsf{senc}(b, a) \vdash s.$$

Such a proof is given below:

$$\dfrac{\dfrac{\langle \mathsf{senc}(s,\langle a,b\rangle),a\rangle}{\mathsf{senc}(s,\langle a,b\rangle)} \quad \dfrac{\dfrac{\langle \mathsf{senc}(s,\langle a,b\rangle),a\rangle}{a} \quad \dfrac{\mathsf{senc}(b,a) \qquad \dfrac{\langle \mathsf{senc}(s,\langle a,b\rangle),a\rangle}{a}}{b}}{\langle a,b\rangle}}{s}$$

The problem whether an intruder can gain certain information s from a set of knowledge T, *i.e.* whether there is a proof of $T \vdash s$ is called the *intruder deduction problem*.

Intruder deduction problem (for a fixed inference system $\mathcal{I}$)

INPUT: a finite set of terms T, and a term s (the secret).
OUTPUT: Does there exist a proof of $T \vdash s$?

This definition is in-line with the concept of deduction introduced in Section 2.1.3. Here, we do not explicitly rely on the concept of frame. Note that for deduction, the ordering in which the messages have been sent is not relevant. Moreover, restriction on names are not necessary. It is assumed that each name is restricted.

3.1.2. Decidability via Locality

To show that the intruder deduction problem is decidable (in PTIME) for an inference system $\mathcal{I}$, we use the notion of *locality* introduced by D. McAllester [20].

Definition 3.2 (locality) *Let $\mathcal{I}$ be an inference system. The system $\mathcal{I}$ is* local *if whenever $T \vdash u$ in $\mathcal{I}$, there exists a proof* Π *of $T \vdash u$ such that $\mathsf{Steps}(\Pi) \subseteq \mathsf{St}(T \cup \{u\})$.*

Given an inference system $\mathcal{I}$, to establish that the intruder deduction problem is decidable, it is actually sufficient to prove that:

1. *a locality result* for the inference system $\mathcal{I}$: checking the existence of a proof of $T \vdash u$ amounts to checking the existence of a local proof that only contains subterms of u and T (there is a polynomial number of subterms),
2. *a one-step-deducibility result* to ensure that we can test (in PTIME) whether a term is deducible in one step from a set of terms by using an instance of one of the inference rules. This result trivially holds for the inference system presented in Example 23.

Then, the existence of a local proof of $T \vdash u$ can be checked in polynomial time by saturation of T with terms deducible in one-step. Thanks to locality, the number of iteration to obtain a saturated set is bounded by the number of terms that can be involved in a local proof. This yields a PTIME algorithm.

Lemma 3.3 (locality) *Let T be a set of terms and u be a term. A minimal proof Π of $T \vdash u$ only contains terms in $\mathit{St}(T \cup \{u\})$,* i.e. *$\mathit{Steps}(\Pi) \subseteq \mathit{St}(T \cup \{u\})$. Moreover, if Π is reduced to a leaf or ends with a decomposition rule, then we have that $\mathit{Steps}(\Pi) \subseteq \mathit{St}(T)$.*

Proof: Let Π be a minimal proof of $T \vdash u$. We prove the result by induction on the size of the proof Π.

Base case: In such a case, the proof Π is reduced to a leaf and we easily conclude.
Induction step: We have that:

$$\Pi = \begin{cases} \Pi_1 \qquad \Pi_n \\ \dfrac{u_1 \quad \cdots \quad u_n}{u}\ \mathsf{R} \end{cases}$$

We distinguish several cases depending on the last inference rule of Π.

- If R is a composition rule, then $u_1, \ldots, u_n$ are subterms of u and we easily conclude by relying on our induction hypothesis.
- If R is a projection rule (say proj_1), then $u_1 = \langle u, v \rangle$ for some v. In such a case, by minimality of Π, we know that Π_1 does not end with a composition rule. Hence, by relying on our induction hypothesis, we have that $\mathsf{Steps}(\Pi_1) \subseteq \mathsf{St}(T)$, and thus $u_1 \in \mathsf{St}(T)$. Moreover, we have that $u \in \mathsf{St}(u_1)$, and thus $u \in \mathsf{St}(T)$. This allows us to conclude that $\mathsf{Steps}(\Pi) \subseteq \mathsf{St}(T)$.

The cases where Π ends with a decryption rule (symmetric and asymmetric) can be done in a similar way. □ □

Proposition 3.1 *The intruder deduction problem is decidable in PTIME for $\mathcal{I}_{\mathsf{DY}}$. Actually, this problem is PTIME complete.*

The PTIME-hardness can be proved by a reduction from HORNSAT.

The concept of locality has been used to establish decidability of several inference systems. For instance, we may want to model digital signature, exclusive or operator, commutative encryption, ...

