MPRI Lecture Notes

Course 2-30

Cryptographic protocols
Formal and Computational Proofs

Authors :

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

This is a preliminary version, for MPRI students only. Please post your comments/corrections.

Contents

Part I

Modelling Protocols and Security Properties

Chapter 1

An Introductory Example

We start with the well-known example of the so-called "Needham-Schroeder public-key protocol" [?], that has been designed in 1978 and for which an attack was found in 1996 by G. Lowe [?], 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 1.1.

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

Figure 1.1: Informal description of the Needham-Schroeder public key protocol

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

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

Figure 1.2: Attack on the Needham-Schroeder public key protocol

Attack. Actually, an attack was found in 1996 by G. Lowe [**?**] on the Needham-Schroeder public-key protocol. The attack described in Figure 1.2 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 1.3).

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

Figure 1.3: Description of the Needham-Schroeder-Lowe protocol

The attack described above cannot be mounted in the corrected version of the protocol. Actually, it is reported in [**?**] that the technique that permitted to find the Lowe attack on the Needham-Schroeder public key protocol found no attack on this 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 1.1 *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 new $N.P$ restricts the name N in P and can for instance be used to model that N is a fresh random number. in$(c, x).P$ models the input of a term on a channel c, which is then substituted for x in process P. out(c, t) outputs a term t on a channel c. The conditional if $M = N$ then P 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:

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

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

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

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

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

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

1.3 An Attack on the Fixed Version of the Protocol

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

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

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

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

Attack. Assume now that we are using such an encryption scheme in the Needham-Schroeder-Lowe protocol and that pairing two group elements $m_1 = g^{m_1'}$ and $m_2 = g^{m_2'}$ is performed in a naive way: $\langle m_1, m_2 \rangle$ is mapped to $g^{m_1' + 2^{|m_1'|} \times m_2'}$ (*i.e.* concatenating the binary representations of the messages m_1' and m_2'). In such a case, an attack can be mounted on the protocol (see Figure 1.4).

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 [?] there is a real attack that is only based on weaknesses of the El Gamal encryption scheme. In particular, the attack does not dependent on how pairing is implemented.

1. $a \to d: \ \{\langle a, N_a\rangle\}_{\mathsf{pk}(d)}$

    1′. $d(a) \to b: \ \{\langle a, N_a\rangle\}_{\mathsf{pk}(b)}$

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

d intercepts this message, and computes

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

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

    3′. $d \to b: \ \{N_b\}_{\mathsf{pk}(d)}$

Figure 1.4: Attack on the Needham-Schroeder-Lowe protocol with El-Gamal encryption.

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

1.4 Further Readings

A survey by Clark and Jacob [**?**] 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 [**?**] 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 [**?**], 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.

1.5 Exercises

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

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

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

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

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

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

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

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

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

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

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

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

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

1. Check that the 'man-in-the-middle' attack described in Figure 1.2 does not exist.

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

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

Chapter 2

A Small Process Calculus

We now define our cryptographic process calculus for describing protocols. This calculus is inspired by the applied pi calculus [?] which is the calculus used by the ProVerif tool [?]. The applied pi calculus is a language for describing concurrent processes and their interactions. It is an extension of the pi calculus [?] with cryptographic primitives. It is designed for describing and analysing a variety of security protocols, such as authentication protocols (*e.g.* [?]), key establishment protocols (*e.g.* [?]), e-voting protocols (*e.g.* [?]), ... 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 [?]. 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. $fn(M)$) the set of variables (resp. names) that occur in M. A term M that does not contain any variable is a *ground term*. The set of positions of a term T is written $pos(T) \subseteq \mathbb{N}^*$, and its set of subterms $st(T)$. The subterm of T at position $p \in pos(T)$ is written $T|_p$. The term obtained by replacing $T|_p$ with a term U in T is denoted $T[U]_p$.

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

By the means of a convergent term rewriting system $\mathcal{R}$, we describe the equations which hold on terms built from the signature. A *term rewriting system* (TRS) is a set of *rewrite rules* $l \rightarrow r$ 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 $S \rightarrow_{\mathcal{R}} T$, if there is $l \rightarrow r$ 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 $\rightarrow^*_{\mathcal{R}}$ the reflexive and transitive closure of $\rightarrow_{\mathcal{R}}$, and by $=_{\mathcal{R}}$ the symmetric, reflexive and transitive closure of $\rightarrow_{\mathcal{R}}$. A TRS $\mathcal{R}$ is *convergent* if it is:

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

A term T is *$\mathcal{R}$-reduced* if there is no term S such that $T \rightarrow_{\mathcal{R}} S$. If $T \rightarrow^*_{\mathcal{R}} S$ 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 2.1 *In order to model the handshake protocol that we will present later on, we introduce the signature:*

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

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

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

Example 2.2 *In order to model the Needham-Schroeder protocol, we will consider the following signature:*

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

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

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

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 [?] 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 $fn(N) \cap \overline{n} = \emptyset$ 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 2.3 *Consider the theory $\mathcal{R}_{\mathsf{senc}}$ given in Example 2.1 and the following frame:*

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

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

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

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

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

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

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 $\phi =_\alpha \nu\overline{n}.\sigma$, $\overline{n} \cap (fn(M) \cup fn(N)) = \emptyset$, and $M\sigma\!\downarrow$ and $N\sigma\!\downarrow$ are both constructor terms that are equal,* i.e. $M\sigma\!\downarrow = N\sigma\!\downarrow$.

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

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

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

Example 2.4 *Consider the rewriting system $\mathcal{R}_{\mathsf{senc}}$ provided in Example 2.1. Consider the frames $\phi = \mathsf{new}\ k.\{^{\mathsf{senc}(s_0,k)}/_{x_1}, {}^{k}/_{x_2}\}$, and $\phi' = \mathsf{new}\ k.\{^{\mathsf{senc}(s_1,k)}/_{x_1}, {}^{k}/_{x_2}\}$. Intuitively, s_0 and s_1 could be the two possible (public) values of a vote. We have $(\mathsf{sdec}(x_1, x_2) =_{\mathcal{R}_{\mathsf{senc}}} s_0)\phi$ whereas $(\mathsf{sdec}(x_1, x_2) \neq_{\mathcal{R}_{\mathsf{senc}}} s_0)\phi'$. Therefore we have that $\phi \not\sim \phi'$. However, we have that:*

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

Example 2.5 *Consider again the rewriting system $\mathcal{R}_{\mathsf{senc}}$ provided in Example 2.1. We have that:*

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

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

$P, Q, R \mathrel{\hat{=}}$ 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 $\mathsf{0}$ 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 $\mathsf{0}$. 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 $\mathsf{0}$.

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

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

We rely on the signature given in Example 2.1. The goal of this protocol is to authenticate B from A's point of view, provided that they share an initial secret w. This is done by a simple challenge-response transaction: A sends a random number (a nonce*) encrypted with the shared secret key w. Then, B decrypts this message, applies a given function (for instance* $\mathsf{f}(n) = n+1$*) to it, and sends the result back, also encrypted with w. Finally, the agent A checks the validity of the result by decrypting the message and checking the decryption against* $\mathsf{f}(n)$*. In our calculus, we can model the protocol as* $\mathsf{new}\ w.(P_A \parallel P_B)$ *where*

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

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

Example 2.7 *Going back to the Needham-Schroeder public key protocol described in Chapter 1 and considering the signature given in Example 2.2, we have that:*

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

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

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

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

Example 2.8 (Scenario 1) *The following specifies a copy of the role of Alice, played by a, with d and a copy of the role of Bob, played by b, with a, as well as the fact that d is dishonest, hence his secret key is leaked.*

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

Example 2.9 (Scenario 2) *Assume that we wish a to execute the role of the initiator, however with any other party, which is specified here by letting the environment give the identity of such another party: the process first receives x_b, that might be bound to any value. The other role is specified in the same way.*

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

Example 2.10 (Scenario 3) *In Example 2.8 and Example 2.9, a was only able to engage the protocol once (and b was only able to engage once in a response). We may wish a (resp. b) be able to execute any number of instances of the role of the initiator (resp. responder).*

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

Example 2.11 (Scenario 4) *Finally, in general, the role of the initiator could be executed by any agent, including b and the role of the responder could be executed by any number of agents as well. We specify an unbounded number of parties, engaging in an unbounded number of sessions by:*

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

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

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

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

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

Extended processes built up from the null process, active substitutions using parallel composition and restriction are called *frames* (extending the notion of frame introduced in Section 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 2.12 *Consider the following process:*

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

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

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:

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

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

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

In particular, any frame can be rewritten as $\mathsf{new}\ \overline{n}.\sigma$ matching the notion of frame introduced in Section 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:

$$\begin{array}{ll}
\textsc{Repl} & !P \xrightarrow{\tau} P' \parallel !P \quad \text{where } P' \text{ is a fresh renaming of } P \\[1ex]
\textsc{Then} & \mathsf{if}\ M = N\ \mathsf{then}\ P\ \mathsf{else}\ Q \xrightarrow{\tau} P \text{ where } M{\downarrow} = N{\downarrow} \text{ and } M{\downarrow} \text{ is a message} \\[1ex]
\textsc{Else} & \mathsf{if}\ M = N\ \mathsf{then}\ P\ \mathsf{else}\ Q \xrightarrow{\tau} Q \text{ where } M{\downarrow} \neq N{\downarrow} \text{ and } M{\downarrow}, N{\downarrow} \text{ are messages} \\[1ex]
\textsc{Comm} & \mathsf{out}(c, M_1).P_1 \parallel \mathsf{in}(c, M_2).P_2 \xrightarrow{\tau} P_1 \parallel P_2\theta \text{ where } \theta \text{ is such that} \\
& \qquad\qquad\qquad \mathsf{Dom}(\theta) = fv(M_2),\ M_2\theta{\downarrow} = M_1{\downarrow}, \text{ and } M_1{\downarrow} \text{ is a message.}
\end{array}$$

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

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

$$\begin{array}{lll}
\textsc{In} & \mathsf{in}(c, x).P \xrightarrow{\mathsf{in}(c,M)}_{\ell} P\{^M/_x\} & \text{where } M \text{ is a message} \\[1ex]
\textsc{Out} & \mathsf{out}(c, M).P \xrightarrow{\mathsf{out}(c,M\downarrow)}_{\ell} P \parallel \{^{M\downarrow}/_x\} & \text{where } x \text{ is a fresh variable and } M{\downarrow} \text{ is a message}
\end{array}$$

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

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

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

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

Example 2.13 *Going back to the handshake protocol described in Example 2.6, the derivation described below represents a normal execution of the protocol. For simplicity of this example we suppose that $x \notin fv(P)$.*

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

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

Example 2.14 *Continuing Example 2.7 we develop some transitions from*

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

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

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

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

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

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

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

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

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

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

2.3.1 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 2.15 (Scenario 1) *Consider again the following process defined in Example 2.7 :*

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

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 2.16 (Scenario 2) *Consider now*

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

*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.

2.3.2 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 [**?**].

> 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.

2.3.3 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 [**?**], inspired from the one of [**?**]. 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 $\overline{w} \subseteq bn(A)$. 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 2.17 *Consider the handshake protocol described in Example 2.6. 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 $\mathsf{f}(\mathsf{sdec}(x_1, x)) \overset{?}{=} \mathsf{sdec}(x_2, x)$ 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.

2.3.4 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 computations, 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 $A \xrightarrow{\mathsf{in}(c,n')} 0$ 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:

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

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

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

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

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 2.18 *Going back to the handshake protocol described in Example 2.6, the derivation described below represents a normal execution of the protocol in the new labelled semantics.*

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

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

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 2.19 *Consider the equational theory described in Example 2.1. We have that:*

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

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

$$\begin{array}{rcl}
\mathsf{out}(c,a).\mathsf{out}(c,a) & \approx_t & \mathsf{out}(c,a) \parallel \mathsf{out}(c,a) \\
\mathsf{out}(c_1,a).\mathsf{out}(c_2,a) & \not\approx_t & \mathsf{out}(c_1,a) \parallel \mathsf{out}(c_2,a) \\
\mathsf{out}(c,a);\mathsf{out}(c,b) & \not\approx_t & \mathsf{out}(c,a) \parallel \mathsf{out}(c,b)
\end{array}$$

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

Example 2.20 *We consider a slightly simplified version of a protocol given in [?] 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:*

$$\begin{array}{rcl}
A \to B & : & \mathsf{aenc}(\langle N_a, \mathsf{pk}(A)\rangle, \mathsf{pk}(B)) \\
B \to A & : & \mathsf{aenc}(\langle N_a, \langle N_b, \mathsf{pk}(B)\rangle\rangle, \mathsf{pk}(A))
\end{array}$$

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: $\mathsf{aenc}(N_b, \mathsf{pk}(B))$. 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 $M = \mathsf{dec}(x, \mathsf{sk}(a))$.

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

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

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

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* else 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 $K(a,a',b) = \mathsf{out}(c, \mathsf{pk}(a)).\mathsf{out}(c, \mathsf{pk}(a')).\mathsf{out}(c, \mathsf{pk}(b))$.

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 [?]. 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 $B \to^* B'$ and $A'\ \mathcal{R}\ B'$ for some B',*
3. *if $A \xrightarrow{\alpha}_\ell A'$ then $B \to^* \xrightarrow{\alpha}_\ell \to^* B'$ 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 2.21).

Lemma 2.1 *Let A and B be two extended processes: $A \approx_\ell B$ implies $A \approx_t B$.*

Example 2.21 *For convenience we introduce for this example a non-deterministic choice operator $+$ and extend the internal reduction by the rule $A + B \to A$ 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.

2.4 Further Readings

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

2.5 Exercises

Exercice 5 ($\star$)
Consider the signature described in Example 2.1. 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:

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

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.

Exercice 7 ($\star$)
Consider the formalisation of the Needham-Schroeder protocol as described in Example 2.7, and the following scenario (see Example 2.15).

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

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 2.6 and that does not exist in this new formalisation.

Part II

Verification in the Symbolic Setting

Chapter 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.* [**?**]. 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 [**?**], many results (*e.g.* [**?**, **?**]) 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 3.1 *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.*

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

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

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 $v_1, \dots, v_n$, there is an instance of an inference rule with conclusion v_0 and hypotheses $v_1, \dots, v_n$. 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 3.2 *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:*

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

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 [**?**].

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* $\mathsf{Steps}(\Pi) \subseteq st(T \cup \{u\})$.

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 3.1.

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 3.1 (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 $st(T \cup \{u\})$,* i.e. *$\mathsf{Steps}(\Pi) \subseteq st(T \cup \{u\})$. Moreover, if Π is reduced to a leaf or ends with a decomposition rule, then we have that $\mathsf{Steps}(\Pi) \subseteq st(T)$.*

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:

$$\Pi = \left\{ \begin{array}{c} \Pi_1 \qquad\qquad \Pi_n \\ \dfrac{u_1 \quad \cdots \quad u_n}{u}\,\mathsf{R} \end{array} \right.$$

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 $\mathsf{Steps}(\Pi_1) \subseteq st(T)$, and thus $u_1 \in st(T)$. Moreover, we have that $u \in st(u_1)$, and thus $u \in st(T)$. This allows us to conclude that $\mathsf{Steps}(\Pi) \subseteq st(T)$.

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 2.14. 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 3.3 *A* Deducibility constraint system *is either* $\perp$ *or a conjunction of deducibility constraints of the form:*

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

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* $fv(T_i) \subseteq fv(u_1, \ldots, u_{i-1})$

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 3.4 (T_x) *Let* $\mathcal{C} = T_1 \overset{?}{\vdash} u_1 \ \wedge \ \ldots \ \wedge \ T_n \overset{?}{\vdash} u_n$ *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* $T \overset{?}{\vdash} u \in \mathcal{C}$ *and* $x \in fv(u)$*.*

Thanks to the monotonicity and the origination properties, for any $x \in fv(\mathcal{C})$, 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 3.5 (solution) *Let* $\mathcal{I}$ *be an inference system. A substitution* σ *is a* solution *of a deducibility constraint system* $\mathcal{C} = T_1 \overset{?}{\vdash} u_1 \wedge \ldots \wedge T_n \overset{?}{\vdash} u_n$ *if there exists a proof of* $T_i\sigma \vdash u_i\sigma$ *in* $\mathcal{I}$ *for every* $i \in \{1, \ldots, n\}$*.*

Example 3.3 *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:*

$$\mathcal{C} \mathrel{\hat{=}} \left\{ \begin{array}{rcl} a,\ b,\ d,\ \mathsf{sk}(d),\ \mathsf{aenc}(\langle a, N_a\rangle, d) & \overset{?}{\vdash} & \mathsf{aenc}(\langle a, x\rangle, b) \\ a,\ b,\ d,\ \mathsf{sk}(d),\ \mathsf{aenc}(\langle a, N_a\rangle, d),\ \mathsf{aenc}(\langle x, N_b\rangle, a) & \overset{?}{\vdash} & \mathsf{aenc}(\langle N_a, y\rangle, a) \end{array} \right.$$

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 3.1). Many parts of this section, including the set of simplification rules, are borrowed from [**?**].

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.

$$
\begin{array}{llll}
\mathsf{R}_1 & \mathcal{C} \wedge T \overset{?}{\vdash} u \rightsquigarrow \mathcal{C} & & \text{if } T \cup \{x \mid (T' \overset{?}{\vdash} x) \in \mathcal{C}, T' \subsetneq T\} \vdash u \\
\mathsf{R}_2 & C \wedge T \overset{?}{\vdash} u \rightsquigarrow_\sigma \mathcal{C}\sigma \wedge T\sigma \overset{?}{\vdash} u\sigma & & \text{if } t \in st(T), \sigma = \mathsf{mgu}(t,u), t \neq u \\
 & & & \qquad\qquad t, u \text{ not variables} \\
\mathsf{R}_3 & \mathcal{C} \wedge T \overset{?}{\vdash} u \rightsquigarrow_\sigma \mathcal{C}\sigma \wedge T\sigma \overset{?}{\vdash} u\sigma & & \text{if } t_1, t_2 \in st(T), \sigma = \mathsf{mgu}(t_1,t_2), \text{ and } t_1 \neq t_2 \\
\mathsf{R}_4 & \mathcal{C} \wedge T \overset{?}{\vdash} u \rightsquigarrow \bot & & \text{if } fv(T \cup \{u\}) = \emptyset \text{ and } T \not\vdash u \\
\mathsf{R}_\mathsf{f} & \mathcal{C} \wedge T \overset{?}{\vdash} \mathsf{f}(u,v) \rightsquigarrow \mathcal{C} \wedge T \overset{?}{\vdash} u \wedge T \overset{?}{\vdash} v & & \text{for } \mathsf{f} \in \{\langle\ ,\ \rangle, \mathsf{senc}, \mathsf{aenc}\}
\end{array}
$$

Figure 3.1: 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 $fv(v\sigma, u\sigma) \subseteq fv(v,u)$.

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 3.2 *The simplification rules transform a deducibility constraint system into a deducibility constraint system.*

Theorem 3.1 *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 3.3 *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 3.1, 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:

$$\Pi_0 = \left\{ \begin{array}{c} \Pi_1 \qquad \ldots \qquad \Pi_n \\ \dfrac{u_1 \qquad\qquad u_n}{u}\ \mathsf{R} \end{array} \right.$$

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

$$\Pi = \left\{ \begin{array}{c} \Pi'_1 \qquad \ldots \qquad \Pi'_n \\ \dfrac{u_1 \qquad\qquad u_n}{u}\ \mathsf{R} \end{array} \right.$$

The proof Π is a simple proof of u. □

Lemma 3.4 *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 $t \in st(T_i) \smallsetminus \mathcal{X}$ 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 $t \in st(T_j)$) 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 $t \in st(T_j) \smallsetminus \mathcal{X}$ 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 3.5 *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:

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

  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 $fv(T_i)$ since $u \in st(T_i)$. 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 $T_j\sigma \vdash w$:

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

  By simplicity, the last rule of the proof Π_1 is a decomposition or an axiom. By Lemma 3.4, there is $t \in st(T_j) \smallsetminus \mathcal{X}$ 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 $t_2 \in fv(T_i)$, 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 $t \in st(T_j) \smallsetminus \mathcal{X}$ 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 3.4 (T_j satisfies the hypotheses of the lemma since $T_j \subseteq T_i$), there is a non-variable subterm $w_1 \in st(T_j)$ 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 θ is a solution of $\mathcal{C}$, then there is a deducibility constraint system $\mathcal{C}'$, a substitution σ, and a solution θ' of $\mathcal{C}'$ such that $\mathcal{C} \rightsquigarrow_\sigma \mathcal{C}'$ and $\theta = \sigma\theta'$.*

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

Consider a minimal unsolved constraint $T_i \overset{?}{\vdash} u_i$ such that u_i is not a variable. We have that $T_i\theta \vdash u_i\theta$. Consider a simple proof Π of $T_i\theta \vdash u_i\theta$. 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 3.4 we obtain that there is $t \in st(T_i) \smallsetminus \mathcal{X}$ 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 $u_i \in st(T_i)$ and we already know that u_i is not a variable. We consider two cases:

      i. There are two distinct terms $t_1, t_2 \in st(T)$ 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 3.5). □

3.3.3 Complexity

The termination stated in Theorem 3.1 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 [**?**]. 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 [**?**]. The NP-hardness can be established with a reduction from 3-SAT.

3.4 Further Readings

Many parts of this section are borrowed from [**?**]. 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 [**?**]. Many results (*e.g.* [**?**, **?**]) 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
Say whether each couple of terms are unifiable or not. If so, give a most general unifier (mgu).

1. $\langle x, b\rangle$ and $\langle a, y\rangle$,
2. $\mathsf{aenc}(x, a)$ and $\mathsf{aenc}(b, x)$,
3. $\langle x, y\rangle$ and $\langle\langle y, y\rangle, a\rangle$,
4. z and $\langle x, y\rangle$.

Exercice 10 ($\star$)
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 11 ($\star\star\star$)
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 12 (⋆)
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 13 (⋆⋆)
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 14 (⋆)
We consider the signature and the inference system given in Example 3.1. Let $T_0 = \{a,\ b,\ c,\ \mathsf{sk}(c),\ \mathsf{aenc}(\langle a, \mathsf{aenc}$

and $\mathcal{C} = \{T_0 \overset{?}{\vdash} \mathsf{aenc}(\langle a, \mathsf{aenc}(x_1, b)\rangle, b)\}$. What are the solutions of $\mathcal{C}$?

Exercice 15 (⋆⋆)
Consider the following protocol (defined informally):

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

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 16 (⋆)
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 17 (⋆⋆)
We consider the following variant of the rule R_3

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

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).

Chapter 4

Unbounded process verification

In this chapter, we first show that the problem of the verification of security protocols with an unbounded number of sessions is undecidable. Next, we present a technique for solving this problem, based on an abstract representation of the protocol by Horn clauses and implemented in the tool ProVerif. Obviously, because of the undecidability, this technique does not always solve the problem. (It may answer “I don’t know” or not terminate.)

4.1 Undecidability

First observe that the existence of an attack is undecidable:

Theorem 4.1 *Given a process P (in our algebra) and a term t, the question of whether there are σ, Q such that $P \xrightarrow{*} \nu\overline{n}(\sigma \| Q)$ and $\nu\overline{n}.\sigma \vdash t$ is undecidable.*

This is even true when the attacker is passive, when the keys are atomic and/or the messages are of bounded sizes ([**?**] for instance).

To give an idea of the proof, we encode a (modified) PCP:

The following problem is undecidable:

Input: two finite sequences $v_0, v_1, \ldots, v_n, w_0, w_1, \ldots, w_n \in \{a, b\}^*$

Question: are there a k and a sequence of indices $i_1, \ldots, i_k \in [1..n]$ such that $v_0.v_{i_1} \cdots .v_{i_k} = w_0.w_{i_1} \cdots .w_{i_k}$?

For any word $u \in \{a, b\}^*$, we define inductively $\widetilde{u}$ as a function from terms to terms as follows:

- if u is the empty word, then $\widetilde{u}$ is the identity
- $\widetilde{a \cdot u}(t) = \langle a, \widetilde{u}(t) \rangle$.

$$P_A = \mathsf{out}(c, \{\langle \widetilde{v_0}(0), \widetilde{w_0}(0) \rangle\}^s_{k_{AB}}).\mathsf{in}(c, \{\langle x, x \rangle\}^s_{k_{AB}}).\mathsf{out}(c, k_{AB})$$

$$P_B = \mathsf{in}(c, \{\langle x, y \rangle\}^s_{k_{AB}}).\mathsf{out}(c, \{\langle \widetilde{v_1}(x), \widetilde{w_1}(x) \rangle\}^s_{k_{AB}} ... \mathsf{out}(c, \{\langle \widetilde{v_n}(x), \widetilde{w_n}(x) \rangle\}^s_{k_{AB}} ...$$

Let $P = \nu k_{AB}.(P_A \| !P_B)$. We can deduce k_{AB} iff the modified PCP has a solution. Note that the attacker has not much to do: only select one of the outputs of P_B and send it to the next instance of P_B.

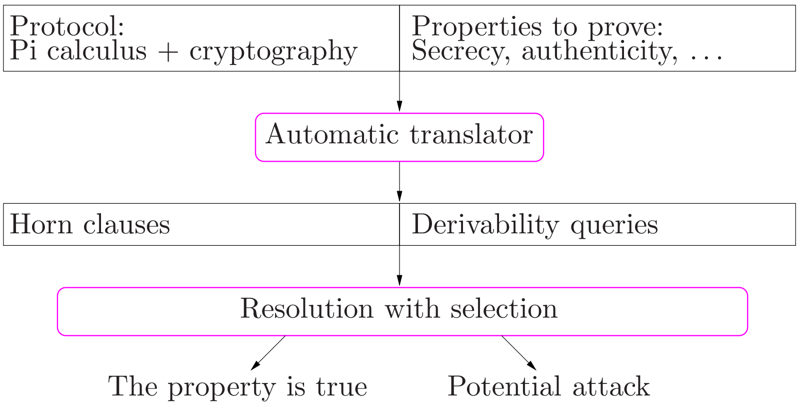


Figure 4.1: Structure of ProVerif

4.2 Structure and Main Features of ProVerif

The structure of ProVerif is represented in Figure 4.1. ProVerif takes as input a model of the protocol in an extension of the pi calculus with cryptography, similar to the calculus of Chapter 2. It supports a wide variety of cryptographic primitives, modeled by rewrite rules or by equations. ProVerif also takes as input the security properties that we want to prove. It can verify various security properties, including secrecy, authentication (correspondences), and some observational equivalence properties. It automatically translates this information into an internal representation by Horn clauses: the protocol is translated into a set of Horn clauses, the security properties to prove are translated into derivability queries on these clauses. ProVerif uses an algorithm based on resolution with free selection to determine whether a fact is derivable from the clauses. If the fact is *not* derivable, then the desired security property is proved. If the fact is derivable, then there may be an attack against the considered property: the derivation may correspond to an attack, but it may also correspond to a "false attack", because the Horn clause representation makes some abstractions. These abstractions are key to the key of an unbounded number of sessions of protocols.

Section 4.3 presents the model of protocols. Section 4.4 presents the Horn clause representation of protocols and the resolution algorithm. Section 4.5 gives the translation from the pi calculus model to Horn clauses for secrecy properties, with extensions to correspondences and equivalences in Sections 4.6 and 4.7 respectively.

4.3 A Formal Model of Security Protocols

This section details the model of protocols used by ProVerif. This calculus was presented in [**?**]; we adapt that presentation.

4.3.1 Syntax and Informal Semantics

Figure 4.2 gives the syntax of terms (data) and processes (programs) of ProVerif's input language. The identifiers a, b, c, k, and similar ones range over names, and x, y, and z range over variables. The syntax also assumes a set of symbols for constructors and destructors; we often use f for a constructor and g for a destructor.

Constructors are used to build terms. Therefore, the terms are variables, names, and constructor applications of the form $f(M_1, \dots, M_n)$; the terms are untyped. On the other hand, destructors do not appear in terms, but only manipulate terms in processes. They are partial functions on terms that processes can apply. The process $\mathsf{let}\ x = g(M_1, \dots, M_n)\ \mathsf{in}\ P\ \mathsf{else}\ Q$

$M, N ::=$	terms
x, y, z	variable
a, b, c, k, s	name
$f(M_1, \ldots, M_n)$	constructor application
$P, Q ::=$	processes
$\mathsf{out}(M, N).P$	output
$\mathsf{in}(M, x).P$	input
0	nil
$P \parallel Q$	parallel composition
$!P$	replication
$\mathsf{new}\ a.P$	restriction
$\mathsf{let}\ x = g(M_1, \ldots, M_n)\ \mathsf{in}\ P\ \mathsf{else}\ Q$	destructor application
$\mathsf{let}\ x = M\ \mathsf{in}\ P$	local definition
$\mathsf{if}\ M = N\ \mathsf{then}\ P\ \mathsf{else}\ Q$	conditional

Figure 4.2: Syntax of the process calculus

tries to evaluate $g(M_1, \ldots, M_n)$; if this succeeds, then x is bound to the result and P is executed, else Q is executed. More precisely, the semantics of a destructor g of arity n is given by a set $\mathrm{def}(g)$ of rewrite rules of the form $g(M_1, \ldots, M_n) \rightarrow M$ where $M_1, \ldots, M_n, M$ are terms without names, and the variables of M also occur in $M_1, \ldots, M_n$. We extend these rules by $g(M'_1, \ldots, M'_n) \rightarrow M'$ if and only if there exist a substitution σ and a rewrite rule $g(M_1, \ldots, M_n) \rightarrow M$ in $\mathrm{def}(g)$ such that $M'_i = \sigma M_i$ for all $i \in \{1, \ldots, n\}$, and $M' = \sigma M$. We assume that the set $\mathrm{def}(g)$ is finite. (It usually contains one or two rules in examples.)

Using these constructors and destructors, we can represent data structures, such as tuples, and cryptographic operations, for instance as follows:

- $n\mathsf{tuple}(M_1, \ldots, M_n)$ is the tuple of the terms $M_1, \ldots, M_n$, where $n\mathsf{tuple}$ is a constructor. (We sometimes abbreviate $n\mathsf{tuple}(M_1, \ldots, M_n)$ to $(M_1, \ldots, M_n)$.) The n projections are destructors $i\mathsf{th}_n$ for $i \in \{1, \ldots, n\}$, defined by

$$i\mathsf{th}_n(n\mathsf{tuple}(x_1, \ldots, x_n)) \rightarrow x_i$$

- $\mathsf{senc}(M, N)$ is the symmetric (shared-key) encryption of the message M under the key N, where senc is a constructor. The corresponding destructor sdec is defined by

$$\mathsf{sdec}(\mathsf{senc}(x, y), y) \rightarrow x$$

  Thus, $\mathsf{sdec}(M', N)$ returns the decryption of M' if M' is a message encrypted under N.

- In order to represent asymmetric (public-key) encryption, we may use two constructors pk and aenc: $\mathsf{pk}(M)$ builds a public key from a secret M and $\mathsf{aenc}(M, N)$ encrypts M under N. The corresponding destructor adec is defined by

$$\mathsf{adec}(\mathsf{aenc}(x, \mathsf{pk}(y)), y) \rightarrow x$$

- As for digital signatures, we may use a constructor sign, and write $\mathsf{sign}(M, N)$ for M signed with the signature key N, and the two destructors check and $\mathsf{getmess}$ with the rewrite rules:

$$\mathsf{check}(\mathsf{sign}(x, y), \mathsf{pk}(y)) \rightarrow x$$
$$\mathsf{getmess}(\mathsf{sign}(x, y)) \rightarrow x$$

- We may represent a one-way hash function by the constructor h. There is no corresponding destructor; so we model that the term M cannot be retrieved from its hash $h(M)$.

Thus, the process calculus supports many of the operations common in security protocols. It has limitations, though: for example, modular exponentiation or XOR cannot be directly represented by a constructor or by a destructor. We explain how we can treat some of these primitives in Section 4.5.3.

Exercice 18
Model signatures that do not reveal the signed message.

The other constructs in the syntax of Figure 4.2 are standard; most of them come from the pi calculus, and were also present in the calculus of Chapter 2.

- The input process $\mathsf{in}(M, x).P$ inputs a message on channel M, and executes P with x bound to the input message. The output process $\mathsf{out}(M, N).P$ outputs the message N on the channel M and then executes P. Here, we use an arbitrary term M to represent a channel: M can be a name, a variable, or a constructor application. The calculus is monadic (in that the messages are terms rather than tuples of terms), but a polyadic calculus can be simulated since tuples are terms. It is also synchronous (in that a process P is executed after the output of a message). As usual, we may omit P when it is 0.

- The nil process 0 does nothing.

- The process $P \parallel Q$ is the parallel composition of P and Q.

- The replication $!P$ represents an unbounded number of copies of P in parallel.

- The restriction $\mathsf{new}\ a.P$ creates a new name a, and then executes P.

- The local definition $\mathsf{let}\ x = M\ \mathsf{in}\ P$ executes P with x bound to the term M.

- The conditional $\mathsf{if}\ M = N\ \mathsf{then}\ P\ \mathsf{else}\ Q$ executes P if M and N reduce to the same term at runtime; otherwise, it executes Q. As usual, we may omit an else clause when it consists of 0.

The name a is bound in the process $\mathsf{new}\ a.P$. The variable x is bound in P in the processes $\mathsf{in}(M, x).P$, $\mathsf{let}\ x = g(M_1, \ldots, M_n)\ \mathsf{in}\ P\ \mathsf{else}\ Q$, and $\mathsf{let}\ x = M\ \mathsf{in}\ P$. We write $fn(P)$ and $fv(P)$ for the sets of names and variables free in P, respectively. A process is closed if it has no free variables; it may have free names. We write $\{M_1/x_1, \ldots, M_n/x_n\}$ for the substitution that replaces $x_1, \ldots, x_n$ with $M_1, \ldots, M_n$, respectively. When σ is such a substitution and D is some expression, we may write σD or $D\sigma$ for the result of applying σ to D; the distinction is one of emphasis at most. Except when stated otherwise, substitutions always map variables (not names) to expressions.

ProVerif's calculus resembles the applied pi calculus [?]. Both calculi are extensions of the pi calculus with (fairly arbitrary) functions on terms. However, there are also important differences between these calculi. The first one is that ProVerif uses destructors instead of the equational theories of the applied pi calculus. (Section 4.5.3 contains further material on equational theories.) The second difference is that ProVerif has a built-in error-handling construct (the else clause of the destructor application), whereas in the applied pi calculus the error-handling must be done "by hand".

4.3.2 Example

We use as a running example a simplified version of the Denning-Sacco key distribution protocol [**?**], omitting certificates and timestamps:

$$\begin{array}{ll} \text{Message 1.} & A \to B : \{\{k\}_{sk_A}\}_{pk_B} \\ \text{Message 2.} & B \to A : \{\mathsf{s}\}_k \end{array}$$

This protocol involves two principals A and B. The key sk_A is the secret key of A, pk_A its public key. Similarly, sk_B and pk_B are the secret and public keys of B, respectively. The key k is a fresh key created by A. A sends this key signed with its private key sk_A and encrypted under the public key of B, pk_B. When B receives this message, B decrypts it and assumes, seeing the signature, that the key k has been generated by A. Then B sends a secret s encrypted under k. Only A should be able to decrypt the message and get the secret s. (The second message is not really part of the protocol, we use it to check if the key k can really be used to exchange secrets between A and B. In fact, there is an attack against this protocol [**?**], so s will not remain secret.)

This protocol can be encoded by the following process:

$$\begin{aligned} &P_0 = \mathsf{new}\ sk_A.\mathsf{new}\ sk_B.\mathsf{let}\ pk_A = \mathsf{pk}(sk_A)\ \mathsf{in}\ \mathsf{let}\ pk_B = \mathsf{pk}(sk_B)\ \mathsf{in}\ \mathsf{out}(c, pk_A).\mathsf{out}(c, pk_B). \\ &\qquad (P_A(pk_A, sk_A) \parallel P_B(pk_B, sk_B, pk_A)) \\ &P_A(pk_A, sk_A) = !\ \mathsf{in}(c, x\_pk_B).\mathsf{new}\ k.\mathsf{out}(c, \mathsf{aenc}(\mathsf{sign}(k, sk_A), x\_pk_B)). \\ &\qquad \mathsf{in}(c, x).\mathsf{let}\ z = \mathsf{sdec}(x, k)\ \mathsf{in}\ 0 \\ &P_B(pk_B, sk_B, pk_A) = !\ \mathsf{in}(c, y).\mathsf{let}\ y' = \mathsf{adec}(y, sk_B)\ \mathsf{in} \\ &\qquad \mathsf{let}\ x\_k = \mathsf{check}(y', pk_A)\ \mathsf{in}\ \mathsf{out}(c, \mathsf{senc}(\mathsf{s}, x\_k)) \end{aligned}$$

Such a process can be given as input to ProVerif, in an ASCII syntax. This process first creates the secret keys sk_A and sk_B, computes the corresponding public keys pk_A and pk_B, and sends these keys on the public channel c, so that the adversary has these public keys. Then, it runs the processes P_A and P_B in parallel. These processes correspond respectively to the roles of A and B in the protocol. They both start with a replication, which makes it possible to model an unbounded number of sessions of the protocol.

The process P_A first receives on the public channel c the key x_pk_B, which is the public key of A's interlocutor in the protocol. This message is not strictly speaking part of the protocol; it makes it possible for the adversary to choose with whom A is going to execute a session. In a standard session of the protocol, this key is pk_B, but the adversary can also choose another key, for instance one of his own keys. Then, P_A executes the role of A: it creates a fresh key k, signs it with its secret key sk_A, then encrypts this message under x_pk_B, and sends the obtained message on channel c. P_A then expects the second message of the protocol on channel c, stores it in x and decrypts it. If decryption succeeds, the result (normally the secret s) is stored in z.

The process P_B receives the first message of the protocol on channel c, stores it in y, decrypts it with sk_B, and verifies the signature with pk_A. (The signature is verified with the key pk_A of A and not with an arbitrary key chosen by the adversary since B sends the second message $\{\mathsf{s}\}_k$ only if its interlocutor is the honest participant A.) If these verifications succeed, B believes that x_k is a key shared between A and B, and it sends the secret s encrypted under x_k. If the protocol is correct, s should remain secret.

In the above model, we have assumed for simplicity that A and B each play only one role of the protocol. One could easily write a more general model in which they play both roles, or one could even provide the adversary with an interface that allows it to dynamically create new protocol participants.