3.2. Deducibility constraints

Assume processes without replication. Then the transition system is finite in depth but might be infinitely branching, as we saw in Example 15. The idea then is to represent in a simple symbolic way the set of terms that satisfy the required conditions. This is what we formalise now.

Definition 3.4 *A* Deducibility constraint system *is either* $\bot$ *or a conjunction of deducibility constraints of the form:*

$$T_1 \stackrel{?}{\vdash} u_1 \wedge \ldots \wedge T_n \stackrel{?}{\vdash} u_n$$

in which $T_1, \ldots, T_n$ *are finite sets of terms,* $u_1, \ldots, u_n$ *are terms. Moreover, we assume that the constraints can be ordered in such a way that:*

- *monotonicity:* $\emptyset \neq T_1 \subseteq T_2 \cdots \subseteq T_n$
- *origination: for every* i*, we have that* $\mathcal{V}(T_i) \subseteq \mathcal{V}(u_1, \ldots, u_{i-1})$

Intuitively, the sets T_i correspond to messages that have been sent on the network, while $u_1, \ldots, u_n$ are the messages that are expected by the processes, hence have to be constructed by the environment. The first condition, called *monotonicity* reflects the fact that the set of messages that have been sent on the network can only increase. In other words, the ordering on the atomic deducibility constraints is a temporal ordering of actions. The second condition (called *origination*) reflects the properties of our processes: variables that occur in a message sent on the network must appear before in messages received from the network.

Definition 3.5 (T_x) *Let* $\mathcal{C} = T_1 \stackrel{?}{\vdash} u_1 \ \wedge \ \ldots \ \wedge \ T_n \stackrel{?}{\vdash} u_n$ *be a deducibility constraint system and* x *be a variable that occurs in* $\mathcal{C}$*.* T_x *is the minimal set (w.r.t. inclusion) among the sets* $T_1, \ldots, T_n$ *such that* $T \stackrel{?}{\vdash} u \in \mathcal{C}$ *and* $x \in \mathcal{V}(u)$*.*

Thanks to the monotonicity and the origination properties, for any $x \in \mathcal{V}(\mathcal{C})$, the set T_x exists and is uniquely defined.

Such constraint systems may be enriched with equations/disequations between terms or other constraints, that correspond to the conditions in the process calculus. We consider (for now) only these simple constraints.

Definition 3.6 (solution) *Let* $\mathcal{I}$ *be an inference system. A substitution* σ *is a* solution *of a deducibility constraint system* $\mathcal{C} = T_1 \stackrel{?}{\vdash} u_1 \wedge \ldots \wedge T_n \stackrel{?}{\vdash} u_n$ *if there exists a proof of* $T_i\sigma \vdash u_i\sigma$ *in* $\mathcal{I}$ *for every* $i \in \{1, \ldots, n\}$*.*

Example 25 *Consider the constraints corresponding to one of the possible Needham-Schroeder symbolic trace. We give explicitly the free names to the attacker and assume that all names that are not explicitly given are (supposedly) secret:*

$$\mathcal{C} \hat{=} \begin{cases} a,\ b,\ d,\ \mathsf{sk}(d),\ \mathsf{aenc}(\langle a, N_a\rangle, d) \overset{?}{\vdash} \mathsf{aenc}(\langle a, x\rangle, b) \\ a,\ b,\ d,\ \mathsf{sk}(d),\ \mathsf{aenc}(\langle a, N_a\rangle, d),\ \mathsf{aenc}(\langle x, N_b\rangle, a) \overset{?}{\vdash} \mathsf{aenc}(\langle N_a, y\rangle, a) \end{cases}$$

The failure of the secrecy of N_b (for this scenario) is given by the additional constraint:

$$a,\ b,\ d, \mathsf{sk}(d), \mathsf{aenc}(\langle a, N_a\rangle, d),\ \mathsf{aenc}(\langle x, N_b\rangle, a),\ \mathsf{aenc}(y, d) \overset{?}{\vdash} N_b$$

A solution of $\mathcal{C}$ in $\mathcal{I}_{\mathsf{DY}}$ is $\sigma = \{x \mapsto N_a, y \mapsto N_b\}$.

3.3. Decision Procedure

We describe here a non-deterministic simplification procedure. It can be simplified in many respects, but we will see that the problem of deciding whether a constraint system has at least one solution is NP-complete anyway (for the $\mathcal{I}_{\mathsf{DY}}$ inference system given in Example 23). Many parts of this section, including the set of simplification rules, are borrowed from [12].

3.3.1. Simplification Rules

We prove that any deducibility constraint system can be transformed into simpler ones, called *solved*. Such simplified constraints are then used to decide the security properties.