Exercice 19
Model the Needham-Schroeder public key protocol of Figure 1.1 in this calculus.

$$
\begin{array}{ll}
P \parallel 0 \equiv P & P \equiv Q \;\Rightarrow\; P \parallel R \equiv Q \parallel R \\
P \parallel Q \equiv Q \parallel P & P \equiv Q \;\Rightarrow\; !P \equiv !Q \\
(P \parallel Q) \parallel R \equiv P \parallel (Q \parallel R) & P \equiv Q \;\Rightarrow\; \mathsf{new}\ a.P \equiv \mathsf{new}\ a.Q \\
\mathsf{new}\ a_1.\mathsf{new}\ a_2.P \equiv \mathsf{new}\ a_2.\mathsf{new}\ a_1.P & P \equiv P \\
\mathsf{new}\ a.(P \parallel Q) \equiv P \parallel \mathsf{new}\ a.Q \quad \text{if } a \notin fn(P) & Q \equiv P \;\Rightarrow\; P \equiv Q \\
 & P \equiv Q,\, Q \equiv R \;\Rightarrow\; P \equiv R
\end{array}
$$

$$
\begin{array}{ll}
\mathsf{out}(N, M).Q \parallel \mathsf{in}(N, x).P \;\rightarrow\; Q \parallel P\{M/x\} & \text{(Red I/O)} \\
\mathsf{let}\ x = g(M_1, \ldots, M_n)\ \mathsf{in}\ P\ \mathsf{else}\ Q \rightarrow P\{M'/x\} & \text{(Red Destr 1)} \\
\quad \text{if } g(M_1, \ldots, M_n) \rightarrow M' & \\
\mathsf{let}\ x = g(M_1, \ldots, M_n)\ \mathsf{in}\ P\ \mathsf{else}\ Q \rightarrow Q & \text{(Red Destr 2)} \\
\quad \text{if there exists no } M' \text{ such that } g(M_1, \ldots, M_n) \rightarrow M' & \\
\mathsf{let}\ x = M\ \mathsf{in}\ P \rightarrow P\{M/x\} & \text{(Red Let)} \\
\mathsf{if}\ M = M\ \mathsf{then}\ P\ \mathsf{else}\ Q \rightarrow P & \text{(Red Cond 1)} \\
\mathsf{if}\ M = N\ \mathsf{then}\ P\ \mathsf{else}\ Q \rightarrow Q \text{ if } M \neq N & \text{(Red Cond 2)} \\
!P \;\rightarrow\; P \parallel !P & \text{(Red Repl)} \\
P \;\rightarrow\; Q \;\Rightarrow\; P \parallel R \;\rightarrow\; Q \parallel R & \text{(Red Par)} \\
P \;\rightarrow\; Q \;\Rightarrow\; \mathsf{new}\ a.P \;\rightarrow\; \mathsf{new}\ a.Q & \text{(Red Res)} \\
P' \equiv P,\, P \;\rightarrow\; Q,\, Q \equiv Q' \;\Rightarrow\; P' \;\rightarrow\; Q' & \text{(Red } \equiv\text{)}
\end{array}
$$

Figure 4.3: Structural congruence and reduction

4.3.3 Formal Semantics

The formal semantics of this calculus can be defined in two ways. We can use a structural congruence and a reduction relation (Figure 4.3), as in Chapter 2. We identify processes up to renaming of bound names and variables. We just add reduction rules for the destructor application, (Red Destr 1) and (Red Destr 2), corresponding respectively to the success and failure of the destructor application. In rule (Red I/O), we allow communications on channels that can be any term. We do not consider frames, extended processes, or labeled transitions. (We define the adversary as a process in Definition 4.1 below, so we do not need to study the interaction of the protocol with an external environment, which is the purpose of the labeled semantics.)

We can also define the semantics by a reduction relation on semantic configurations, as in Figure 4.4. A semantic configuration is a pair $E, \mathcal{P}$ where the environment E is a finite set of names and $\mathcal{P}$ is a finite multiset of closed processes. The environment E must contain at least all free names of processes in $\mathcal{P}$. The configuration $\{a_1, \ldots, a_n\}, \{P_1, \ldots, P_n\}$ corresponds intuitively to the process $\mathsf{new}\ a_1 \ldots \mathsf{new}\ a_n.(P_1 \parallel \ldots \parallel P_n)$. The semantics of the calculus is defined by a reduction relation $\rightarrow$ on semantic configurations, shown in Figure 4.4. The rule (Red Res) is the only one that uses renaming. The new semantics (in particular the fact that only (Red Res) uses renaming) provides simplifications in the definitions of correspondences (see Section 4.3.4.2) and in the proofs that correspondences hold. Except when mentioned otherwise, we will focus on this new semantics.

4.3.4 Security Properties

4.3.4.1 Secrecy

We assume that the protocol is executed in the presence of an adversary that can listen to all messages, compute, and send all messages it has, following the so-called Dolev-Yao model [?]. Thus, an adversary can be represented by any process that has a set of public names S in its

$$E, \mathcal{P} \cup \{\, 0 \,\} \rightarrow E, \mathcal{P} \qquad \text{(Red Nil)}$$

$$E, \mathcal{P} \cup \{\, !P \,\} \rightarrow E, \mathcal{P} \cup \{\, P, !P \,\} \qquad \text{(Red Repl)}$$

$$E, \mathcal{P} \cup \{\, P \parallel Q \,\} \rightarrow E, \mathcal{P} \cup \{\, P, Q \,\} \qquad \text{(Red Par)}$$

$$E, \mathcal{P} \cup \{\, \mathsf{new}\ a.P \,\} \rightarrow E \cup \{a'\}, \mathcal{P} \cup \{\, P\{a'/a\} \,\} \qquad \text{(Red Res)}$$

$$\text{where } a' \notin E.$$

$$E, \mathcal{P} \cup \{\, \mathsf{out}(N, M).Q, \mathsf{in}(N, x).P \,\} \rightarrow E, \mathcal{P} \cup \{\, Q, P\{M/x\} \,\} \qquad \text{(Red I/O)}$$

$$E, \mathcal{P} \cup \{\, \mathsf{let}\ x = g(M_1, \dots, M_n)\ \mathsf{in}\ P\ \mathsf{else}\ Q \,\} \rightarrow E, \mathcal{P} \cup \{\, P\{M'/x\} \,\} \qquad \text{(Red Destr 1)}$$

$$\text{if } g(M_1, \dots, M_n) \rightarrow M'$$

$$E, \mathcal{P} \cup \{\, \mathsf{let}\ x = g(M_1, \dots, M_n)\ \mathsf{in}\ P\ \mathsf{else}\ Q \,\} \rightarrow E, \mathcal{P} \cup \{\, Q \,\} \qquad \text{(Red Destr 2)}$$

$$\text{if there exists no } M' \text{ such that } g(M_1, \dots, M_n) \rightarrow M'$$

$$E, \mathcal{P} \cup \{\, \mathsf{let}\ x = M\ \mathsf{in}\ P \,\} \rightarrow E, \mathcal{P} \cup \{\, P\{M/x\} \,\} \qquad \text{(Red Let)}$$

$$E, \mathcal{P} \cup \{\, \mathsf{if}\ M = M\ \mathsf{then}\ P\ \mathsf{else}\ Q \,\} \rightarrow E, \mathcal{P} \cup \{\, P \,\} \qquad \text{(Red Cond 1)}$$

$$E, \mathcal{P} \cup \{\, \mathsf{if}\ M = N\ \mathsf{then}\ P\ \mathsf{else}\ Q \,\} \rightarrow E, \mathcal{P} \cup \{\, Q \,\} \qquad \text{(Red Cond 2)}$$

$$\text{if } M \neq N$$

Figure 4.4: Operational semantics

initial knowledge. (Although the initial knowledge of the adversary contains only names in S, one can give any terms to the adversary by sending them on a channel in S.)

Definition 4.1 *Let S be a finite set of names. The closed process Q is an S-*adversary *if and only if $fn(Q) \subseteq S$.*

Intuitively, a process P preserves the secrecy of M when M cannot be output on a public channel, in a run of P with any adversary. Formally, we define that a trace outputs M as follows:

Definition 4.2 *We say that a trace $\mathcal{T} = E_0, \mathcal{P}_0 \rightarrow^* E', \mathcal{P}'$ outputs M if and only if $\mathcal{T}$ contains a reduction $E, \mathcal{P} \cup \{\, \mathsf{out}(c, M).Q, \mathsf{in}(c, x).P \,\} \rightarrow E, \mathcal{P} \cup \{\, Q, P\{M/x\} \,\}$ for some E, $\mathcal{P}$, x, P, Q, and $c \in S$.*

We can finally define secrecy:

Definition 4.3 *The closed process P* preserves the secrecy of *M from S if and only if for any S-adversary Q, for any E_0 containing $fn(P_0) \cup S \cup fn(M)$, for any trace $\mathcal{T} = E_0, \{P_0, Q\} \rightarrow^* E', \mathcal{P}'$, the trace $\mathcal{T}$ does not output M.*

This notion of secrecy is similar to that of [?, ?, ?]: a term M is secret if the adversary cannot get it by listening and sending messages, and performing computations.

4.3.4.2 Correspondences

As already mentioned in Section 2.3.2, correspondence properties are used to capture relationships between events that can be expressed in the form "if an event e has been executed, then events $e'_1, \dots, e'_n$ have been previously executed." Moreover, these events may contain arguments. Correspondences can be used to model authentication [?, ?].

In order to define correspondences, we add an instruction for executing events in the syntax of processes:

$P, Q ::=$	processes
$\quad \dots$	(see Figure 4.2)
$\quad \mathsf{event}(M).P$	event

The semantics of events is defined by the transition:

$$E, \mathcal{P} \cup \{\, \mathsf{event}(M).P \,\} \rightarrow E, \mathcal{P} \cup \{\, P \,\} \qquad \text{(Red Event)}$$

We recall that, in the semantics of Figure 4.4, the rule (Red Res) is the only one that uses renaming. This is important so that the parameters of events are not renamed after the execution of the event, to be able to compare them with the parameters of events executed later. This is why this semantics simplifies the definition of correspondences.

We adapt the definition of an S-adversary, so that it does not contain events.

Non-injective correspondences

Definition 4.4 *We say that a trace $\mathcal{T} = E_0, \mathcal{P}_0 \rightarrow^* E', \mathcal{P}'$ executes $\mathsf{event}(M)$ if and only if $\mathcal{T}$ contains a reduction $E, \mathcal{P} \cup \{\, \mathsf{event}(M).P \,\} \rightarrow E, \mathcal{P} \cup \{\, P \,\}$ for some E, $\mathcal{P}$, P.*

The correspondence $\mathsf{event}(M) \rightsquigarrow \bigwedge_{k=1}^{l} \mathsf{event}(M_k)$, formally defined below, means intuitively that, if an instance of $\mathsf{event}(M)$ is executed, then a corresponding instance of each of the events $\mathsf{event}(M_1)$, …, $\mathsf{event}(M_l)$ has been executed.

Definition 4.5 *The closed process P_0* satisfies the correspondence

$$\mathsf{event}(M) \rightsquigarrow \bigwedge_{k=1}^{l} \mathsf{event}(M_k)$$

against S-adversaries if and only if, for any S-adversary Q, for any E_0 containing $fn(P_0) \cup S \cup fn(M) \cup \bigcup_k fn(M_k)$, for any substitution σ, for any trace $\mathcal{T} = E_0, \{P_0, Q\} \rightarrow^* E', \mathcal{P}'$, if $\mathcal{T}$ executes $\mathsf{event}(\sigma M)$, then there exists σ' such that $\sigma' M = \sigma M$ and, for all $k \in \{1, \dots, l\}$, $\mathcal{T}$ executes $\mathsf{event}(\sigma' M_k)$ as well.

The variables in M are universally quantified (because, in Definition 4.5, σ is universally quantified). The variables in M_k that do not occur in M are existentially quantified (because σ' is existentially quantified).

This definition is very general; a particular case is non-injective agreement:

Definition 4.6 Non-injective agreement *is a correspondence of the form $\mathsf{event}(e(x_1, \dots, x_n)) \rightsquigarrow \mathsf{event}(e'(x_1, \dots, x_n))$.*

Intuitively, the correspondence $\mathsf{event}(e(x_1, \dots, x_n)) \rightsquigarrow \mathsf{event}(e'(x_1, \dots, x_n))$ means that, if an event $e(M_1, \dots, M_n)$ is executed, then the event $e'(M_1, \dots, M_n)$ has also been executed. This definition can be used to represent Lowe's notion of non-injective agreement [?].

Injective correspondences

Definition 4.7 *We say that the event $\mathsf{event}(M)$ is executed at step τ in a trace $\mathcal{T} = E_0, \mathcal{P}_0 \rightarrow^* E', \mathcal{P}'$ if and only if the τ-th reduction of $\mathcal{T}$ is of the form $E, \mathcal{P} \cup \{\, \mathsf{event}(M).P \,\} \rightarrow E, \mathcal{P} \cup \{\, P \,\}$ for some E, $\mathcal{P}$, P.*

Intuitively, an injective correspondence $\mathsf{event}(M) \rightsquigarrow \mathsf{inj}\ \mathsf{event}(M')$ requires that each event $\mathsf{event}(\sigma M)$ is enabled by distinct events $\mathsf{event}(\sigma M')$, while a non-injective correspondence $\mathsf{event}(M) \rightsquigarrow \mathsf{event}(M')$ allows several events $\mathsf{event}(\sigma M)$ to be enabled by the same event $\mathsf{event}(\sigma M')$.

Definition 4.8 *The closed process P_0* satisfies the correspondence

$$\mathsf{event}(M) \rightsquigarrow \bigwedge_{k=1}^{l} \mathsf{inj}\ \mathsf{event}(M_k)$$

against S-adversaries if and only if, for any S-adversary Q, for any E_0 containing $fn(P_0) \cup S \cup fn(M) \cup \bigcup_k fn(M_k)$, for any trace $\mathcal{T} = E_0, \{P_0, Q\} \rightarrow^* E', \mathcal{P}'$, there exist injective functions ϕ_{jk} from a subset of steps in $\mathcal{T}$ to steps in $\mathcal{T}$ such that for all τ, if the event $\mathsf{event}(\sigma M)$ *is executed at step τ in $\mathcal{T}$ for some σ, then there exists σ' such that $\sigma' M = \sigma M$ and, for all $k \in \{1, \ldots, l\}$, $\phi_k(\tau)$ is defined and* $\mathsf{event}(\sigma' M_k)$ *is executed at step $\phi_k(\tau)$ in $\mathcal{T}$.*

The functions ϕ_k map execution steps of events $\mathsf{event}(\sigma M)$ to the execution steps of the events $\mathsf{event}(\sigma' M_k)$ that enable $\mathsf{event}(\sigma M)$. The injectivity of ϕ_k guarantees that distinct executions of $\mathsf{event}(\sigma M)$ correspond to distinct executions of $\mathsf{event}(\sigma' M_k)$.

Definition 4.9 Injective agreement *is a correspondence of the form* $\mathsf{event}(e(x_1, \ldots, x_n)) \rightsquigarrow \mathsf{inj}\ \mathsf{event}(e'(x_1, \ldots, x_n))$.

Injective agreement requires that the number of executions of $\mathsf{event}(e(M_1, \ldots, M_n))$ is smaller than the number of executions of $\mathsf{event}(e'(M_1, \ldots, M_n))$: each execution of $\mathsf{event}(e(M_1, \ldots, M_n))$ corresponds to a distinct execution of $\mathsf{event}(e'(M_1, \ldots, M_n))$. This corresponds to Lowe's agreement specification [**?**].

More general definitions of correspondences can be found in [**?**].

Example As an example, we consider a simplified version of the Woo and Lam one-way public-key authentication protocol, version of [**?**], in which host names are replaced by public keys, which makes interaction with a server useless. The protocol is:

Message 1.	$A \rightarrow B:$	pk_A
Message 2.	$B \rightarrow A:$	b
Message 3.	$A \rightarrow B:$	$\{pk_A, pk_B, b\}_{sk_A}$

A first sends to B its public key, B replies with a nonce (fresh name), and A sends its public key, the public key of B, and the nonce all signed with its private key sk_A. This protocol can be represented by the process P:

$$\begin{aligned}
&P_A(sk_A, pk_A) = !\ \mathsf{in}(c, x\_pk_B).\mathsf{event}(e_A(x\_pk_B)).\mathsf{out}(c, pk_A).\mathsf{in}(c, x\_b).\\
&\quad \mathsf{out}(c, \mathsf{sign}((pk_A, x\_pk_B, x\_b), sk_A))\\
&P_B(sk_B, pk_B, pk_A) = !\ \mathsf{in}(c, x\_pk_A).\mathsf{new}\ b.\mathsf{out}(c, b).\mathsf{in}(c, m).\\
&\quad \mathsf{if}\ (x\_pk_A, pk_B, b) = \mathsf{check}(m, x\_pk_A)\ \mathsf{then}\\
&\quad \mathsf{if}\ x\_pk_A = pk_A\ \mathsf{then}\ \mathsf{event}(e_B(pk_B))\\
&P = \mathsf{new}\ sk_A.\mathsf{new}\ sk_B.\mathsf{let}\ pk_A = \mathsf{pk}(sk_A)\ \mathsf{in}\ \mathsf{let}\ pk_B = \mathsf{pk}(sk_B)\ \mathsf{in}\\
&\quad \mathsf{out}(c, pk_A).\mathsf{out}(c, pk_B).(P_A(sk_A, pk_A) \parallel P_B(sk_B, pk_B, pk_A))
\end{aligned}$$

The channel c is public: The adversary can send and listen on it. The process P begins with the creation of the secret and public keys of A and B. The public keys are output on channel c to model that the adversary has them in its initial knowledge. Then the protocol itself starts: P_A represents A, P_B represents B. Both principals can run an unbounded number of sessions, hence the replications. We consider that A and B are both willing to talk to any principal. So, to determine to whom A will talk, we consider that A first inputs a message containing the public key of its interlocutor (this interlocutor is therefore chosen by the adversary). Then A starts the protocol by executing an event $\mathsf{event}(e_A(x\_pk_B))$, whose intuitive meaning is "A has started a session with the host of public key x_pk_B". At the end of P_B, when B thinks he talks with A (that is, when $x\_pk_A = pk_A$), he executes an event $\mathsf{event}(e_B(pk_B))$, whose meaning is "$B$ thinks he has completed a session with A".[1] If the protocol is correct, B should only execute $\mathsf{event}(e_B(pk_B))$ when A has first executed $\mathsf{event}(e_A(pk_B))$. This can be formalized by the correspondence:

$$\mathsf{event}(e_B(x)) \rightsquigarrow \mathsf{event}(e_A(x))$$

We may also require that each execution of e_B corresponds to a distinct event e_A, so that different runs of B correspond to different runs of A. This is formalized by the injective correspondence:

$$\mathsf{event}(e_B(x)) \rightsquigarrow \mathsf{inj}\ \mathsf{event}(e_A(x))$$

This is what we are going to check next.

4.3.4.3 Observational Equivalence

The notion of indistinguishability is a powerful concept which allows us to reason about complex properties that cannot be expressed as reachability or correspondence properties. The notion of indistinguishability is generally named *observational equivalence* in the formal model. Intuitively, two processes P and Q are observationally equivalent, written $P \approx Q$, when an active adversary cannot distinguish P from Q. Using this notion, one can for instance specify that a process P follows its specification Q by saying that $P \approx Q$. ProVerif can prove some observational equivalences, but not all of them because their proof is complex. Formally, observational equivalence can be defined as follows, using the first semantics of ProVerif's calculus. An evaluation context C is a closed context built from $[\,]$, $C \parallel P$, $P \parallel C$, and $\mathsf{new}\ a.C$.

Definition 4.10 *The process P* emits on *M ($P\!\downarrow_M$) if and only if $P \equiv C[\mathsf{out}(M,N).R]$ for some evaluation context C that does not bind $fn(M)$.*

Observational equivalence $\approx$ *is the largest symmetric relation $\mathcal{R}$ on closed processes such that $P\ \mathcal{R}\ Q$ implies*

1. *if $P \rightarrow^* \downarrow_M$ then $Q \rightarrow^* \downarrow_M$;*
2. *if $P \rightarrow^* P'$ then $Q \rightarrow^* Q'$ and $P'\ \mathcal{R}\ Q'$ for some Q';*
3. *$C[P]\ \mathcal{R}\ C[Q]$ for all evaluation contexts C.*

In the applied pi calculus, this notion coincides with the notion of labeled bisimilarity presented in Section 2.3.4 [**?**]. It can be defined without using labeled transitions; on the other hand, it uses a universal quantification over all contexts, which complicates its proof. (Typically, the goal is to prove observational equivalence; a proof method is to design a labeled transition system such that observational equivalence is equivalent to labeled bisimilarity, and to prove labeled bisimilarity instead.)

Observational equivalence can also be used to formalize a notion of secrecy, strong secrecy, which means that the attacker is unable to distinguish when the secret changes. In other words, the value of the secret should not affect the observable behavior of the protocol. Such a notion is useful to capture the adversary's ability to learn partial information about the secret: when the adversary learns the first component of a pair, for instance, the whole pair is secret in the sense of reachability (the adversary cannot reconstruct the whole pair because it does not have the second component), but it is not secret in the sense of strong secrecy (the adversary can notice changes in the value of the pair, since it has its first component). Strong secrecy also detects

[1] Note that the event e_B must not be executed when B thinks he talks to the adversary. Indeed, in this case, it is correct that no event e_A has been executed by the interlocutor of B, since the adversary never executes events.

$p ::=$	patterns
$\quad x$	$\quad$ variable
$\quad a[p_1, \dots, p_n]$	$\quad$ name
$\quad f(p_1, \dots, p_n)$	$\quad$ constructor application
$F ::= \mathit{pred}(p_1, \dots, p_n)$	fact
$R ::= F_1 \wedge \dots \wedge F_n \Rightarrow F$	Horn clause

Figure 4.5: Syntax of ProVerif's internal protocol representation

implicit flows, which happen for instance when the secret is compared with another value. The concept of strong secrecy is particularly important when the secret consists of known values. Consider for instance a process P that uses a boolean b. The variable b can take two values, true or false, which are both known to the adversary, so it is not secret in the sense of reachability. However, one may express that b is strongly secret by saying that $P\{\mathsf{true}/b\} \approx P\{\mathsf{false}/b\}$: the adversary cannot determine whether b is true or false. Strong secrecy can be formally defined as follows:

Definition 4.11 *P preserves the secrecy of its free variables if and only if for all closed substitutions θ and θ' such that $\mathsf{Dom}(\theta) = \mathsf{Dom}(\theta') = \mathit{fv}(P)$, $P\theta \approx P\theta'$. (The substitution avoids name captures by first alpha renaming P if necessary.)*

4.3.5 Some Other Models

In addition to variants of the pi calculus, such as the spi calculus [**?**] and the applied pi calculus [**?**], there also exist other models of security protocols, for instance strand spaces [**?**] and multiset rewriting [**?**].

4.4 The Horn Clause Representation of Protocols

4.4.1 Definition of this Representation

Internally, ProVerif translates the protocol into a representation by a set of Horn clauses; the syntax of these clauses is given in Figure 4.5. In this figure, x ranges over variables, a over names, f over function symbols, and p over predicate symbols. The patterns p represent messages that are exchanged between participants of the protocol. (Patterns are terms; we use the word patterns to distinguish them from terms of the process calculus.) A variable can represent any pattern. Names represent atomic values, such as keys and nonces (random numbers). Each principal has the ability of creating new names: fresh names are created at each run of the protocol. Here, the created names are considered as functions of the messages previously received by the principal that creates the name. Thus, names are distinguished only when the preceding messages are different. As noticed by Martín Abadi (personal communication), this approximation is in fact similar to the approximation done in some type systems (such as [**?**]): the type of the new name depends on the types in the environment. It is enough to handle many protocols, and can be enriched by adding other parameters to the name. In particular, we shall see in Section 4.6 that, by adding as parameter a session identifier that takes a different value in each run of the protocol, one can distinguish all names. This is necessary for proving authentication but not for secrecy, so we omit session identifiers here for simplicity. The constructor applications $f(M_1, \dots, M_n)$ build patterns. A fact $F = \mathit{pred}(p_1, \dots, p_n)$ expresses a property of the messages $p_1, \dots, p_n$. Several predicates pred can be used but, for a first example, we are going to use a single predicate $\mathsf{attacker}$, such that the fact $\mathsf{attacker}(p)$ means "the attacker may have the message p". A clause $R = F_1 \wedge \ldots \wedge F_n \Rightarrow F$ means that, if all facts $F_1, \ldots, F_n$ are true, then F is also true. A clause with no hypothesis $\Rightarrow F$ is written simply F.

We use illustrate the encoding of a protocol on the example of Section 4.3.2:

$$\begin{array}{ll} \text{Message 1.} & A \rightarrow B : \{\{k\}_{sk_A}\}_{pk_B} \\ \text{Message 2.} & B \rightarrow A : \{s\}_k \end{array}$$

4.4.1.1 Representation of the Abilities of the Attacker

We first present the encoding of the computation abilities of the attacker. The encoding of the protocol itself will be detailed in Section 4.4.1.2.

During its computations, the attacker can apply all constructors and destructors. If f is a constructor of arity n, this leads to the clause:

$$\mathsf{attacker}(x_1) \wedge \ldots \wedge \mathsf{attacker}(x_n) \Rightarrow \mathsf{attacker}(f(x_1, \ldots, x_n)).$$

If g is a destructor, for each rewrite rule $g(M_1, \ldots, M_n) \rightarrow M$ in $\mathrm{def}(g)$, we have the clause:

$$\mathsf{attacker}(M_1) \wedge \ldots \wedge \mathsf{attacker}(M_n) \Rightarrow \mathsf{attacker}(M).$$

The destructors never appear in the clauses, they are coded by pattern-matching on their parameters (here $M_1, \ldots, M_n$) in the hypothesis of the clause and generating their result in the conclusion. In the particular case of public-key encryption, this yields:

$$\begin{aligned} &\mathsf{attacker}(m) \wedge \mathsf{attacker}(pk) \Rightarrow \mathsf{attacker}(\mathsf{aenc}(m, pk)), \\ &\mathsf{attacker}(sk) \Rightarrow \mathsf{attacker}(\mathsf{pk}(sk)), \\ &\mathsf{attacker}(\mathsf{aenc}(m, \mathsf{pk}(sk))) \wedge \mathsf{attacker}(sk) \Rightarrow \mathsf{attacker}(m), \end{aligned} \tag{4.1}$$

where the first two clauses correspond to the constructors aenc and pk, and the last clause corresponds to the destructor pdec. When the attacker has an encrypted message $\mathsf{aenc}(m, pk)$ and the decryption key sk, then it also has the cleartext m. (We assume that the cryptography is perfect, hence the attacker can obtain the cleartext from the encrypted message only if it has the key.)

Clauses for signatures (sign, $\mathsf{getmess}$, check) and for shared-key encryption (senc, sdec) are given in Figure 4.6.

The clauses above describe the computation abilities of the attacker. Moreover, the attacker initially has the public keys of the protocol participants. Therefore, we add the clauses $\mathsf{attacker}(\mathsf{pk}(sk_A[\,]))$ and $\mathsf{attacker}(\mathsf{pk}(sk_B[\,]))$. We also give a name a to the attacker, that will represent all names it can generate: $\mathsf{attacker}(a[\,])$. In particular, $a[\,]$ can represent the secret key of any dishonest participant, his public key being $\mathsf{pk}(a[\,])$, which the attacker can compute by the clause for constructor pk.

4.4.1.2 Representation of the Protocol Itself

Now, we describe how the protocol itself is represented. We consider that A and B are willing to talk to any principal, A, B but also malicious principals that are represented by the attacker. Therefore, the first message sent by A can be $\mathsf{aenc}(\mathsf{sign}(k, sk_A[\,]), \mathsf{pk}(x))$ for any x. We leave to the attacker the task of starting the protocol with the principal it wants, that is, the attacker will send a preliminary message to A, mentioning the public key of the principal with which A should talk. This principal can be B, or another principal represented by the attacker. Hence, if the attacker has some key $\mathsf{pk}(x)$, it can send $\mathsf{pk}(x)$ to A; A replies with his first message, which the attacker can intercept, so the attacker obtains $\mathsf{aenc}(\mathsf{sign}(k, sk_A[\,]), \mathsf{pk}(x))$. Therefore, we have a clause of the form

$$\mathsf{attacker}(\mathsf{pk}(x)) \Rightarrow \mathsf{attacker}(\mathsf{aenc}(\mathsf{sign}(k, sk_A[\,]), \mathsf{pk}(x))).$$

Moreover, a new key k is created each time the protocol is run. Hence, if two different keys $\mathsf{pk}(x)$ are received by A, the generated keys k are certainly different: k depends on $\mathsf{pk}(x)$. The clause becomes:

$$\mathsf{attacker}(\mathsf{pk}(x)) \Rightarrow \mathsf{attacker}(\mathsf{aenc}(\mathsf{sign}(k[\mathsf{pk}(x)], sk_A[\,]), \mathsf{pk}(x))). \tag{4.2}$$

When B receives a message, he decrypts it with his secret key sk_B, so B expects a message of the form $\mathsf{aenc}(x', \mathsf{pk}(sk_B[\,]))$. Next, B tests whether A has signed x', that is, B evaluates $\mathsf{check}(x', pk_A)$, and this succeeds only when $x' = \mathsf{sign}(y, sk_A[\,])$. If so, he assumes that the key y is only known by A, and sends a secret s (a constant that the attacker does not have a priori) encrypted under y. We assume that the attacker relays the message coming from A, and intercepts the message sent by B. Hence the clause:

$$\mathsf{attacker}(\mathsf{aenc}(\mathsf{sign}(y, sk_A[\,]), \mathsf{pk}(sk_B[\,]))) \Rightarrow \mathsf{attacker}(\mathsf{senc}(\mathsf{s}, y)).$$

Remark 4.1 *With these clauses, A cannot play the role of B and vice-versa. In order to model a situation in which all principals play both roles, we can replace all occurrences of $sk_B[\,]$ with $sk_A[\,]$ in the clauses above. Then A plays both roles, and is the only honest principal. A single honest principal is sufficient for proving secrecy properties by [**?**].*

More generally, a protocol that contains n messages is encoded by n sets of clauses. If a principal X sends the ith message, the ith set of clauses contains clauses that have as hypotheses the patterns of the messages previously received by X in the protocol, and as conclusion the pattern of the ith message. There may be several possible patterns for the previous messages as well as for the sent message, in particular when the principal X uses a function defined by several rewrite rules, such as the function exp of Section 4.5.3. In this case, a clause must be generated for each combination of possible patterns. Moreover, the hypotheses of the clauses describe all messages previously received, not only the last one. This is important since in some protocols the fifth message for instance can contain elements received in the first message. The hypotheses summarize the history of the exchanged messages.

4.4.1.3 Summary

To sum up, a protocol can be represented by three sets of Horn clauses, as detailed in Figure 4.6 for the protocol of Section 4.3.2:

- Clauses representing the computation abilities of the attacker: constructors, destructors, and name generation.
- Facts corresponding to the initial knowledge of the attacker. In general, there are facts giving the public keys of the participants and/or their names to the attacker.
- Clauses representing the messages of the protocol itself. There is one set of clauses for each message in the protocol. In the set corresponding to the ith message, sent by principal X, the clauses are of the form $\mathsf{attacker}(p_{j_1}) \wedge \ldots \wedge \mathsf{attacker}(p_{j_n}) \Rightarrow \mathsf{attacker}(p_i)$ where p_{j_1}, $\ldots$, p_{j_n} are the patterns of the messages received by X before sending the ith message, and p_i is the pattern of the ith message.

Exercice 20
Give the representation by Horn clauses of the Needham-Schroeder public-key protocol of Figure 1.1.

Computation abilities of the attacker:

For each constructor f of arity n:

$$\mathsf{attacker}(x_1) \wedge \ldots \wedge \mathsf{attacker}(x_n) \Rightarrow \mathsf{attacker}(f(x_1, \ldots, x_n))$$

For each destructor g, for each rewrite rule $g(M_1, \ldots, M_n) \rightarrow M$ in $\mathrm{def}(g)$:

$$\mathsf{attacker}(M_1) \wedge \ldots \wedge \mathsf{attacker}(M_n) \Rightarrow \mathsf{attacker}(M)$$

that is

aenc	$\mathsf{attacker}(m) \wedge \mathsf{attacker}(pk) \Rightarrow \mathsf{attacker}(\mathsf{aenc}(m, pk))$
pk	$\mathsf{attacker}(sk) \Rightarrow \mathsf{attacker}(\mathsf{pk}(sk))$
pdec	$\mathsf{attacker}(\mathsf{aenc}(m, \mathsf{pk}(sk))) \wedge \mathsf{attacker}(sk) \Rightarrow \mathsf{attacker}(m)$
sign	$\mathsf{attacker}(m) \wedge \mathsf{attacker}(sk) \Rightarrow \mathsf{attacker}(\mathsf{sign}(m, sk))$
getmess	$\mathsf{attacker}(\mathsf{sign}(m, sk)) \Rightarrow \mathsf{attacker}(m)$
check	$\mathsf{attacker}(\mathsf{sign}(m, sk)) \wedge \mathsf{attacker}(\mathsf{pk}(sk)) \Rightarrow \mathsf{attacker}(m)$
senc	$\mathsf{attacker}(m) \wedge \mathsf{attacker}(k) \Rightarrow \mathsf{attacker}(\mathsf{senc}(m, k))$
sdec	$\mathsf{attacker}(\mathsf{senc}(m, k)) \wedge \mathsf{attacker}(k) \Rightarrow \mathsf{attacker}(m)$
Name generation:	$\mathsf{attacker}(a[\,])$

Initial knowledge: $\mathsf{attacker}(\mathsf{pk}(sk_A[\,]))$, $\mathsf{attacker}(\mathsf{pk}(sk_B[\,]))$

The protocol:

First message: $\mathsf{attacker}(\mathsf{pk}(x)) \Rightarrow \mathsf{attacker}(\mathsf{aenc}(\mathsf{sign}(k[\mathsf{pk}(x)], sk_A[\,]), \mathsf{pk}(x)))$

Second message: $\mathsf{attacker}(\mathsf{aenc}(\mathsf{sign}(y, sk_A[\,]), \mathsf{pk}(sk_B[\,]))) \Rightarrow \mathsf{attacker}(\mathsf{senc}(\mathsf{s}, y))$

Figure 4.6: Summary the Horn clause representation of the protocol of Section 4.3.2

4.4.1.4 Approximations

The reader can notice that the Horn clause representation of protocols is approximate. Specifically, the number of repetitions of each action is ignored, since Horn clauses can be applied any number of times. So a step of the protocol can be completed several times, as long as the previous steps have been completed at least once between the same principals (even when future steps have already been completed). For instance, consider the following protocol (communicated by Véronique Cortier)

First step: A sends $\{(N_1, M)\}_k, \{(N_2, M)\}_k$
Second step: If A receives $\{(x, M)\}_k$, he replies with x
Third step: If A receives N_1, N_2, he replies with s

where N_1, N_2, and M are nonces. In an exact model, A never sends s, since $\{(N_1, M)\}_k$ or $\{(N_2, M)\}_k$ can be decrypted, but not both. In the Horn clause model, even though the first step is executed once, the second step may be executed twice for the same M (that is, the corresponding clause can be applied twice), so that both $\{(N_1, M)\}_k$ and $\{(N_2, M)\}_k$ can be decrypted, and A may send s. We have a false attack against the secrecy of s.

However, the important point is that the approximations are sound: if an attack exists in a more precise model, such as the applied pi calculus [?] or multiset rewriting [?], then it also exists in the Horn clause representation. This is shown for the applied pi calculus in [?] and for multiset rewriting in [?]. In particular, [?] shows formally that the only approximation with respect to the multiset rewriting model is that the number of repetitions of actions is ignored. Performing approximations enables us to build a much more efficient verifier, which will be able to handle larger and more complex protocols. Another advantage is that the verifier does not have to limit the number of runs of the protocol. The price to pay is that false attacks may be found by the verifier: sequences of clause applications that do not correspond to a protocol run, as illustrated above. False attacks appear in particular for protocols with temporary secrets: when some value first needs to be kept secret and is revealed later in the protocol, the Horn clause model considers that this value can be reused in the beginning of the protocol, thus breaking the protocol. When a false attack is found, we cannot know whether the protocol is secure or not: a real attack may also exist. A more precise analysis is required in this case. Fortunately, the Horn clause representation is precise enough so that false attacks are rare. (This is demonstrated by the experiments, see Section 4.8.)

4.4.1.5 Secrecy Criterion

A basic goal is to determine secrecy properties: for instance, can the attacker get the secret s? That is, can the fact $\mathsf{attacker}(\mathsf{s})$ be derived from the clauses? If $\mathsf{attacker}(\mathsf{s})$ can be derived, the sequence of clauses applied to derive $\mathsf{attacker}(\mathsf{s})$ will lead to the description of an attack. This is the notion of secrecy of Section 4.3.4.1.

In our running example, $\mathsf{attacker}(\mathsf{s})$ is derivable from the clauses. The derivation is as follows. The attacker generates a fresh name $a[\,]$ (considered as a secret key), it computes $\mathsf{pk}(a[\,])$ by the clause for pk, obtains $\mathsf{aenc}(\mathsf{sign}(k[\mathsf{pk}(a[\,])], sk_A[\,]), \mathsf{pk}(a[\,]))$ by the clause for the first message. It decrypts this message using the clause for pdec and its knowledge of $a[\,]$, thus obtaining $\mathsf{sign}(k[\mathsf{pk}(a[\,])], sk_A[\,])$. It reencrypts the signature under $\mathsf{pk}(sk_B[\,])$ by the clause for aenc (using its initial knowledge of $\mathsf{pk}(sk_B[\,])$), thus obtaining $\mathsf{aenc}(\mathsf{sign}(k[\mathsf{pk}(a[\,])], sk_A[\,]), \mathsf{pk}(sk_B[\,]))$. By the clause for the second message, it obtains $\mathsf{senc}(\mathsf{s}, k[\mathsf{pk}(a[\,])])$. On the other hand, from $\mathsf{sign}(k[\mathsf{pk}(a[\,])], sk_A[\,])$, it obtains $k[\mathsf{pk}(a[\,])]$ by the clause for $\mathsf{getmess}$, so it can decrypt $\mathsf{senc}(\mathsf{s}, k[\mathsf{pk}(a[\,])])$ by the clause for sdec, thus obtaining s. In other words, the attacker starts a session between A and a dishonest participant of secret key $a[\,]$. It gets the first message $\mathsf{aenc}(\mathsf{sign}(k, sk_A[\,]), \mathsf{pk}(a[\,]))$, decrypts it, reencrypts it under $\mathsf{pk}(sk_B[\,])$, and sends it to B. For B, this message looks like the first message of a session between A and B, so B replies with $\mathsf{senc}(\mathsf{s}, k)$, which the attacker can decrypt since it obtains k from the first message. The obtained derivation corresponds to the known attack against this protocol. In contrast, if we fix the protocol by adding the public key of B in the first message $\{\{(pk_B, k)\}_{sk_A}\}_{pk_B}$, $\mathsf{attacker}(\mathsf{s})$ is not derivable from the clauses, so the fixed protocol preserves the secrecy of s.