R_1	$\mathcal{C} \wedge T \overset{?}{\vdash} u$	$\rightsquigarrow$ $\mathcal{C}$	if $T \cup \{x \mid (T' \overset{?}{\vdash} x) \in \mathcal{C}, T' \subsetneq T\} \vdash u$
R_2	$C \wedge T \overset{?}{\vdash} u$	$\rightsquigarrow_\sigma \mathcal{C}\sigma \wedge T\sigma \overset{?}{\vdash} u\sigma$	if $t \in \mathsf{St}(T), \sigma = \mathsf{mgu}(t, u), t \neq u$ t, u not variables
R_3	$\mathcal{C} \wedge T \overset{?}{\vdash} u$	$\rightsquigarrow_\sigma \mathcal{C}\sigma \wedge T\sigma \overset{?}{\vdash} u\sigma$	if $t_1, t_2 \in \mathsf{St}(T), \sigma = \mathsf{mgu}(t_1, t_2)$, and $t_1 \neq t_2$
R_4	$\mathcal{C} \wedge T \overset{?}{\vdash} u$	$\rightsquigarrow$ $\bot$	if $\mathcal{V}(T \cup \{u\}) = \emptyset$ and $T \not\vdash u$
R_f	$\mathcal{C} \wedge T \overset{?}{\vdash} \mathsf{f}(u, v)$	$\rightsquigarrow$ $\mathcal{C} \wedge T \overset{?}{\vdash} u \wedge T \overset{?}{\vdash} v$	for $\mathsf{f} \in \{\langle\ ,\ \rangle, \mathsf{senc}, \mathsf{aenc}\}$

Figure 5. Simplification rules.

All the rules are indexed by a substitution (when there is no index then the identity substitution is assumed). We write $\mathcal{C} \rightsquigarrow^*_\sigma \mathcal{C}'$ if there are constraint systems $\mathcal{C}_1, \ldots, \mathcal{C}_n$ such that $\mathcal{C} \rightsquigarrow_{\sigma_0} \mathcal{C}_1 \rightsquigarrow_{\sigma_1} \ldots \rightsquigarrow_{\sigma_n} \mathcal{C}'$ and $\sigma = \sigma_0\sigma_1 \ldots \sigma_n$. We denote by $\sigma = \mathsf{mgu}(u, v)$ a most general unifier of u and v, such that $\mathcal{V}(v\sigma, u\sigma) \subseteq \mathcal{V}(v, u)$.

A constraint system is called *solved* if it is different from $\bot$ and if each of its constraints is of the form $T \overset{?}{\vdash} x$, where x is a variable. Note that the empty constraint system is solved. Solved constraint systems are particularly simple since they always have a solution. Indeed, let T_1 be the smallest (w.r.t. inclusion) left-hand side of a constraint. From the definition of a constraint system we have that $T_1 \neq \emptyset$ and has no variable. Then

the substitution τ defined by $x\tau = t_1$ where $t_1 \in T_1$ for every variable x, is a solution since $T \vdash x\theta$ for any constraint $T \overset{?}{\vdash} x$ of the solved constraint system.

Given a constraint system $\mathcal{C}$, we say that T_i is a *minimal unsolved left-hand side* of $\mathcal{C}$ if T_i is a left-hand side of $\mathcal{C}$ and for all $T \overset{?}{\vdash} u \in \mathcal{C}$ such that $T \subsetneq T_i$, we have that u is a variable.

Lemma 3.7 *The simplification rules transform a deducibility constraint system into a deducibility constraint system.*

Theorem 3.8 *Let $\mathcal{C}$ be an unsolved constraint system.*

1. *(Termination) There is no infinite chain $\mathcal{C} \rightsquigarrow_{\sigma_1} \mathcal{C}_1 \ldots \rightsquigarrow_{\sigma_n} \mathcal{C}_n$.*
2. *(Correctness) If $\mathcal{C} \rightsquigarrow^*_\sigma \mathcal{C}'$ for some constraint system $\mathcal{C}'$ and some substitution σ and if θ is a solution of $\mathcal{C}'$ then $\sigma\theta$ is a solution of $\mathcal{C}$.*
3. *(Completeness) If θ is a solution of $\mathcal{C}$, then there exist a solved constraint system $\mathcal{C}'$ and substitutions σ, θ' such that $\theta = \sigma\theta'$, $\mathcal{C} \rightsquigarrow^*_\sigma \mathcal{C}'$ and θ' is a solution of $\mathcal{C}'$.*

Termination and correctness are quite easy to show. For termination, it is easy to see that the number of variables is non-increasing. Furthermore, this number strictly decreases by the rules R_2 and R_3. Any other rule strictly reduces the total size of the right hand sides of the constraint (here, the "size" is the number of symbols in the term). Completeness is more involved and its proof is detailed in Section 3.3.2. Getting a polynomial bound on the length of simplification sequences requires to consider a particular strategy.

3.3.2. Completeness

First, we show that proofs considered in solutions of constraints can be narrowed to so-called *simple proofs*. Let $T_1 \subseteq T_2 \subseteq \ldots \subseteq T_n$. We say that a proof Π of $T_i \vdash u$ is *left minimal* if, whenever there is a proof of $T_j \vdash u$ for some $j < i$, then Π is also a proof of $T_j \vdash u$. In other words, the left-minimal proofs are those that can be performed in a minimal T_j . We say that a proof is *simple* if all its subproofs are left minimal and there is no repeated label on any branch. Note that a subproof of a simple proof is simple.

Lemma 3.9 *Let $T_1 \subseteq T_2 \subseteq \ldots \subseteq T_n$ be a sequence of sets of terms and u be a term such that $T_i \vdash u$. There exists a simple proof Π of $T_i \vdash u$.*

Proof: Let i be a minimal index for which there is a proof of $T_i \vdash u$. Thanks to Lemma 3.3, there is a local proof Π_0 of $T_i \vdash u$. We prove the lemma by induction on the size of Π_0.

Base case: Π_0 is reduced to a leaf. In such a case, Π_0 is a simple proof.
Induction step: Consider the last rule in the proof of u:

$$\Pi_0 = \left\{ \dfrac{\begin{array}{ccc}\Pi_1 & & \Pi_n \\ u_1 & \cdots & u_n\end{array}}{u}\ \mathsf{R} \right.$$

For every $j = 1, ..., n$, we have that Π_j is a proof of $T_i \vdash u_j$. By induction hypothesis, there are simple proofs Π'_j of u_j. If u appears as a node in some of these proofs, let Π be the corresponding subproof and we get the desired result. Otherwise, let

$$\Pi = \left\{ \dfrac{\begin{array}{ccc} \Pi'_1 & \cdots & \Pi'_n \\ u_1 & & u_n \end{array}}{u}\ \mathsf{R} \right.$$

The proof Π is a simple proof of u. □ □

Lemma 3.10 *Let $\mathcal{C}$ be an unsolved constraint system, θ be a solution of $\mathcal{C}$ and $T_i \overset{?}{\vdash} u_i$ be a minimal unsolved constraint of $\mathcal{C}$. Let u be a term. If there is a simple proof of $T_i\theta \vdash u$ having the last rule an axiom or a decomposition then there is $t \in \mathsf{St}(T_i) \setminus \mathcal{X}$ such that $t\theta = u$.*

Proof: Let Π be a simple proof of $T_i\theta \vdash u$ such that its last rule is an axiom or a decomposition. Let j be the minimal indice such that $T_j\theta \vdash u$. Note that $j \leq i$ and by definition of a simple proof, we have that Π is also a simple proof of $T_j\theta \vdash u$.

- The last rule is an axiom. Then $u \in T_j\theta$. There is $t \in T_j$ (thus $t \in \mathsf{St}(T_j)$) such that $t\theta = u$. If t is a variable then $T_t \overset{?}{\vdash} t$ is a constraint in $\mathcal{C}$ with $T_t \subsetneq T_j$ (see the definition of a constraint system). Hence $T_t\theta \vdash t\theta$, that is $T_t\theta \vdash u$, which contradicts the minimality of j. Thus, as required, t is not a variable.
- The last rule is a decomposition. Suppose that it is a symmetric decryption. That is, there is w such that $T_j\theta \vdash \mathsf{senc}(u, w)$, and $T_j\theta \vdash w$. By simplicity of the proof, the last rule applied when obtaining $\mathsf{senc}(u, w)$ is an axiom or a decomposition, otherwise the same node would appear twice. Then, applying the induction hypothesis we have that there is $t \in \mathsf{St}(T_j) \setminus \mathcal{X}$ such that $t\theta = \mathsf{senc}(u, w)$. It follows that $t = \mathsf{senc}(t', t'')$ with $t'\theta = u$. If t' is a variable then $T_{t'}\theta \vdash t'\theta$. That is $T_{t'}\theta \vdash u$, which again contradicts the minimality of j. Hence t' is not variable, as required.

For the other decomposition rules the same reasoning holds. □ □

Lemma 3.11 *Let $\mathcal{C} = T_0 \overset{?}{\vdash} x_0, \ldots, T_{i-1} \overset{?}{\vdash} x_{i-1}, T_i \overset{?}{\vdash} u, \ldots$ be a constraint system and σ be a solution of $\mathcal{C}$ such that*

1. *T_i does not contain two distinct subterms t_1, t_2 with $t_1\sigma = t_2\sigma$,*
2. *u is a non-variable subterm of T_i.*

Then $T'_i \vdash u$, where $T'_i = T_i \cup \{x \mid (T \overset{?}{\vdash} x) \in \mathcal{C}, T \subsetneq T_i\}$.

Proof: Let j be minimal such that $T_j\sigma \vdash u\sigma$. Thus $j \leq i$ and $T_j \subseteq T_i$. Consider a simple proof Π of $T_j\sigma \vdash u\sigma$. We reason by induction on the depth of Π.