Next, we formally define when a given fact can be derived from a given set of clauses. We shall see in the next section how we determine that. Technically, the hypotheses $F_1, \ldots, F_n$ of a clause are considered as a multiset. This means that the order of the hypotheses is irrelevant, but the number of times a hypothesis is repeated is important. (This is not related to multiset rewriting models of protocols: the semantics of a clause does not depend on the number of repetitions of its hypotheses, but considering multisets is necessary in the proof of the resolution algorithm.) We use R for clauses (logic programming *rules*), H for hypothesis, and C for conclusion.

Definition 4.12 (Subsumption) *We say that $H_1 \Rightarrow C_1$ subsumes $H_2 \Rightarrow C_2$, and we write $(H_1 \Rightarrow C_1) \sqsupseteq (H_2 \Rightarrow C_2)$, if and only if there exists a substitution σ such that $\sigma C_1 = C_2$ and $\sigma H_1 \subseteq H_2$ (multiset inclusion).*

We write $R_1 \sqsupseteq R_2$ when R_2 can be obtained by adding hypotheses to a particular instance of R_1. In this case, all facts that can be derived by R_2 can also be derived by R_1.

A derivation is defined as follows, as illustrated in Figure 4.7.

Definition 4.13 (Derivability) *Let F be a closed fact, that is, a fact without variable. Let $\mathcal{R}$ be a set of clauses. F is derivable from $\mathcal{R}$ if and only if there exists a derivation of F from $\mathcal{R}$, that is, a finite tree defined as follows:*

1. *Its nodes (except the root) are labeled by clauses $R \in \mathcal{R}$;*
2. *Its edges are labeled by closed facts;*
3. *If the tree contains a node labeled by R with one incoming edge labeled by F_0 and n outgoing edges labeled by $F_1, \ldots, F_n$, then $R \sqsupseteq F_1 \wedge \ldots \wedge F_n \Rightarrow F_0$.*

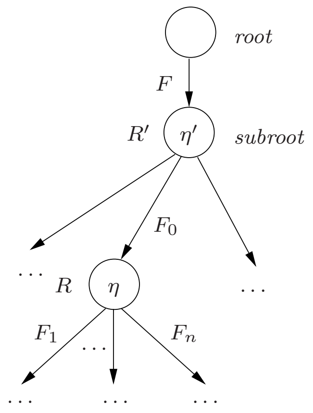


Figure 4.7: Derivation of F

4. The root has one outgoing edge, labeled by F. The unique son of the root is named the subroot*.*

In a derivation, if there is a node labeled by R with one incoming edge labeled by F_0 and n outgoing edges labeled by $F_1, \dots, F_n$, then F_0 can be derived from $F_1, \dots, F_n$ by the clause R. Therefore, there exists a derivation of F from $\mathcal{R}$ if and only if F can be derived from clauses in $\mathcal{R}$ (in classical logic).

4.4.2 Resolution Algorithm

The internal protocol representation is a set of Horn clauses, and our goal is to determine whether a given fact can be derived from these clauses or not. This is exactly the problem solved by usual Prolog systems. However, we cannot use such systems here, because they would not terminate. For instance, the clause:

$$\mathsf{attacker}(\mathsf{aenc}(m, \mathsf{pk}(sk))) \wedge \mathsf{attacker}(sk) \Rightarrow \mathsf{attacker}(m)$$

leads to considering more and more complex terms, with an unbounded number of encryptions. We could of course limit arbitrarily the depth of terms to solve the problem, but we can do much better than that.

As detailed below, the main idea is to combine pairs of clauses by resolution, and to guide this resolution process by a selection function: ProVerif's resolution algorithm is resolution with free selection [**?**, **?**, **?**]. This algorithm is similar to ordered resolution with selection, used by [**?**], but without the ordering constraints.

Notice that, since a term is secret when a fact is *not* derivable from the clauses, soundness in terms of security (if the verifier claims that there is no attack, then there is no attack) corresponds to the completeness of the resolution algorithm in terms of logic programming (if the algorithm claims that a fact is not derivable, then it is not). The resolution algorithm that we use must therefore be complete.

4.4.2.1 The Basic Algorithm

Let us first define resolution: when the conclusion of a clause R unifies with a hypothesis of another (or the same) clause R', resolution infers a new clause that corresponds to applying R and R' one after the other. Formally, resolution is defined as follows:

Definition 4.14 *Let R and R' be two clauses, $R = H \Rightarrow C$, and $R' = H' \Rightarrow C'$. Assume that there exists $F_0 \in H'$ such that C and F_0 are unifiable and σ is the most general unifier of C*

```
$\mathsf{saturate}(\mathcal{R}_0) =$
  1. $\mathcal{R} \leftarrow \emptyset$.
     For each $R \in \mathcal{R}_0$, $\mathcal{R} \leftarrow \mathsf{elim}(\{R\} \cup \mathcal{R})$.
  2. Repeat until a fixpoint is reached
       for each $R \in \mathcal{R}$ such that $sel(R) = \emptyset$,
         for each $R' \in \mathcal{R}$, for each $F_0 \in sel(R')$ such that $R \circ_{F_0} R'$ is defined,
           $\mathcal{R} \leftarrow \mathsf{elim}(\{R \circ_{F_0} R'\} \cup \mathcal{R})$.
  3. Return $\{R \in \mathcal{R} \mid sel(R) = \emptyset\}$.
```

Figure 4.8: Resolution algorithm

and F_0. In this case, we define $R \circ_{F_0} R' = \sigma(H \cup (H' \setminus \{F_0\})) \Rightarrow \sigma C'$. The clause $R \circ_{F_0} R'$ is the result of resolving R' with R upon F_0; it can be inferred from R and R':

$$\frac{R = H \Rightarrow C \qquad R' = H' \Rightarrow C'}{R \circ_{F_0} R' = \sigma(H \cup (H' \setminus \{F_0\})) \Rightarrow \sigma C'}$$

For example, if R is the clause (4.2), R' is the clause (4.1), and the fact F_0 is $F_0 = \mathsf{attacker}(\mathsf{aenc}(m, \mathsf{pk}(sk)))$, then $R \circ_{F_0} R'$ is

$$\mathsf{attacker}(\mathsf{pk}(x)) \wedge \mathsf{attacker}(x) \Rightarrow \mathsf{attacker}(\mathsf{sign}(k[\mathsf{pk}(x)], sk_A[\,]))$$

with the substitution $\sigma = \{sk \mapsto x, m \mapsto \mathsf{sign}(k[\mathsf{pk}(x)], sk_A[\,])\}$.

We guide the resolution by a selection function:

Definition 4.15 *A* selection function *sel is a function from clauses to sets of facts, such that $sel(H \Rightarrow C) \subseteq H$. If $F \in sel(R)$, we say that F is selected in R. If $sel(R) = \emptyset$, we say that no hypothesis is selected in R, or that the conclusion of R is selected.*

The resolution algorithm is correct (sound and complete) with any selection function, as we show below. However, the choice of the selection function can change dramatically the behavior of the algorithm. The essential idea of the algorithm is to combine clauses by resolution only when the facts unified in the resolution are selected. We will therefore choose the selection function to reduce the number of possible unifications between selected facts. Having several selected facts slows down the algorithm, because it has more choices of resolutions to perform, therefore we will select at most one fact in each clause. In the case of protocols, facts of the form $\mathsf{attacker}(x)$, with x variable, can be unified will all facts of the form $\mathsf{attacker}(p)$. Therefore, we should avoid selecting them. So a basic selection function is a function sel_0 that satisfies the constraint

$$sel_0(H \Rightarrow C) = \begin{cases} \emptyset & \text{if } \forall F \in H, \exists x \text{ variable, } F = \mathsf{attacker}(x) \\ \{F_0\} & \text{where } F_0 \in H \text{ and } \forall x \text{ variable, } F_0 \neq \mathsf{attacker}(x) \end{cases} \tag{4.3}$$

The resolution algorithm is described in Figure 4.8. It transforms the initial set of clauses into a new one that derives the same facts.

The resolution algorithm, $\mathsf{saturate}(\mathcal{R}_0)$, contains 3 steps.

- The first step inserts in $\mathcal{R}$ the initial clauses representing the protocol and the attacker (clauses that are in $\mathcal{R}_0$), after elimination of subsumed clauses by elim: if R' subsumes R, and R and R' are in $\mathcal{R}$, then R is removed by $\mathsf{elim}(\mathcal{R})$.
- The second step is a fixpoint iteration that adds clauses created by resolution. The resolution of clauses R and R' is added only if no hypothesis is selected in R and the hypothesis F_0 of R' that we unify is selected. When a clause is created by resolution, it is added to the set of clauses $\mathcal{R}$. Subsumed clauses are eliminated from $\mathcal{R}$.

- At last, the third step returns the set of clauses of $\mathcal{R}$ with no selected hypothesis.

Basically, $\mathsf{saturate}$ preserves derivability (it is both sound and complete):

Theorem 4.2 (Correctness of $\mathsf{saturate}$) *Let F be a closed fact. F is derivable from $\mathcal{R}_0$ if and only if it is derivable from $\mathsf{saturate}(\mathcal{R}_0)$.*

This result is proved by transforming a derivation of F from $\mathcal{R}_0$ into a derivation of F from $\mathsf{saturate}(\mathcal{R}_0)$. Basically, when the derivation contains a clause R' with $sel(R') \neq \emptyset$, we replace in this derivation two clauses R, with $sel(R) = \emptyset$, and R' that have been combined by resolution during the execution of $\mathsf{saturate}$ with a single clause $R \circ_{F_0} R'$. This replacement decreases the number of clauses in the derivation, so it terminates, and, upon termination, all clauses of the obtained derivation satisfy $sel(R') = \emptyset$ so they are in $\mathsf{saturate}(\mathcal{R}_0)$. A detailed proof is given in Section 4.4.2.2.

Usually, resolution with selection is used for proofs by refutation. That is, the negation of the goal F is added to the clauses, under the form of a clause without conclusion: $F \Rightarrow$. The goal F is derivable if and only if the empty clause "$\Rightarrow$" can be derived. Here, for non-closed goals, we also want to be able to know which instances of the goal can be derived. That is why we prove that the clauses in $\mathsf{saturate}(\mathcal{R}_0)$ derive the same facts as the clauses in $\mathcal{R}_0$.

We can determine which instances of $pred(p_1, \dots, p_n)$ are derivable, as follows:

Corollary 4.1 *Let $\mathsf{solve}_{\mathcal{R}_0}(pred(p_1, \dots, p_n)) = \{H \Rightarrow pred(p'_1, \dots, p'_n) \mid H \Rightarrow pred'(p'_1, \dots, p'_n) \in \mathsf{saturate}(\mathcal{R}'_0)\}$, where $pred'$ is a new predicate and $\mathcal{R}'_0 = \mathcal{R}_0 \cup \{pred(p_1, \dots, p_n) \Rightarrow pred'(p_1, \dots, p_n)\}$.*

The fact $\sigma pred(p_1, \dots, p_n)$ is derivable from $\mathcal{R}_0$ if and only if there exists a clause $H \Rightarrow pred(p'_1, \dots, p'_n)$ in $\mathsf{solve}_{\mathcal{R}_0}(pred(p_1, \dots, p_n))$ and a substitution σ' such that $\sigma' pred(p'_1, \dots, p'_n) = \sigma pred(p_1, \dots, p_n)$ and $\sigma' H$ is derivable from $\mathcal{R}'_0$.

Proof : The fact $\sigma pred(p_1, \dots, p_n)$ is derivable from $\mathcal{R}_0$ if and only if $\sigma pred'(p_1, \dots, p_n)$ is derivable from $\mathcal{R}'_0$, so by Theorem 4.2, if and only if $\sigma pred'(p_1, \dots, p_n)$ is derivable from $\mathsf{saturate}(\mathcal{R}'_0)$, so if and only if there exists a clause $H \Rightarrow pred(p'_1, \dots, p'_n)$ in $\mathsf{solve}_{\mathcal{R}_0}(pred(p_1, \dots, p_n))$ and a substitution σ' such that $\sigma' pred(p'_1, \dots, p'_n) = \sigma pred(p_1, \dots, p_n)$ and $\sigma' H$ is derivable from $\mathsf{saturate}(\mathcal{R}'_0)$, that is, from $\mathcal{R}'_0$. □

In particular, if $\mathsf{solve}_{\mathcal{R}_0}(\mathsf{attacker}(p)) = \emptyset$, then $\mathsf{attacker}(p)$ is not derivable from $\mathcal{R}_0$ (and if $\mathsf{solve}_{\mathcal{R}_0}(\mathsf{attacker}(p))$ is not empty for the selection function sel_0, at least one instance of $\mathsf{attacker}(p)$ is derivable, since H will contain facts of the form $\mathsf{attacker}(x)$, an instance of which is derivable by $\mathsf{attacker}(a[\,])$).

4.4.2.2 Proofs

In this section, we detail the proof of Theorem 4.2. We first need to prove a few preliminary lemmas. The first one shows that two nodes in a derivation can be replaced by one when combining their clauses by resolution.

Lemma 4.1 (Resolution) *Consider a derivation containing a node η', labeled R'. Let F_0 be a hypothesis of R'. Then there exists a son η of η', labeled R, such that the edge $\eta' \rightarrow \eta$ is labeled by an instance of F_0, $R \circ_{F_0} R'$ is defined, and one obtains a derivation of the same fact by replacing the nodes η and η' with a node η'' labeled $R'' = R \circ_{F_0} R'$.*

Proof : This proof is illustrated in Figure 4.9. Let $R' = H' \Rightarrow C'$, H'_1 be the multiset of the labels of the outgoing edges of η', and C'_1 the label of its incoming edge. We have $R' \sqsupseteq (H'_1 \Rightarrow C'_1)$, so there exists a substitution σ such that $\sigma H' \subseteq H'_1$ and $\sigma C' = C'_1$. Since $F_0 \in H'$, $\sigma F_0 \in H'_1$, so there is an outgoing edge of η' labeled σF_0. Let η be the node at the

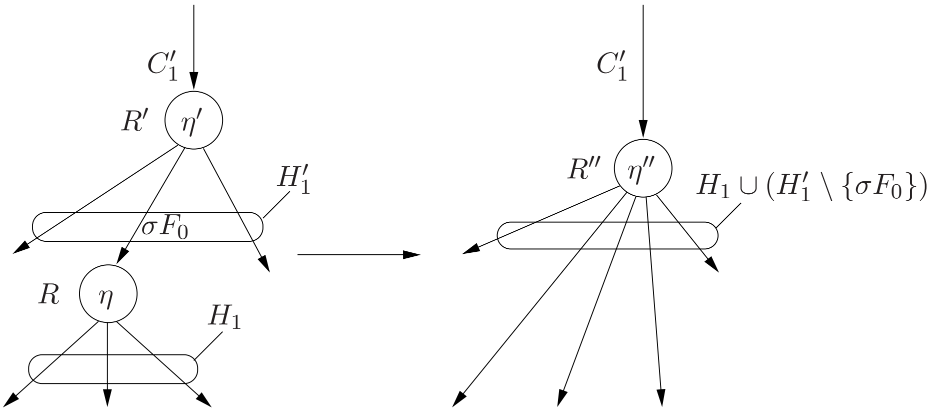


Figure 4.9: Merging of nodes of Lemma 4.1

end of this edge, let $R = H \Rightarrow C$ be the label of η. We rename the variables of R so that they are distinct from the variables of R'. Let H_1 be the multiset of the labels of the outgoing edges of η. So $R \sqsupseteq (H_1 \Rightarrow \sigma F_0)$. By the above choice of distinct variables, we can then extend σ so that $\sigma H \subseteq H_1$ and $\sigma C = \sigma F_0$.

The edge $\eta' \to \eta$ is labeled σF_0, instance of F_0. Since $\sigma C = \sigma F_0$, the facts C and F_0 are unifiable, so $R \circ_{F_0} R'$ is defined. Let σ' be the most general unifier of C and F_0, and σ'' such that $\sigma = \sigma''\sigma'$. We have $R \circ_{F_0} R' = \sigma'(H \cup (H' \setminus \{F_0\})) \Rightarrow \sigma' C'$. Moreover, $\sigma''\sigma'(H \cup (H' \setminus \{F_0\})) \subseteq H_1 \cup (H'_1 \setminus \{\sigma F_0\})$ and $\sigma''\sigma' C' = \sigma C' = C'_1$. Hence $R'' = R \circ_{F_0} R' \sqsupseteq (H_1 \cup (H'_1 \setminus \{\sigma F_0\})) \Rightarrow C'_1$. The multiset of labels of outgoing edges of η'' is precisely $H_1 \cup (H'_1 \setminus \{\sigma F_0\})$ and the label of its incoming edge is C'_1, therefore we have obtained a correct derivation by replacing η and η' with η''. □

Lemma 4.2 (Subsumption) *If a node η of a derivation D is labeled by R, then one obtains a derivation D' of the same fact as D by relabeling η with a clause R' such that $R' \sqsupseteq R$.*

Proof : Let H be the multiset of labels of outgoing edges of the considered node η, and C be the label of its incoming edge. We have $R \sqsupseteq H \Rightarrow C$. By transitivity of $\sqsupseteq$, $R' \sqsupseteq H \Rightarrow C$. So we can relabel η with R'. □

Lemma 4.3 (Saturation) *At the end of* saturate, *$\mathcal{R}$ satisfies the following properties:*

1. *For all $R \in \mathcal{R}_0$, R is subsumed by a clause in $\mathcal{R}$;*

2. *Let $R \in \mathcal{R}$ and $R' \in \mathcal{R}$. Assume that $sel(R) = \emptyset$ and there exists $F_0 \in sel(R')$ such that $R \circ_{F_0} R'$ is defined. In this case, $R \circ_{F_0} R'$ is subsumed by a clause in $\mathcal{R}$.*

Proof : To prove the first property, let $R \in \mathcal{R}_0$. We show that, after the addition of R to $\mathcal{R}$, R is subsumed by a clause in $\mathcal{R}$.

In the first step of saturate, we execute the instruction $\mathcal{R} \leftarrow \mathsf{elim}(\{R\} \cup \mathcal{R})$. After execution of this instruction, R is subsumed by a clause in $\mathcal{R}$.

Assume that we execute $\mathcal{R} \leftarrow \mathsf{elim}(\{R''\} \cup \mathcal{R})$ for some clause R'' and that, before this execution, R is subsumed by a clause in $\mathcal{R}$, say R'. If R' is removed by this instruction, there exists a clause R'_1 in $\mathcal{R}$ that subsumes R', so by transitivity of subsumption, R'_1 subsumes R, hence R is subsumed by the clause $R'_1 \in \mathcal{R}$ after this instruction. If R' is not removed by this instruction, then R is subsumed by the clause $R' \in \mathcal{R}$ after this instruction.

Hence, at the end of saturate, R is subsumed by a clause in $\mathcal{R}$, which proves the first property.