Base case: Π is reduced to a leaf. Then there is $t \in T_j$ such that $t\sigma = u\sigma$. By hypothesis 1, we deduce that $t = u$. Hence, we have that $u \in T_j$ and thus $T_i' \vdash u$, as required.

Induction step: We analyse the different cases, depending on the last rule R of Π:

- *Case* R is a composition rule. Assume for example that $\mathsf{R} = \mathsf{SE}$. In such a case, we have that:

$$\Pi = \left\{ \begin{array}{cc} \Pi_1 & \Pi_2 \\ v_1 & v_2 \\ \hline \mathsf{senc}(v_1, v_2) \end{array} \right.$$

  with $u\sigma = \mathsf{senc}(v_1, v_2)$. Since u is not a variable, $u = \mathsf{senc}(u_1, u_2)$, $u_1\sigma = v_1$, and $u_2\sigma = v_2$. If u_1 (resp. u_2) is a variable then u_1 (resp. u_2) belongs to $\mathcal{V}(T_i)$ since $u \in \mathsf{St}(T_i)$. Again, this implies $u_1 \in T_i'$ (resp. $u_2 \in T_i'$). Otherwise, u_1 (resp. u_2) is not a variable. Then, by induction hypothesis, $T_i' \vdash u_1$ (resp. $T_i' \vdash u_2$). Hence in both cases we have that $T_i' \vdash u_1$ and $T_i' \vdash u_2$. This allows us to conclude that $T_i' \vdash u$.
- *Case* $\mathsf{R} = \mathsf{SD}$. In such a case, there is w such that $T_j\sigma \vdash \mathsf{senc}(u\sigma, w)$, and $T_j\sigma \vdash w$:

$$\Pi = \left\{ \begin{array}{cc} \Pi_1 & \Pi_2 \\ \mathsf{senc}(u\sigma, w) & w \\ \hline u\sigma \end{array} \right.$$

  By simplicity, the last rule of the proof Π_1 is a decomposition or an axiom. By Lemma 3.10, there is $t \in \mathsf{St}(T_j) \setminus \mathcal{X}$ such that $t\sigma = \mathsf{senc}(u\sigma, w)$. Let $t = \mathsf{senc}(t_1, t_2)$ with $t_1\sigma = u\sigma$, and $t_2\sigma = w$. By induction hypothesis, $T_i' \vdash t$. Since $t_1\sigma = u\sigma$, by hypothesis 1, we have that $t_1 = u$.
  Now, if t_2 is a variable, and since $t_2 \in \mathcal{V}(T_i)$, we have that $T_{t_2} \subsetneq T_i$ and thus $t_2 \in T_i'$. If t_2 is not a variable, then, from $T_j\sigma \vdash t_2\sigma$ and by induction hypothesis, $T_i' \vdash t_2$. So, in any case, $T_i' \vdash t_2$.
  Hence, we have both that $T_i' \vdash \mathsf{senc}(u, t_2)$ and $T_i' \vdash t_2$, from which we conclude that $T_i' \vdash u$, by symmetric decryption.
- *Case* $\mathsf{R} = \mathsf{PKD}$. In such a case, there is w such that $T_j\sigma \vdash \mathsf{sk}(w)$ and $T_j\sigma \vdash \mathsf{aenc}(u\sigma, w)$. As in the previous case, there is $t \in \mathsf{St}(T_j) \setminus \mathcal{X}$ such that $t\sigma = \mathsf{aenc}(u\sigma, w)$. By induction hypothesis, $T_i' \vdash t$. Let $t = \mathsf{aenc}(t_1, t_2)$. As in the previous case, we have that $t_1\sigma = u\sigma$, and thus $t_1 = u$ (thanks to hypothesis 1).
  The last rule in the proof of $T_j\sigma \vdash \mathsf{sk}(w)$ is a decomposition (no composition rule can yield a term headed with $\mathsf{sk}(\_)$). Then, by Lemma 3.10 (T_j satisfies the hypotheses of the lemma since $T_j \subseteq T_i$), there is a non-variable subterm $w_1 \in \mathsf{St}(T_j)$ such that $w_1\sigma = \mathsf{sk}(w)$. Let $w_1 = \mathsf{sk}(w_2)$. By induction hypothesis, $T_j' \vdash \mathsf{sk}(w_2)$. Moreover, since $w_2\sigma = t_2\sigma$, by hypothesis 2, we have that $w_2 = t_2$, Finally, from $T_i' \vdash \mathsf{aenc}(u, w_2)$ and $T_i' \vdash \mathsf{sk}(w_2)$, we conclude that $T_i' \vdash u$.