In order to prove the second property, we just need to notice that the fixpoint is reached at the end of $\mathsf{saturate}$, so $\mathcal{R} = \mathsf{elim}(\{R \circ_{F_0} R'\} \cup \mathcal{R})$. Hence, $R \circ_{F_0} R'$ is eliminated by elim, so it is subsumed by some clause in $\mathcal{R}$. □

Proof of Theorem 4.2: Assume that F is derivable from $\mathcal{R}_0$ and consider a derivation of F from $\mathcal{R}_0$. We show that F is derivable from $\mathsf{saturate}(\mathcal{R}_0)$.

We consider the value of the set of clauses $\mathcal{R}$ at the end of $\mathsf{saturate}$. For each clause R in $\mathcal{R}_0$, R is subsumed by a clause in $\mathcal{R}$ (Lemma 4.3, Property 1). So, by Lemma 4.2, we can replace all clauses R in the considered derivation with a clause in $\mathcal{R}$. Therefore, we obtain a derivation D of F from $\mathcal{R}$.

Next, we build a derivation of F from $\mathcal{R}_1$, where $\mathcal{R}_1 = \mathsf{saturate}(\mathcal{R}_0)$. If D contains a node labeled by a clause not in $\mathcal{R}_1$, we can transform D as follows. Let η' be a lowest node of D labeled by a clause not in $\mathcal{R}_1$. So all sons of η' are labeled by elements of $\mathcal{R}_1$. Let R' be the clause labeling η'. Since $R' \notin \mathcal{R}_1$, $sel(R') \neq \emptyset$. Take $F_0 \in sel(R')$. By Lemma 4.1, there exists a son of η of η' labeled by R, such that $R \circ_{F_0} R'$ is defined, and we can replace η and η' with a node η'' labeled by $R \circ_{F_0} R'$. Since all sons of η' are labeled by elements of $\mathcal{R}_1$, $R \in \mathcal{R}_1$. Hence $sel(R) = \emptyset$. So, by Lemma 4.3, Property 2, $R \circ_{F_0} R'$ is subsumed by a clause R'' in $\mathcal{R}$. By Lemma 4.2, we can relabel η'' with R''. The total number of nodes strictly decreases since η and η' are replaced with a single node η''.

So we obtain a derivation D' of F from $\mathcal{R}$, such that the total number of nodes strictly decreases. Hence, this replacement process terminates. Upon termination, all clauses are in $\mathcal{R}_1$. So we obtain a derivation of F from $\mathcal{R}_1$, which is the expected result.

For the converse implication, notice that, if a fact is derivable from $\mathcal{R}_1$, then it is derivable from $\mathcal{R}$, and that all clauses added to $\mathcal{R}$ do not create new derivable facts: if a fact is derivable by applying the clause $R \circ_{F_0} R'$, then it is also derivable by applying R and R'. □

4.4.2.3 Optimizations

The resolution algorithm uses several optimizations, in order to speed up resolution. The first two are standard, while the last three are specific to protocols.

Elimination of duplicate hypotheses If a clause contains several times the same hypotheses, the duplicate hypotheses are removed, so that at most one occurrence of each hypothesis remains.

Elimination of tautologies If a clause has a conclusion that is already in the hypotheses, this clause is a tautology: it does not derive new facts. Such clauses are removed.

Elimination of hypotheses $\mathsf{attacker}(x)$ If a clause $H \Rightarrow C$ contains in its hypotheses $\mathsf{attacker}(x)$, where x is a variable that does not appear elsewhere in the clause, then the hypothesis $\mathsf{attacker}(x)$ is removed. Indeed, the attacker always has at least one message, so $\mathsf{attacker}(x)$ is always satisfied for some value of x.

Decomposition of data constructors A data constructor is a constructor f of arity n that comes with associated destructors g_i for $i \in \{1, \dots, n\}$ defined by $g_i(f(x_1, \dots, x_n)) \to x_i$. Data constructors are typically used for representing data structures. Tuples are examples of data constructors. For each data constructor f, the following clauses are generated:

$$\mathsf{attacker}(x_1) \wedge \ldots \wedge \mathsf{attacker}(x_n) \Rightarrow \mathsf{attacker}(f(x_1, \dots, x_n)) \tag{Rf}$$

$$\mathsf{attacker}(f(x_1, \dots, x_n)) \Rightarrow \mathsf{attacker}(x_i) \tag{Rg}$$

Therefore, $\mathsf{attacker}(f(p_1, \ldots, p_n))$ is derivable if and only if $\forall i \in \{1, \ldots, n\}$, $\mathsf{attacker}(p_i)$ is derivable. When a fact of the form $\mathsf{attacker}(f(p_1, \ldots, p_n))$ is met, it is replaced with $\mathsf{attacker}(p_1) \wedge \ldots \wedge \mathsf{attacker}(p_n)$. If this replacement is done in the conclusion of a clause $H \Rightarrow \mathsf{attacker}(f(p_1, \ldots, p_n))$, n clauses are created: $H \Rightarrow \mathsf{attacker}(p_i)$ for each $i \in \{1, \ldots, n\}$. This replacement is of course done recursively: if p_i itself is a data constructor application, it is replaced again. The clauses (Rf) and (Rg) for data constructors are left unchanged. (When $\mathsf{attacker}(x)$ cannot be selected, the clauses (Rf) and (Rg) for data constructors are in fact not necessary, because they generate only tautologies during resolution. However, when $\mathsf{attacker}(x)$ can be selected, which cannot be excluded with certain extensions, these clauses may become necessary for soundness.)

Secrecy assumptions When the user knows that a fact will not be derivable, he can tell it to the verifier. (When this fact is of the form $\mathsf{attacker}(p)$, the user tells that p remains secret.) The tool then removes all clauses which have this fact in their hypotheses. At the end of the computation, the tool checks that the fact is indeed underivable from the obtained clauses. If the user has given erroneous information, an error message is displayed. Even in this case, the verifier never wrongly claims that a protocol is secure.

Mentioning such underivable facts prunes the search space, by removing useless clauses. This speeds up the resolution algorithm. In most cases, the secret keys of the principals cannot be known by the attacker. So, examples of underivable facts are $\mathsf{attacker}(sk_A[\,])$, $\mathsf{attacker}(sk_B[\,])$, ...

For simplicity, the proofs given in Section 4.4.2.2 do not take into account these optimizations. For a full proof, we refer the reader to [**?**, Appendix C].

4.4.2.4 Termination

In general, the resolution algorithm may not terminate. (The derivability problem is undecidable.) In practice, however, it terminates in most examples.

Blanchet and Podelski have shown that it always terminates on a large and interesting class of protocols, the *tagged protocols* [**?**]. They consider protocols that use as cryptographic primitives only public-key encryption and signatures with atomic keys, shared-key encryption, message authentication codes, and hash functions. Basically, a protocol is tagged when each application of a cryptographic primitive is marked with a distinct constant tag. It is easy to transform a protocol into a tagged protocol by adding tags. For instance, our example of protocol can be transformed into a tagged protocol, by adding the tags c_0, c_1, c_2 to distinguish the encryptions and signature:

$$\begin{array}{ll} \text{Message 1.} & A \rightarrow B : \{(c_1, \{(c_0, k)\}_{sk_A})\}_{pk_B} \\ \text{Message 2.} & B \rightarrow A : \{(c_2, s)\}_k \end{array}$$

Adding tags preserves the expected behavior of the protocol, that is, the attack-free executions are unchanged. In the presence of attacks, the tagged protocol may be more secure. Hence, tagging is a feature of good protocol design, as explained e.g. in [**?**]: the tags are checked when the messages are received; they facilitate the decoding of the received messages and prevent confusions between messages. More formally, tagging prevents type-flaw attacks [**?**], which occur when a message is taken for another message. However, the tagged protocol is potentially more secure than its untagged version, so, in other words, a proof of security for the tagged protocol does not imply the security of its untagged version. (Note that the tagging scheme considered here differs from the tagging schemes of Section 6.3.)

To illustrate the effect of tagging, we consider the Needham-Schroeder shared-key protocol. The algorithm does not terminate on its original version, which is untagged. It terminates after adding tags. In this protocol, we have two messages of the form:

$$\begin{array}{lll} \text{Message 4.} & B \rightarrow A: & \{N_B\}_K \\ \text{Message 5.} & A \rightarrow B: & \{N_B - 1\}_K \end{array}$$

where N_B is a nonce. Representing this with a function $\mathsf{minusone}(x) = x - 1$, the algorithm does not terminate.

Indeed, message 5 is represented by a clause of the form:

$$H \wedge \mathsf{attacker}(\mathsf{senc}(n, k)) \Rightarrow \mathsf{attacker}(\mathsf{senc}(\mathsf{minusone}(n), k))$$

where the hypothesis H describes other messages previously received by A. After some resolution steps, we obtain a clause of the form

$$\mathsf{attacker}(\mathsf{senc}(n, K)) \Rightarrow \mathsf{attacker}(\mathsf{senc}(\mathsf{minusone}(n), K)) \qquad \text{(Loop)}$$

for some term K. The fact $\mathsf{attacker}(\mathsf{senc}(\mathsf{minusone}(N_B), K))$ is also derived, so a resolution step with (Loop) yields: $\mathsf{attacker}(\mathsf{senc}(\mathsf{minusone}(\mathsf{minusone}(N_B)), K))$. This can again be resolved with (Loop), so that we finally have a cycle that derives $\mathsf{attacker}(\mathsf{senc}(\mathsf{minusone}^n(N_B), K))$ for all n.

When tags are added, the rule (Loop) becomes:

$$\mathsf{attacker}(\mathsf{senc}((c_1, n), K)) \Rightarrow \mathsf{attacker}(\mathsf{senc}((c_2, \mathsf{minusone}(n)), K)) \qquad \text{(NoLoop)}$$

and the previous loop is removed because c_2 does not unify with c_1. The fact $\mathsf{attacker}(\mathsf{senc}((c_2, \mathsf{minusone}(N_B)), K))$ is derived, but this does not yield a loop.

Other authors have proved related results: Ramanujan and Suresh [**?**] have shown that secrecy is decidable for tagged protocols. However, their tagging scheme is stronger since it forbids blind copies. A blind copy happens when a protocol participant sends back part of a message he received without looking at what is contained inside this part. On the other hand, they obtain a decidability result, while [**?**] obtains a termination result for an algorithm which is sound, efficient in practice, but approximate. Arapinis and Duflot [**?**] extend this result but still forbid blind copies. Comon-Lundh and Cortier [**?**] show that an algorithm using ordered binary resolution, ordered factorization and splitting terminates on protocols that blindly copy at most one term in each message. In contrast, the result of [**?**] puts no limit on the number of blind copies, but requires tagging.

For protocols that are not tagged, heuristics have been designed to adapt the selection function in order to obtain termination more often. We refer the reader to [**?**, Section 8.2] for more details.

It is also possible to obtain termination in all cases at the cost of additional abstractions. For instance, Goubault-Larrecq shows that one can abstract the clauses into clauses in the decidable class $\mathcal{H}_1$ [**?**], by losing some relational information on the messages.

4.5 Translation from the Pi Calculus

Given a closed process P_0 in the language of Section 4.3 and a set of names S, ProVerif builds a set of Horn clauses, representing the protocol in parallel with any S-adversary, in the same style as the clauses presented in the previous section. This translation was originally given in [**?**]. The clauses use *facts* defined by the following grammar:

$F ::=$	facts
$\mathsf{attacker}(p)$	attacker knowledge
$\mathsf{mess}(p, p')$	message on a channel

The fact $\mathsf{attacker}(p)$ means that the attacker may have p, and the fact $\mathsf{mess}(p, p')$ means that the message p' may appear on channel p. The clauses are of the form $F_1 \wedge \ldots \wedge F_n \Rightarrow F$, where $F_1, \ldots, F_n, F$ are facts. They comprise clauses for the attacker and clauses for the protocol, defined below. These clauses form the set $\mathcal{R}_{P_0,S}$.

4.5.1 Clauses for the Attacker

The abilities of the attacker are represented by the following clauses:

$$\begin{array}{lr}
\text{For each } a \in S,\ \mathsf{attacker}(a[\,]) & \text{(Init)} \\
\mathsf{attacker}(b_0[\,]) & \text{(Rn)} \\
\text{For each public constructor } f \text{ of arity } n, & \\
\quad \mathsf{attacker}(x_1) \wedge \ldots \wedge \mathsf{attacker}(x_n) \Rightarrow \mathsf{attacker}(f(x_1, \ldots, x_n)) & \text{(Rf)} \\
\text{For each destructor } g, & \\
\quad \text{for each rewrite rule } g(M_1, \ldots, M_n) \to M \text{ in } \mathrm{def}(g), & \text{(Rg)} \\
\qquad \mathsf{attacker}(M_1) \wedge \ldots \wedge \mathsf{attacker}(M_n) \Rightarrow \mathsf{attacker}(M) & \\
\mathsf{mess}(x, y) \wedge \mathsf{attacker}(x) \Rightarrow \mathsf{attacker}(y) & \text{(Rl)} \\
\mathsf{attacker}(x) \wedge \mathsf{attacker}(y) \Rightarrow \mathsf{mess}(x, y) & \text{(Rs)}
\end{array}$$

The clause (Init) represents the initial knowledge of the attacker. The clause (Rn) means that the attacker can generate new names. The clauses (Rf) and (Rg) mean that the attacker can apply all operations to all terms it has, (Rf) for constructors, (Rg) for destructors. For (Rg), notice that the rewrite rules in $\mathrm{def}(g)$ do not contain names and that terms without names are also patterns, so the clauses have the required format. Clause (Rl) means that the attacker can listen on all channels it has, and (Rs) that it can send all messages it has on all channels it has.

If $c \in S$, we can replace all occurrences of $\mathsf{mess}(c[\,], p)$ with $\mathsf{attacker}(p)$ in the clauses. Indeed, these facts are equivalent by the clauses (Rl) and (Rs).

4.5.2 Clauses for the Protocol

When a function ρ associates a pattern with each name and variable, and f is a constructor, we extend ρ as a substitution by $\rho(f(M_1, \ldots, M_n)) = f(\rho(M_1), \ldots, \rho(M_n))$.

The translation $[\![P]\!]\rho H$ of a process P is a set of clauses, where ρ is a function that associates a pattern with each name and variable, and H is a sequence of facts of the form $\mathsf{mess}(p, p')$. The environment ρ maps each variable and name to its associated pattern representation. The sequence H keeps track of messages received by the process, since these may trigger other messages. The empty sequence is denoted by $\emptyset$; the concatenation of a fact F to the sequence H is denoted by $H \wedge F$.

$$\begin{array}{l}
[\![0]\!]\rho H = \emptyset \\
[\![P \mid\mid Q]\!]\rho H = [\![P]\!]\rho H \cup [\![Q]\!]\rho H \\
[\![!P]\!]\rho H = [\![P]\!]\rho H \\
[\![\mathsf{new}\ a.P]\!]\rho H = [\![P]\!](\rho[a \mapsto a[p'_1, \ldots, p'_n]\,])H \\
\quad \text{where } H = \mathsf{mess}(p_1, p'_1) \wedge \ldots \wedge \mathsf{mess}(p_n, p'_n) \\
[\![\mathsf{in}(M, x).P]\!]\rho H = [\![P]\!](\rho[x \mapsto x])(H \wedge \mathsf{mess}(\rho(M), x)) \\
[\![\mathsf{out}(M, N).P]\!]\rho H = [\![P]\!]\rho H \cup \{H \Rightarrow \mathsf{mess}(\rho(M), \rho(N))\} \\
[\![\mathsf{let}\ x = g(M_1, \ldots, M_n)\ \mathsf{in}\ P\ \mathsf{else}\ Q]\!]\rho H = \bigcup\{[\![P]\!]((\sigma\rho)[x \mapsto \sigma' p'])(\sigma H) \\
\quad \mid g(p'_1, \ldots, p'_n) \to p' \text{ is in } \mathrm{def}(g) \text{ and } (\sigma, \sigma') \text{ is a most general pair of} \\
\quad \text{substitutions such that } \sigma\rho(M_1) = \sigma' p'_1, \ldots, \sigma\rho(M_n) = \sigma' p'_n\} \cup [\![Q]\!]\rho H
\end{array}$$

$$\begin{aligned}
&[\![\mathsf{let}\ x = M\ \mathsf{in}\ P]\!]\rho H = [\![P]\!](\rho[x \mapsto \rho(M)])H \\
&[\![\mathsf{if}\ M = N\ \mathsf{then}\ P\ \mathsf{else}\ Q]\!]\rho H = [\![P]\!](\sigma\rho)(\sigma H) \cup [\![Q]\!]\rho H \\
&\quad \text{where } \sigma \text{ is the most general unifier of } \rho(M) \text{ and } \rho(N)
\end{aligned}$$

The translation of a process is a set of Horn clauses that express that it may send certain messages.

- The nil process does nothing, so its translation is empty.
- The clauses for the parallel composition of processes P and Q are the union of clauses for P and Q.
- The replication is ignored, because all Horn clauses are applicable arbitrarily many times.
- For the restriction, we replace the restricted name a in question with the pattern $a[p'_1, \ldots, p'_n]$, where $p'_1, \ldots, p'_n$ are the previous inputs.
- The sequence H is extended in the translation of an input, with the input in question.
- The translation of an output adds a clause, meaning that the output is triggered when all conditions in H are true.
- The translation of a destructor application is the union of the clauses for the cases where the destructor succeeds (with an appropriate substitution) and where the destructor fails. For simplicity, we assume that the else branch of destructors may always be executed; this is sufficient in most cases, since the else branch is often empty or just sends an error message. For a more precise treatment, see [**?**, Section 9.2].
- The conditional if $M = N$ then P else Q is in fact equivalent to let $x = \mathsf{equal}(M, N)$ in P else Q, where the destructor equal is defined by $\mathsf{equal}(x, x) \to x$, so the translation of the conditional is a particular case of the destructor application. We give it explicitly since it is particularly simple.

This translation of the protocol into Horn clauses introduces approximations. The actions are considered as implicitly replicated, since the clauses can be applied any number of times. This approximation implies that the tool fails to prove protocols that first need to keep some value secret and later reveal it. For instance, consider the process $\mathsf{new}\ d.(\mathsf{out}(d, s).\mathsf{out}(c, d) \parallel \mathsf{in}(d, x))$. This process preserves the secrecy of s, because s is output on the private channel d and received by the input on d, before the adversary gets to know d by the output of d on the public channel c. However, the Horn clause method cannot prove this property, because it treats this process like a variant with additional replications $\mathsf{new}\ d.(!\mathsf{out}(d, s).\mathsf{out}(c, d) \parallel !\mathsf{in}(d, x))$, which does not preserve the secrecy of s.

4.5.2.1 Summary and Correctness

Let $\rho = \{a \mapsto a[\,] \mid a \in \mathit{fn}(P_0)\}$. We define the clauses corresponding to the process P_0 as:

$$\mathcal{R}_{P_0,S} = [\![P_0]\!]\rho\emptyset \cup \{\mathsf{attacker}(a[\,]) \mid a \in S\} \cup \{(\mathrm{Rn}), (\mathrm{Rf}), (\mathrm{Rg}), (\mathrm{Rl}), (\mathrm{Rs})\}$$

Exercice 21
Translate the process of Section 4.3.2 into clauses. Compare with the clauses given in Section 4.4.

Exercice 22
Translate the process of Exercise 19 into clauses.