The proof is similar for the other decomposition rules. □ □

Proposition 3.2 (Completeness for one step) *If $\mathcal{C}$ is an unsolved deducibility constraint system and σ is a solution of $\mathcal{C}$, then there is a deducibility constraint system $\mathcal{C}'$, a substitution τ, and a solution σ' of $\mathcal{C}'$ such that $\mathcal{C} \rightsquigarrow_\tau \mathcal{C}'$ and $\sigma = \tau\sigma'$.*

Proof: Let $\mathcal{C}$ be an unsolved constraint system and σ be a solution of $\mathcal{C}$. We show that there is a constraint system $\mathcal{C}'$ and a solution σ' of $\mathcal{C}'$ such that $\mathcal{C} \rightsquigarrow_\tau \mathcal{C}'$ and $\sigma = \tau\sigma'$.

Consider a minimal unsolved constraint $T_i \stackrel{?}{\vdash} u_i$ such that u_i is not a variable. We have that $T_i\sigma \vdash u_i\sigma$. Consider a simple proof Π of $T_i\sigma \vdash u_i\sigma$. We analyse the different cases depending on the last rule of Π.

1. *The last rule is a composition.* Suppose that it is the pairing rule. That is, there are w_1, w_2 such that $T_i\theta \vdash w_1$, $T_i\theta \vdash w_2$ and $\langle w_1, w_2\rangle = u_i\theta$. Since u_i is not a variable there exists u', u'' such that $u_i = \langle u', u''\rangle$. Hence we can apply the simplification rule $\mathsf{R_f}$ in order to obtain $\mathcal{C}'$. Since $u'\theta = w_1$ and $u''\theta = w_2$, the substitution θ is also a solution to $\mathcal{C}'$. For the other composition rules the same reasoning holds.
2. *The last rule is an axiom or a decomposition.* Applying Lemma 3.10 we obtain that there is $t \in \mathsf{St}(T_i) \setminus \mathcal{X}$ such that $t\theta = u_i\theta$. We can have the following two possibilities:
   (a) If $t \neq u_i$ then we apply the simplification rule R_2.
   (b) Otherwise, if $t = u_i$, then $u_i \in \mathsf{St}(T_i)$ and we already know that u_i is not a variable. We consider two cases:
      i. There are two distinct terms $t_1, t_2 \in \mathsf{St}(T)$ such that $t_1\theta = t_2\theta$. Then we apply the simplification rule R_3.
      ii. Otherwise, the simplification rule R_1 can be applied (Lemma 3.11). □

□

3.3.3. Complexity

The termination stated in Theorem 3.8 does not provide with tight complexity bounds. In fact, applying the simplification rules may lead to branches of exponential length in the size of the constraint system [12]. Inspecting the completeness proof, there is still some room for choosing a strategy to ensure that the length of each branch is polynomially bounded in $\mathcal{C}$ (while keeping completeness). Note that correctness is independent of the order of the rules application.

Moreover, for any suitable representation of terms, we have that $|u\sigma, v\sigma| < |u, v|$ where $\sigma = \mathsf{mgu}(u, v)$. Hence, if we use a DAG representation of terms, when $\mathcal{C} \rightsquigarrow_\sigma^* \mathcal{C}'$, we have that the size of $\mathcal{C}'$ is polynomially bounded in the size of $\mathcal{C}$. As a consequence, the security problem is in co-NP and it is actually co-NP-complete [24]. The NP-hardness can be established with a reduction from 3-SAT.

3.4. Further Readings

Many parts of this section are borrowed from [12]. Hence, more details can be found in this paper. Another decision procedure based on constraint simplification rules has been

proposed by J. Millen and V. Shmatikov [21]. Many results (*e.g.* [11,7]) have been obtained within this framework. In particular, this framework has been extended by several authors to deal with algebraic properties of cryptographic primitives.

3.5. Exercises

Exercice 9 ($\star$)
Consider the following inference system:

$$\frac{x \quad y}{\langle x, y\rangle} \qquad \frac{\langle x, y\rangle}{x} \qquad \frac{\langle x, y\rangle}{y} \qquad \frac{x \quad y}{\mathsf{senc}(x, y)} \qquad \frac{\mathsf{senc}(x, y) \quad y}{x}$$

Let $T = \{\mathsf{senc}(s, \langle k_1, k_2\rangle),\ \mathsf{senc}(k_1, k_3),\ k_3,\ k_2\}$.

1. Enumerate all the subterms of T.
2. The term s is deducible from T. Give a derivation witnessing this fact.
3. Among the subterms of T, give those that are deducible.
4. Give a term u that is not a subterm of T and such that $T \vdash u$.

Exercice 10 ($\star\star\star$)
Consider the following inference system:

$$\frac{x \quad y}{\langle x, y\rangle} \qquad \frac{\langle x, y\rangle}{x} \qquad \frac{\langle x, y\rangle}{y} \qquad \frac{x \quad y}{\mathsf{senc}(x, y)} \qquad \frac{\mathsf{senc}(x, y) \quad y}{x}$$

In order to decide whether a term s is deducible from a set of terms T in the inference system described above, we propose the following algorithm:

Algorithm:

1. Apply as much as possible the decryption and the projection rules. This leads to a set of terms called $\mathsf{analz}(T)$.
2. Check whether s can be obtained by applying the encryption and the pairing rules. The (infinite) set of terms obtained by applying the composition rules is denoted $\mathsf{synth}(\mathsf{analz}(T))$.

If $s \in \mathsf{synth}(\mathsf{analz}(T))$ then the algorithm return *yes*. Otherwise, it returns *no*.

1. Show that this algorithm terminates.
2. Show that this algorithm is sound, *i.e.* if the algorithm returns yes then $T \vdash s$.
3. The algorithm is not complete, *i.e.* there exist T and s such that $T \vdash s$, and for which the algorithm returns no. Find an example illustrating this fact.
4. Give an hypothesis on T that allows one to restore completeness.
5. Show that the algorithm is complete when this hypothesis is fulfilled.

Exercice 11 ($\star$)
We consider the following inference system allowing us to model asymmetric encryption.

$$\frac{x \quad y}{\mathsf{aenc}(x,y)} \qquad \frac{\mathsf{aenc}(x,\mathsf{pk}(z)) \quad \mathsf{sk}(z)}{x} \qquad \frac{z}{\mathsf{pk}(z)}$$

Is this inference system local, or not? If so, give a proof. If not, give a derivation witnessing this fact.

Exercice 12 ($\star\star$)
Consider the following inference system allowing us to model digital signature.

$$\frac{x \quad \mathsf{sk}(z)}{\mathsf{sign}(x,\mathsf{sk}(z))} \qquad \frac{\mathsf{sign}(x,\mathsf{sk}(z)) \quad \mathsf{vk}(z)}{x} \qquad \frac{z}{\mathsf{vk}(z)}$$

1. This inference system is not local according to Definition 3.2. Give an example witnessing this fact.
2. Show that the intruder deduction problem is decidable.
   You can use the technique described in this chapter and extend the notion of subterm to restore the locality property.

Exercice 13 ($\star$)
We consider the signature and the inference system given in Example 23. Let $T_0 = \{a,\ b,\ c,\ \mathsf{sk}(c),\ \mathsf{aenc}(\langle a, \mathsf{aenc}(s,b)\rangle, b)\}$ and $\mathcal{C} = \{T_0 \stackrel{?}{\vdash} \mathsf{aenc}(\langle a, \mathsf{aenc}(x_1,b)\rangle, b)\}$. What are the solutions of $\mathcal{C}$?

Exercice 14 ($\star\star$)
Consider the following protocol (defined informally):

$$A \to B : \langle \mathsf{aenc}(k_1, \mathsf{pk}(b)), \mathsf{aenc}(k_2, \mathsf{pk}(b))\rangle$$
$$B \to A : \mathsf{senc}(k_1, k_2)$$

Here k_1 and k_2 represent two keys that are freshly generated by A at the beginning of each session.

1. Write formally the processes corresponding to an instance of the role A played by two honest agents a,b and an instance of the role B with the same two honest agents
2. Give a deduction constraint system corresponding to the only relevant symbolic trace for the processes of the previous question.
3. Apply the simplification rules to this constraint system and derive all possible attacks on the secrecy of k_1 (resp. k_2) for this scenario.

Exercice 15 ($\star$)
Give an example showing that the rule R_3 is necessary for the completeness of the procedure. More precisely, this example has to show that Proposition 3.2 will be wrong without this rule.

Exercice 16 ($\star\star$)
We consider the following variant of the rule R_3

$$\mathsf{R}'_3 : \quad \mathcal{C} \wedge T \overset{?}{\vdash} u \quad \rightsquigarrow_\sigma \mathcal{C}\sigma \wedge T\sigma \overset{?}{\vdash} u\sigma \quad \text{if } t_1, t_2 \in \mathsf{St}(T) \setminus \mathcal{X}, \sigma = \mathsf{mgu}(t_1, t_2), \text{ and } t_1 \neq t_2$$

1. Show that R'_3 is not sufficient to restore completeness, *i.e.* give an example witnessing the fact that Proposition 3.2 is wrong if we use the rule R'_3 instead of R_3.
2. Consider the set of simplification rules R_1, R_2, R'_3, R_f. Show that this set of rules is complete if we consider symmetric encryption/decryption and pairing/projection (no asymmetric encryption).

Ackowledgements

Bruno Blanchet, Steve Kremer and David Pointcheval contributed to these notes and we warmly thank them for their help.

References

[1] Spore, the security protocol open repository. `//www.lsv.ens-cachan.fr/spore`.

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

[3] Martín Abadi and Cédric Fournet. Mobile values, new names, and secure communication. In *Proc. 28th Symposium on Principles of Programming Languages (POPL'01)*, pages 104–115. ACM Press, 2001.