Theorem 4.3 (Correctness of the clauses) *Let P_0 be a closed process. Let M be a closed term and p be the pattern obtained from the term M by replacing all names a with $a[\,]$. If $\mathsf{attacker}(p)$ is not derivable from $\mathcal{R}_{P_0,S}$, then P_0 preserves the secrecy of M from S.*

The proof of this result relies on a type system to express the soundness of the clauses on P_0, and on the subject reduction of this type system to show that soundness of the clauses is preserved during all executions of the process. This technique was introduced in [?] where a similar result is proved. [?] also shows an equivalence between an instance of a generic type system for proving secrecy properties of protocols and the Horn clause verification method. This instance is the most precise instance of this generic type system.

By combining Theorem 4.3 with Corollary 4.1, we obtain:

Corollary 4.2 *Let P_0 be a closed process. Let M be a closed term and p be the pattern obtained from the term M by replacing all names a with $a[\,]$. If $\mathsf{solve}_{\mathcal{R}_{P_0,S}}(\mathsf{attacker}(p)) = \emptyset$, then P_0 preserves the secrecy of M from S.*

4.5.3 Extension to Equational Theories

ProVerif has been extended to handle primitives defined by equational theories [?]. The term algebra consists of constructors equipped with an equational theory, defined by a finite set of equations. For example, we can model a symmetric encryption scheme in which decryption always succeeds (but may return a meaningless message) by the equations

$$\begin{aligned} \mathsf{sdec}(\mathsf{senc}(x,y),y) &= x \\ \mathsf{senc}(\mathsf{sdec}(x,y),y) &= x \end{aligned} \tag{4.4}$$

where senc and sdec are constructors. The first equation is standard; the second one makes it possible to avoid that the equality test $\mathsf{senc}(\mathsf{sdec}(M,N),N) = M$ reveals that M is a ciphertext under N. These equations are satisfied by block ciphers, which are bijective.

We can also model the Diffie-Hellman key agreement [?] using equations. The Diffie-Hellman key agreement relies on the following property of modular exponentiation: $(g^a)^b = (g^b)^a = g^{ab}$ in a cyclic multiplicative subgroup G of $\mathbb{Z}_p^*$, where p is a large prime number and g is a generator of G, and on the assumption that it is difficult to compute g^{ab} from g^a and g^b, without knowing the random numbers a and b (computational Diffie-Hellman assumption), or on the stronger assumption that it is difficult to distinguish g^a, g^b, g^{ab} from g^a, g^b, g^c without knowing the random numbers a, b, and c (decisional Diffie-Hellman assumption). These properties are exploited to establish a shared key between two participants A and B of a protocol: A chooses randomly a and sends g^a to B; symmetrically, B chooses randomly b and sends g^b to A. A can then compute $(g^b)^a$, since it has a and receives g^b, while B computes $(g^a)^b$. These two values being equal, they can be used to compute the shared key. The adversary, on the other hand, has g^a and g^b but not a and b so by the computational Diffie-Hellman assumption, it cannot compute the key. (This exchange resists passive attacks only; to resist active attacks, we need additional ingredients, for instance signatures.) We can model the Diffie-Hellman key agreement by the equation [?, ?]

$$\mathsf{exp}(\mathsf{exp}(\mathsf{g},x),y) = \mathsf{exp}(\mathsf{exp}(\mathsf{g},y),x) \tag{4.5}$$

where g is a constant and exp is modular exponentiation. Obviously, this is a basic model: it models the main functional equation but misses many algebraic relations that exist in the group G.

The main idea of our extension to equations is to translate these equations into a set of rewrite rules associated to constructors. For instance, the equations (4.4) are translated into the rewrite rules

$$\begin{array}{ll} \mathsf{senc}(x,y) \to \mathsf{senc}(x,y) & \mathsf{sdec}(x,y) \to \mathsf{sdec}(x,y) \\ \mathsf{senc}(\mathsf{sdec}(x,y),y) \to x & \mathsf{sdec}(\mathsf{senc}(x,y),y) \to x \end{array} \tag{4.6}$$

while the equation (4.5) is translated into

$$\mathsf{exp}(x,y) \to \mathsf{exp}(x,y) \qquad \mathsf{exp}(\mathsf{exp}(\mathsf{g},x),y) \to \mathsf{exp}(\mathsf{exp}(\mathsf{g},y),x) \tag{4.7}$$

Intuitively, these rewrite rules allow one, by applying them *exactly once* for each constructor, to obtain the various forms of the terms modulo the considered equational theory.[2] The constructors are then simply evaluated like destructors in the calculus above. One has formally defined when a set of rewrite rules models an equational theory, and designed algorithms that compute from the equations the rewrite rules that model the equational theory in question [**?**, Section 5]. Then, each trace in the calculus with equational theory corresponds to a trace in the calculus with rewrite rules, and conversely [**?**, Lemma 1].[3] We are then reduced to the simpler case in which there are no equations. The main advantage of this technique is that resolution can still use ordinary syntactic unification (instead of having to use unification modulo the equational theory), and therefore remains efficient.

This extension to equations still has limitations: it does not allow us to model associative operations, such as exclusive or, since this would require an infinite number of rewrite rules. It may be possible to handle these symbols using unification modulo the equational theory instead of syntactic unification, at the cost of a larger complexity. In the case of a bounded number of sessions, exclusive or is handled in [**?**, **?**] and a more complete theory of modular exponentiation is handled in [**?**]. A unification algorithm for modular exponentiation is presented in [**?**]. For an unbounded number of sessions, extensions of the Horn clause approach that can handle XOR and Diffie-Hellman key agreements with more detailed algebraic relations (including equations of the multiplicative group modulo p) have been proposed by Küsters and Truderung: they handle XOR provided one of its two arguments is a constant in the clauses that model the protocol [**?**] and Diffie-Hellman key agreements provided the exponents are constants in the clauses that model the protocol [**?**]; they proceed by transforming the initial clauses into richer clauses on which the standard resolution algorithm is applied.

4.5.4 Extension to Scenarios with Several Phases

In some protocol studies, we consider scenarios in which a certain process is executed, then, in a second phase, this process stops and another process starts. For instance, when we model the compromise of long term keys, we consider that, in a first phase, the protocol runs normally, then, in a second phase, some keys are published. We aim at determining which secrets of the sessions of the protocol run in the first phase are preserved even though some keys are compromised. (This is the notion of forward secrecy.)

Such scenarios can be represented in ProVerif thanks to an extension of the syntax: the process $\mathsf{phase}\ n.P$ represents a process P that runs in phase number n. The system first runs the processes in phase 0. Then, at some point during execution, it moves to phase 1. At this point, only the processes $\mathsf{phase}\ n.P$ for $n \geq 1$ ready to run are kept (that is, the processes in phase 0 are stopped) and the processes $\mathsf{phase}\ 1.P$ are executed. Then, we move to phase 2, and so on.

This extension is translated into Horn clauses as follows. We consider predicates $\mathsf{attacker}_n$ and mess_n for each phase n, instead of the predicates $\mathsf{attacker}$ and mess. The clauses for the

[2] The rewrite rules like $\mathsf{sdec}(x,y) \to \mathsf{sdec}(x,y)$ are necessary so that sdec always succeeds. Thanks to this rule, the evaluation of $\mathsf{sdec}(M,N)$ succeeds and leaves this term unchanged when M is not of the form $\mathsf{senc}(M',N)$.

[3] More precisely, the inequality tests of (Red Destr 2) must still be performed modulo the equational theory, even in the calculus with rewrite rules.

protocol use the predicate mess_n to translate the process P in $\mathsf{phase}\ n.P$; the clauses for the adversary are repeated for each $\mathsf{attacker}_n$. Moreover, the clauses

$$\mathsf{attacker}_n(x) \Rightarrow \mathsf{attacker}_{n+1}(x) \tag{Rp}$$

for all n transmit the knowledge of the adversary from one phase to the next.

This extension was presented in [**?**, Section 8] and [**?**, Section 9.3]. An application of this extension will be mentioned in Section 4.7.1.

4.6 Extension to Correspondences

In this section, we extend the translation of the process calculus to Horn clauses, defined in the previous section for secrecy, to the proof of correspondences.

4.6.1 From Secrecy to Correspondences

In the analysis for secrecy, when $\mathsf{attacker}(p)$ is derivable from the clauses the attacker *may* have p, that is, when $\mathsf{attacker}(p)$ is not derivable from the clauses, we are sure that the attacker cannot have p, but the converse is not true, because the Horn clauses can be applied any number of times, which is not true in general for all actions of the process. Similarly, when $\mathsf{mess}(p, p')$ is derivable from the clauses, the message p' *may* be sent on channel p. Hence the analysis overapproximates the execution of actions.

Let us now consider that we want to prove a correspondence, for instance $\mathsf{event}(e_1(x)) \rightsquigarrow \mathsf{event}(e_2(x))$. In order to prove this correspondence, we can overapproximate the executions of event e_1: if we prove the correspondence with this overapproximation, it will also hold in the exact semantics. So we can easily extend our analysis for secrecy with an additional predicate event, such that $\mathsf{event}(p)$ means that $\mathsf{event}(p)$ may have been executed. We generate clauses $\mathsf{mess}(p_1, p'_1) \wedge \ldots \wedge \mathsf{mess}(p_n, p'_n) \Rightarrow \mathsf{event}(p)$ when the process executes $\mathsf{event}(p)$ after receiving $p'_1, \ldots, p'_n$ on channels $p_1, \ldots, p_n$ respectively. However, such an overapproximation cannot be done for the event e_2: if we prove the correspondence after overapproximating the execution of e_2, we are not really sure that e_2 will be executed, so the correspondence may be wrong in the exact semantics. Therefore, we have to use a different method for treating e_2.

We use the following idea: we fix the exact set $\mathcal{E}$ of allowed events $e_2(p)$ and, in order to prove $\mathsf{event}(e_1(x)) \rightsquigarrow \mathsf{event}(e_2(x))$, we check that only events $e_1(p)$ for p such that $e_2(p) \in \mathcal{E}$ can be executed. If we prove this property for any value of $\mathcal{E}$, we have proved the desired correspondence. So we introduce a predicate $\mathsf{m\text{-}event}$, such that $\mathsf{m\text{-}event}(e_2(p))$ is true if and only if $e_2(p) \in \mathcal{E}$. We generate clauses $\mathsf{mess}(p_1, p'_1) \wedge \ldots \wedge \mathsf{mess}(p_n, p'_n) \wedge \mathsf{m\text{-}event}(e_2(p_0)) \Rightarrow \mathsf{mess}(p, p')$ when the process outputs p' on channel p after executing the event $e_2(p_0)$ and receiving $p'_1, \ldots, p'_n$ on channels $p_1, \ldots, p_n$ respectively. In other words, the output of p' on channel p can be executed only when $\mathsf{m\text{-}event}(e_2(p_0))$ is true, that is, $e_2(p_0) \in \mathcal{E}$. (When the output of p' on channel p is under several events, the clause contains several $\mathsf{m\text{-}event}$ facts in its hypothesis. We also have similar clauses with $\mathsf{event}(e_1(p))$ instead of $\mathsf{mess}(p, p')$ when the event e_1 is executed after executing e_2 and receiving $p'_1, \ldots, p'_n$ on channels $p_1, \ldots, p_n$ respectively.)

For instance, if the events $e_2(p_1)$ and $e_2(p_2)$ are executed in a certain trace of the protocol, we define $\mathcal{E} = \{e_2(p_1), e_2(p_2)\}$, so that $\mathsf{m\text{-}event}(e_2(p_1))$ and $\mathsf{m\text{-}event}(e_2(p_2))$ are true and all other $\mathsf{m\text{-}event}$ facts are false. Then we show that the only events e_1 that may be executed are $e_1(p_1)$ and $e_1(p_2)$. We prove a similar result for all values of $\mathcal{E}$, which proves the desired correspondence.

In order to determine whether a fact is derivable from the clauses, we use a resolution-based algorithm. The resolution is performed for an unknown value of $\mathcal{E}$. So, basically, we keep $\mathsf{m\text{-}event}$ facts without trying to evaluate them (which we cannot do since $\mathcal{E}$ is unknown). In the vocabulary of resolution, we never select $\mathsf{m\text{-}event}$ facts. Thus the obtained result holds for any value of $\mathcal{E}$, which allows us to prove correspondences. In order to prove the correspondence $\mathsf{event}(e_1(x)) \rightsquigarrow \mathsf{event}(e_2(x))$, we show that $\mathsf{event}(e_1(p))$ is derivable only when $\mathsf{m\text{-}event}(e_2(p))$ holds. We transform the initial set of clauses into a set of clauses that derives the same facts. If, in the obtained set of clauses, all clauses that conclude $\mathsf{event}(e_1(p))$ contain $\mathsf{m\text{-}event}(e_2(p))$ in their hypotheses, then $\mathsf{event}(e_1(p))$ is derivable only when $\mathsf{m\text{-}event}(e_2(p))$ holds, so the desired correspondence holds.

We still have to solve one problem: the reasoning above does not take into account that patterns p which occur in clauses differ from terms M, which represent messages. In patterns, we use a special encoding of names: a name a created by a restriction $\mathsf{new}\ a$ is represented by a function $a[p_1, \dots, p_n]$ of the messages $p_1, \dots, p_n$ received above the restriction, so that names created after receiving different messages are distinguished in the analysis (which is important for the precision of the analysis). However, this encoding still merges names created by the same restriction after receiving the same messages. For example, in the process $!\mathsf{in}(c, x)\mathsf{new}\ a$, the names created by $\mathsf{new}\ a$ are represented by $a[x]$, so several names created for the same value of x are merged. This merging is not acceptable for the verification of correspondences, because when we prove $\mathsf{event}(e_1(x)) \rightsquigarrow \mathsf{event}(e_2(x))$, we must make sure that x contains exactly the same names in $e_1(x)$ and in $e_2(x)$. In order to solve this problem, we label each replication with a *session identifier* i, which is an integer that takes a different value for each copy of the process generated by the replication. We add session identifiers as arguments to our encoding of names, which becomes $a[M_1, \dots, M_n, i_1, \dots, i_{n'}]$ where $i_1, \dots, i_{n'}$ are the session identifiers of the replications above the restriction $\mathsf{new}\ a$. For example, in the process $!\mathsf{in}(c, x)\mathsf{new}\ a$, the names created by $\mathsf{new}\ a$ are represented by $a[x, i]$. Each execution of the restriction is then associated with a distinct value of the session identifiers $i_1, \dots, i_{n'}$, so each name has a distinct encoding. We detail and formalize this encoding in Section 4.6.2.

4.6.2 Instrumented Processes

We consider a closed process P_0 representing the protocol we wish to verify. We assume that the bound names of P_0 have been renamed so that they are pairwise distinct and distinct from names in $S \cup fn(P_0)$ and in the correspondence to prove. We denote by Q a particular adversary; below, we prove the correspondence properties for any Q. Furthermore, we assume that, in the initial configuration $E_0, \{P_0, Q\}$, the names of E_0 not in $S \cup fn(P_0)$ or in the correspondence to prove have been renamed to fresh names, and the bound names of Q have been renamed so that they are pairwise distinct and fresh. (These renamings do not change the satisfied correspondences, since $\mathsf{new}\ a.P$ and the renamed process $\mathsf{new}\ a'.P\{a'/a\}$ reduce to the same configuration by (Red Res).)

In order to define formally the patterns associated with a name, we use a notion of instrumented processes. The syntax of instrumented processes is defined as follows:

- The replication $!P$ is labeled with a variable i in the set V_s of variable session identifiers (disjoint from the set V_o of ordinary variables): $!^i P$. The process $!^i P$ represents copies of P for a countable number of values of i. The variable i is a session identifier. It indicates which copy of P, that is, which session, is executed.

- The restriction $\mathsf{new}\ a.P$ is labeled with a restriction label ℓ: $\mathsf{new}\ a : \ell.P$, where ℓ is either $a[M_1, \dots, M_n, i_1, \dots, i_{n'}]$ for restrictions in honest processes or $b_0[a[i_1, \dots, i_{n'}]]$ for restrictions in the adversary. The symbol b_0 is a special name function symbol, distinct from all other such symbols. Using a specific instrumentation for the adversary is helpful so that all names generated by the adversary are encoded by instances of $b_0[x]$. They are therefore easy to generate. This labeling of restrictions is similar to a Church-style typing: ℓ can be considered as the type of a. (This type is polymorphic since it can contain variables.)

The instrumented processes are then generated by the following grammar:

$P, Q ::=$	instrumented processes
$\quad !^i P$	replication
$\quad \mathsf{new}\ a : \ell.P$	restriction
$\quad \ldots$ (as in the standard calculus)	

For instrumented processes, a semantic configuration $S, E, \mathcal{P}$ consists of a set S of session identifiers that have not yet been used by $\mathcal{P}$, an environment E that is a mapping from names to closed patterns of the form $a[\ldots]$, and a finite multiset of instrumented processes $\mathcal{P}$. The first semantic configuration uses any countable set of session identifiers S_0. The domain of E must always contain all free names of processes in $\mathcal{P}$, and the initial environment maps all names a to the pattern $a[\,]$. The semantic rules (Red Repl) and (Red Res) become:

$$S, E, \mathcal{P} \cup \{\, !^i P \,\} \to S \setminus \{\lambda\}, E, \mathcal{P} \cup \{\, P\{\lambda/i\}, !^i P \,\} \text{ where } \lambda \in S \qquad \text{(Red Repl)}$$

$$\begin{aligned} &S, E, \mathcal{P} \cup \{\, \mathsf{new}\ a : \ell.P \,\} \\ &\quad \to S, E[a' \mapsto E(\ell)\,], \mathcal{P} \cup \{\, P\{a'/a\} \,\} \text{ if } a' \notin \mathsf{Dom}(E) \end{aligned} \qquad \text{(Red Res)}$$

where the mapping E is extended to all terms as a substitution by $E(f(M_1, \ldots, M_n)) = f(E(M_1), \ldots, E(M_n))$ and to restriction labels by $E(a[M_1, \ldots, M_n, i_1, \ldots, i_{n'}]) = a[E(M_1), \ldots, E(M_n), i_1, \ldots, i_{n'}]$ and $E(b_0[a[i_1, \ldots, i_{n'}]]) = b_0[a[i_1, \ldots, i_{n'}]]$, so that it maps terms and restriction labels to patterns. The rule (Red Repl) takes an unused constant session identifier λ in S, and creates a copy of P with session identifier λ. The rule (Red Res) creates a fresh name a', substitutes it for a in P, and adds to the environment E the mapping of a' to its encoding $E(\ell)$. Other semantic rules $E, \mathcal{P} \to E, \mathcal{P}'$ simply become $S, E, \mathcal{P} \to S, E, \mathcal{P}'$.

The instrumented process $P'_0 = \mathrm{instr}(P_0)$ associated with the process P_0 is built from P_0 as follows:

- We label each replication $!P$ of P_0 with a distinct, fresh session identifier i, so that it becomes $!^i P$.

- We label each restriction $\mathsf{new}\ a$ of P_0 with $a[t, s]$, so that it becomes $\mathsf{new}\ a : a[t, s]$, where s is the sequence of session identifiers that label replications above $\mathsf{new}\ a$ in the abstract syntax tree of P'_0, in the order from top to bottom; t is the sequence of variables x that store received messages in inputs $\mathsf{in}(M, x)$ above $\mathsf{new}\ a$ in P_0 and results of non-deterministic destructor applications $\mathsf{let}\ x = g(\ldots)\ \mathsf{in}\ P\ \mathsf{else}\ Q$ above $\mathsf{new}\ a$ in P_0. (A destructor is said to be non-deterministic when it may return several different results for the same arguments. Adding the result of destructor applications to t is useful to improve precision, only for non-deterministic destructors. For deterministic destructors, the result of the destructor can be uniquely determined from the other elements of t, so the addition is useless. If we add the result of non-deterministic destructors to t, we can show that the relative completeness result of [**?**] still holds in the presence of non-deterministic destructors. This result shows that, for secrecy, the Horn clause approach is at least as precise as a large class of type systems.)