[4] Martín Abadi and Cédric Fournet. Private authentication. *Theoretical Computer Science*, 322(3):427–476, 2004.

[5] Martín Abadi, Bruno Blanchet, and Cédric Fournet. Just fast keying in the pi calculus. *ACM Transactions on Information and System Security (TISSEC)*, 10(3):1–59, 2007.

[6] Alessandro Armando, David A. Basin, Yohan Boichut, Yannick Chevalier, Luca Compagna, Jorge Cuéllar, Paul Hankes Drielsma, Pierre-Cyrille Héam, Olga Kouchnarenko, Jacopo Mantovani, Sebastian Mödersheim, David von Oheimb, Michaël Rusinowitch, Judson Santiago, Mathieu Turuani, Luca Viganò, and Laurent Vigneron. The avispa tool for the automated validation of internet security protocols and applications. In *Computer Aided Verification*, volume 3576 of *Lecture Notes in Computer Science*. Springer, 2005.

[7] Mathieu Baudet. Deciding security of protocols against off-line guessing attacks. In *Proc. 12th Conference on Computer and Communications Security*, pages 16–25. ACM Press, 2005.

[8] B. Blanchet, H. Comon-Lundh, S. Delaune, C. Fournet, S. Kremer, and D. Pointcheval. Cryptographic protocols: formal and computational proofs of security. Lecture Notes of the Parisian Master of Research in Computer Science (MPRI), 2011. Available on the MPRI web pages.

[9] Bruno Blanchet. An efficient cryptographic protocol verifier based on Prolog rules. In *Computer Security Foundations Workshop (CSFW'01)*, 2001.

[10] J. Clark and J. Jacob. A survey of authentication protocol literature. `http://www.cs.york.ac.uk/~jac/papers/drareviewps.ps`, 1997.

[11] H. Comon-Lundh and V. Shmatikov. Intruder deductions, constraint solving and insecurity decision in preence of exclusive or. In P. Kolaitis, editor, *Eighteenth Annual IEEE Symposium on Logic in Computer Science*, Ottawa, Canada, June 2003. IEEE Computer Society.

[12] Hubert Comon-Lundh, Véronique Cortier, and Eugen Zlinescu. Deciding security properties of cryptographic protocols. application to key cycles. *Transaction on Computational Logic*, 11(2), 2010.

[13] Ricardo Corin, Jeroen Doumen, and Sandro Etalle. Analysing password protocol security against off-line dictionary attacks. *ENTCS*, 121:47–63, 2005.

[14] V. Cortier and S. Kremer, editors. *Formal Models and Techniques for Analyzing Security Protocols*. IOS Press, 2011.

[15] Véronique Cortier, Stéphanie Delaune, and Pascal Lafourcade. A survey of algebraic properties used in cryptographic protocols. *Journal of Computer Security*, 14(1):1–43, 2006.

[16] Stéphanie Delaune, Steve Kremer, and Mark D. Ryan. Verifying privacy-type properties of electronic voting protocols. *Journal of Computer Security*, 17(4):435–487, July 2009.

[17] Cédric Fournet and Martín Abadi. Hiding names: Private authentication in the applied pi calculus. In *Proc. International Symposium on Software Security (ISSS'02)*, volume 2609 of *Lecture Notes in Computer Science*, pages 317–338. Springer, 2003.

[18] Gavin Lowe. An attack on the needham-schroeder public-key authentication protocol. *Information Processing Letters*, 56(3):131–133, 1996.

[19] Gavin Lowe. A hierarchy of authentication specification. In *10th Computer Security Foundations Workshop (CSFW '97), June 10-12, 1997, Rockport, Massachusetts, USA*, pages 31–44. IEEE Computer Society, 1997.

[20] David McAllester. Automatic recognition of tractability in inference relations. *Journal of the ACM*, 40(2), 1993.

[21] J. Millen and V. Shmatikov. Constraint solving for bounded-process cryptographic protocol analysis. In *Proc. 8th ACM Conference on Computer and Communications Security*, 2001.

[22] Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes, i & ii. *Inf. Comput.*, 100(1):1–77, 1992.

[23] R. Needham and M. Schroeder. Using encryption for authentification in large networks of computers. *Communications of the ACM*, 21(12):993–999, 1978.

[24] Michael Rusinowitch and Mathieu Turuani. Protocol insecurity with finite number of sessions is np-complete. In *Proc.14th IEEE Computer Security Foundations Workshop*, Cape Breton, Nova Scotia, June 2001.

[25] M. Ryan and B. Smyth. Applied pi calculus. In V. Cortier and S. Kremer, editors, *Formal Models and Techniques for Analyzing Security Protocols*. IOS Press, To appear.

[26] Bogdan Warinschi. A computational analysis of the needham-schroeder protocol. In *16th Computer security foundation workshop (CSFW)*, pages 248–262. IEEE, 2003.