  Hence names are represented by functions $a[t, s]$ of the inputs and results of destructor applications in t and the session identifiers in s. In each trace of the process, at most one name corresponds to a given $a[t, s]$, since different copies of the restriction have different values of session identifiers in s. Therefore, different names are not merged by the verifier.

For the adversary, we use a slightly different instrumentation. We build the instrumented process $Q' = \mathrm{instrAdv}(Q)$ as follows:

- We label each replication $!P$ of Q with a distinct, fresh session identifier i, so that it becomes $!^i P$.

- We label each restriction $\mathsf{new}\ a$ of Q with $b_0[a[s]]$, so that it becomes $\mathsf{new}\ a : b_0[a[s]]$, where s is the sequence of session identifiers that label replications above $\mathsf{new}\ a$ in Q'. (Including the session identifiers as arguments of nonces is necessary for soundness, as discussed in Section 4.6.1. Including the messages previously received as arguments of nonces is important for precision in the case of honest processes, in order to relate the nonces to these messages. It is however useless for the adversary: since we consider any S-adversary Q, we have no definite information on the relation between nonces generated by the adversary and messages previously received by the adversary.)

Example 4.1 *The instrumentation of the example process of Section 4.3.4.2 is*

$$\begin{aligned}
&P_A(sk_A, pk_A) = !^{i_A}\ \mathsf{in}(c, x\_pk_B).\mathsf{event}(e_A(x\_pk_B)).\mathsf{out}(c, pk_A).\mathsf{in}(c, x\_b).\\
&\qquad \mathsf{out}(c, \mathsf{sign}((pk_A, x\_pk_B, x\_b), sk_A))\\
&P_B(sk_B, pk_B, pk_A) = !^{i_B}\ \mathsf{in}(c, x\_pk_A).\mathsf{new}\ b : b[x\_pk_A, i_B].\mathsf{out}(c, b).\mathsf{in}(c, m).\\
&\qquad \mathsf{if}\ (x\_pk_A, pk_B, b) = \mathsf{check}(m, x\_pk_A)\ \mathsf{then}\\
&\qquad \mathsf{if}\ x\_pk_A = pk_A\ \mathsf{then}\ \mathsf{event}(e_B(pk_B))\\
&P = \mathsf{new}\ sk_A : sk_A[\,].\mathsf{new}\ sk_B : sk_B[\,].\mathsf{let}\ pk_A = \mathsf{pk}(sk_A)\ \mathsf{in}\ \mathsf{let}\ pk_B = \mathsf{pk}(sk_B)\ \mathsf{in}\\
&\qquad \mathsf{out}(c, pk_A).\mathsf{out}(c, pk_B).(P_A(sk_A, pk_A) \parallel P_B(sk_B, pk_B, pk_A))
\end{aligned}$$

The replications are instrumented with session identifiers i_A and i_B respectively, and the restriction $\mathsf{new}\ b$ *is instrumented with the pattern $b[x\_pk_A, i_B]$.*

The semantics of instrumented processes allows exactly the same communications and events as the one of standard processes. More precisely, let $\mathcal{P}$ be a multiset of instrumented processes. We define $\mathrm{unInstr}(\mathcal{P})$ as the multiset of processes of $\mathcal{P}$ without the instrumentation. We define that an instrumented trace executes an event $\mathsf{event}(M)$ (at step τ) by naturally extending Definitions 4.4 and 4.7. Thus we have:

Proposition 4.1 *If $E_0, \{P_0, Q\} \rightarrow^* E_1, \mathcal{P}_1$, then there exist E'_1 and $\mathcal{P}'_1$ such that for any S, countable set of session identifiers, there exists S' such that $S, \{a \mapsto a[\,] \mid a \in E_0\}, \{\mathrm{instr}(P_0), \mathrm{instrAdv}(Q)\} \rightarrow^* S', E'_1, \mathcal{P}'_1$, $\mathsf{Dom}(E'_1) = E_1$, $\mathrm{unInstr}(\mathcal{P}'_1) = \mathcal{P}_1$, and both traces execute the same events at the same steps.*

Conversely, if $S, \{a \mapsto a[\,] \mid a \in E_0\}, \{\mathrm{instr}(P_0), \mathrm{instrAdv}(Q)\} \rightarrow^* S', E'_1, \mathcal{P}'_1$, then $E_0, \{P_0, Q\} \rightarrow^* \mathsf{Dom}(E'_1), \mathrm{unInstr}(\mathcal{P}'_1)$, and both traces execute the same events at the same steps.

Proof : This is an easy proof by induction on the length of the traces. The reduction rules applied in both traces are rules with the same name. □

We say that $\mathsf{event}(p)$ is executed (at step τ) in the instrumented trace $\mathcal{T} = S_0, E_0, \mathcal{P}_0 \rightarrow^* S', E', \mathcal{P}'$ when there exists a term M such that $\mathsf{event}(M)$ is executed (at step τ) in $\mathcal{T}$ and $E'(M) = p$.

4.6.3 Generation of Horn Clauses and Resolution

Given a closed process P_0 and a set of names S, ProVerif first instruments P_0 to obtain $P'_0 = \mathrm{instr}(P_0)$, then it builds a set of Horn clauses, representing the protocol in parallel with any S-adversary. The clauses use the facts:

$F ::=$	facts
$\mathsf{attacker}(p)$	attacker knowledge
$\mathsf{mess}(p, p')$	message on a channel
$\mathsf{m\text{-}event}(p)$	must-event
$\mathsf{event}(p)$	may-event

The predicates $\mathsf{attacker}$ and mess are as before. The fact $\mathsf{m\text{-}event}(p)$ means that $\mathsf{event}(M)$ must have been executed with M corresponding to p, and $\mathsf{event}(p)$ that $\mathsf{event}(M)$ may have been executed with M corresponding to p. The clauses are of the form $F_1 \wedge \ldots \wedge F_n \Rightarrow F$, where F_1, $\ldots, F_n, F$ are facts. They comprise clauses for the attacker and clauses for the protocol, defined below. These clauses form the set $\mathcal{R}_{P_0',S}$.

The clauses describing the attacker are almost the same as for the verification of secrecy in Section 4.5.1. The only difference is that, here, the attacker is given an infinite set of fresh names $b_0[x]$, instead of only one fresh name $b_0[\,]$. Indeed, we cannot merge all fresh names created by the attacker, since we have to make sure that different terms are represented by different patterns for the verification of correspondences to be correctly implemented, as seen in Section 4.6.1. So the clause (Rn) becomes $\mathsf{attacker}(b_0[x])$.

The clauses for the protocol are modified as follows:

$$\begin{aligned}
&[\![!^i P]\!]\rho H = [\![P]\!](\rho[i \mapsto i])H\\
&[\![\mathsf{new}\ a : a[M_1, \ldots, M_n, i_1, \ldots, i_{n'}].P]\!]\rho H =\\
&\quad [\![P]\!](\rho[a \mapsto a[\rho(M_1), \ldots, \rho(M_n), \rho(i_1), \ldots, \rho(i_{n'})]\,])H\\
&[\![\mathsf{event}(M).P]\!]\rho H = [\![P]\!]\rho(H \wedge \mathsf{m\text{-}event}(\rho(M))) \cup \{H \Rightarrow \mathsf{event}(\rho(M))\}
\end{aligned}$$

The other cases are the same as for secrecy.

- The replication only inserts the new session identifier i in the environment ρ. It is otherwise ignored, because all Horn clauses are applicable arbitrarily many times.

- For the restriction, we replace the restricted name a in question with the pattern $a[\rho(M_1),$ $\ldots, \rho(M_n), \rho(i_1), \ldots, \rho(i_{n'})]$. By definition of the instrumentation, this pattern contains the previous inputs, results of non-deterministic destructor applications, and session identifiers.

- The translation of an event adds the hypothesis $\mathsf{m\text{-}event}(\rho(M))$ to H, meaning that P can be executed only if the event has been executed first. Furthermore, it adds a clause, meaning that the event is triggered when all conditions in H are true.

Remark 4.2 *Depending on the form of the correspondences we want to prove, we can sometimes simplify the clauses generated for events. Suppose that all arguments of events in the process and in correspondences are of the form $f(M_1, \ldots, M_n)$ for some function symbol f.*

If, for a certain function symbol f, events $\mathsf{event}(f(\ldots))$ occur only before $\rightsquigarrow$ in the desired correspondences, then it is easy to see in the following theorems that hypotheses of the form $\mathsf{m\text{-}event}(f(\ldots))$ in clauses can be removed without changing the result, so the clauses generated by the event $\mathsf{event}(M)$ when M is of the form $f(\ldots)$ can be simplified into:

$$[\![\mathsf{event}(M).P]\!]\rho H = [\![P]\!]\rho H \cup \{H \Rightarrow \mathsf{event}(\rho(M))\}$$

(Intuitively, since the events $\mathsf{event}(f(\ldots))$ occur only before $\rightsquigarrow$ in the desired correspondences, we never prove that an event $\mathsf{event}(f(\ldots))$ has been executed, so the facts $\mathsf{m\text{-}event}(f(\ldots))$ are useless.)

Similarly, if $\mathsf{event}(f(\ldots))$ occurs only after $\rightsquigarrow$ in the desired correspondences, then clauses that conclude a fact of the form $\mathsf{event}(f(\ldots))$ can be removed without changing the result, so the clauses generated by the event $\mathsf{event}(M)$ when M is of the form $f(\ldots)$ can be simplified into:

$$[\![\mathsf{event}(M).P]\!]\rho H = [\![P]\!]\rho(H \wedge \mathsf{m\text{-}event}(\rho(M)))$$

(Intuitively, since the events $\mathsf{event}(f(\ldots))$ occur only after $\rightsquigarrow$ in the desired correspondences, we never prove properties of the form "if $\mathsf{event}(f(\ldots))$ has been executed, then ... ", so clauses that conclude $\mathsf{event}(f(\ldots))$ are useless.)

Let $\rho = \{a \mapsto a[\,] \mid a \in \mathit{fn}(P_0')\}$. We define the clauses corresponding to the instrumented process P_0' as:

$$\mathcal{R}_{P_0',S} = [\![P_0']\!]\rho\emptyset \cup \{\mathsf{attacker}(a[\,]) \mid a \in S\} \cup \{(\mathrm{Rn}), (\mathrm{Rf}), (\mathrm{Rg}), (\mathrm{Rl}), (\mathrm{Rs})\}$$

Exercice 23
Give the clauses for the process of Section 4.3.4.2.

Exercice 24
Model mutual authentication in the Needham-Schroeder public-key protocol (see Figure 1.1), by correspondences in a process. Give the corresponding Horn clauses.

Theorem 4.4 (Correctness of the clauses) *Let P_0 be a closed process and Q be an S-adversary. Let $P_0' = \mathrm{instr}(P_0)$ and $Q' = \mathrm{instrAdv}(Q)$. Consider a trace $\mathcal{T} = S_0, E_0, \{P_0', Q'\} \rightarrow^* S', E', \mathcal{P}'$, with $\mathit{fn}(P_0') \cup S \subseteq \mathsf{Dom}(E_0)$ and $E_0(a) = a[\,]$ for all $a \in \mathsf{Dom}(E_0)$. Assume that, if $\mathcal{T}$ executes $\mathsf{event}(p)$, then $\mathsf{m\text{-}event}(p) \in \mathcal{F}_{\mathrm{me}}$. Finally, assume that $\mathcal{T}$ executes $\mathsf{event}(p')$. Then $\mathsf{event}(p')$ is derivable from $\mathcal{R}_{P_0',S} \cup \mathcal{F}_{\mathrm{me}}$.*

This result shows that, if the only executed events are those allowed in $\mathcal{F}_{\mathrm{me}}$ and an event $\mathsf{event}(p')$ is executed, then $\mathsf{event}(p')$ is derivable from the clauses. It is proved in [**?**, Appendix B], using a technique similar to that for secrecy.

We use the same resolution algorithm as for secrecy, except that we never select facts $\mathsf{m\text{-}event}(p)$. So the selection function becomes

$$sel_1(H \Rightarrow C) = \begin{cases} \emptyset & \text{if } \forall F \in H,\ F = \mathsf{attacker}(x) \text{ for some variable } x \text{ or} \\ & \qquad F = \mathsf{m\text{-}event}(p) \text{ for some pattern } p \\ \{F_0\} & \text{where } F_0 \in H \text{ and } F_0 \neq \mathsf{attacker}(x) \text{ for any variable } x \text{ and} \\ & \qquad F_0 \neq \mathsf{m\text{-}event}(p) \text{ for any pattern } p \end{cases} \tag{4.8}$$

By adapting the proof of Theorem 4.2, taking into account that we never select $\mathsf{m\text{-}event}(p)$, we obtain:

Theorem 4.5 (Correctness of saturate) *Let F be a closed fact. F is derivable from $\mathcal{R}_0 \cup \mathcal{F}_{\mathrm{me}}$ if and only if F is derivable from $\mathsf{saturate}(\mathcal{R}_0) \cup \mathcal{F}_{\mathrm{me}}$.*

Corollary 4.3 *Let $\mathsf{solve}_{\mathcal{R}_0}(\mathit{pred}(p_1, \ldots, p_n)) = \{H \Rightarrow \mathit{pred}(p_1', \ldots, p_n') \mid H \Rightarrow \mathit{pred}'(p_1', \ldots, p_n') \in \mathsf{saturate}(\mathcal{R}_0')\}$, where pred' is a new predicate and $\mathcal{R}_0' = \mathcal{R}_0 \cup \{\mathit{pred}(p_1, \ldots, p_n) \Rightarrow \mathit{pred}'(p_1, \ldots, p_n)\}$.*

The fact $\sigma\mathit{pred}(p_1, \ldots, p_n)$ is derivable from $\mathcal{R}_0 \cup \mathcal{F}_{\mathrm{me}}$ if and only if there exists a clause $H \Rightarrow \mathit{pred}(p_1', \ldots, p_n')$ in $\mathsf{solve}_{\mathcal{R}_0}(\mathit{pred}(p_1, \ldots, p_n))$ and a substitution σ' such that $\sigma'\mathit{pred}(p_1', \ldots, p_n') = \sigma\mathit{pred}(p_1, \ldots, p_n)$ and $\sigma' H$ is derivable from $\mathcal{R}_0' \cup \mathcal{F}_{\mathrm{me}}$.

Theorem 4.6 below states formally how we can interpret the result of resolution in terms of executed events. It is proved by combining Corollary 4.3 and Theorem 4.4.

Theorem 4.6 (Main theorem for events) *Let P_0 be a closed process and $P_0' = \mathrm{instr}(P_0)$. Let Q be an S-adversary and $Q' = \mathrm{instrAdv}(Q)$.*

Consider a trace $\mathcal{T} = S_0, E_0, \{P_0', Q'\} \rightarrow^* S', E', P'$, with $\mathit{fn}(P_0') \cup S \subseteq \mathsf{Dom}(E_0)$ and $E_0(a) = a[\,]$ for all $a \in \mathsf{Dom}(E_0)$.

If $\mathcal{T}$ executes an instance $\mathsf{event}(p')$ of $\mathsf{event}(p)$, then there exist a clause $H \Rightarrow C \in \mathsf{solve}_{\mathcal{R}_{P_0',S}}(\mathsf{event}(p))$ and a substitution σ such that $\mathsf{event}(p') = \sigma C$ and, for all $\mathsf{m\text{-}event}(p'')$ in σH, $\mathcal{T}$ executes $\mathsf{event}(p'')$.

Proof : Let $\mathcal{F}_{\mathrm{me}} = \{\mathsf{m\text{-}event}(p'') \mid \mathcal{T} \text{ executes } \mathsf{event}(p'')\}$. By Theorem 4.4, since $\mathcal{T}$ executes $\mathsf{event}(p')$, $\mathsf{event}(p')$ is derivable from $\mathcal{R}_{P'_0,S} \cup \mathcal{F}_{\mathrm{me}}$. By Corollary 4.3, there exist a clause $R = H \Rightarrow C$ in $\mathsf{solve}_{\mathcal{R}_{P'_0,S}}(\mathsf{event}(p))$ and a substitution σ such that $\sigma C = \mathsf{event}(p')$ and all elements of σH are derivable from $\mathcal{R}'_0 \cup \mathcal{F}_{\mathrm{me}}$. For all $\mathsf{m\text{-}event}(p)$ in σH, $\mathsf{m\text{-}event}(p)$ is derivable from $\mathcal{R}'_0 \cup \mathcal{F}_{\mathrm{me}}$. Since no clause in $\mathcal{R}'_0$ has a conclusion of the form $\mathsf{m\text{-}event}(\_)$, $\mathsf{m\text{-}event}(p) \in \mathcal{F}_{\mathrm{me}}$. Given the choice of $\mathcal{F}_{\mathrm{me}}$, this means that $\mathcal{T}$ executes $\mathsf{event}(p)$. □

4.6.4 Non-injective correspondences

Theorem 4.7 *Let P_0 be a closed process and $P'_0 = \mathrm{instr}(P_0)$. Let M_k ($k \in \{1,\ldots,l\}$) and M be terms. Let p_k, p be the patterns obtained by replacing names a with patterns $a[\,]$ in the terms M_k, M respectively. Assume that, for all clauses R in $\mathsf{solve}_{\mathcal{R}_{P'_0,S}}(\mathsf{event}(p))$, there exist σ' and H such that $R = H \wedge \mathsf{m\text{-}event}(\sigma' p_1) \wedge \ldots \wedge \mathsf{m\text{-}event}(\sigma' p_l) \Rightarrow \sigma' \mathsf{event}(p)$.*

Then P_0 satisfies the correspondence $\mathsf{event}(M) \rightsquigarrow \bigwedge_{k=1}^{l} \mathsf{event}(M_k)$ against S-adversaries.

Proof : Let Q be an S-adversary and $Q' = \mathrm{instrAdv}(Q)$. Consider a trace $\mathcal{T} = E_0, \{P_0, Q\} \rightarrow^* E', \mathcal{P}'$ with $fn(P'_0) \cup S \cup fn(M) \cup \bigcup_k fn(M_k) \subseteq E_0$ that executes $\sigma\mathsf{event}(M)$ for some substitution σ. Let $\mathcal{T}' = S_0, \{a \mapsto a[\,] \mid a \in E_0\}, \{P'_0, Q'\} \rightarrow^* S'_1, E'_1, \mathcal{P}'_1$ be the corresponding instrumented trace, which executes $\mathsf{event}(E'_1(\sigma M))$, instance of $\mathsf{event}(p)$. By Theorem 4.6, there exist $R = H' \Rightarrow C' \in \mathsf{solve}_{\mathcal{R}_{P'_0,S}}(\mathsf{event}(p))$ and σ'' such that $\mathsf{event}(E'_1(\sigma M)) = \sigma'' C'$ and for all $\mathsf{m\text{-}event}(p'')$ in $\sigma'' H'$, $\mathcal{T}'$ executes $\mathsf{event}(p'')$. All clauses R in $\mathsf{solve}_{\mathcal{R}_{P'_0,S}}(\mathsf{event}(p))$ are of the form $H \wedge \mathsf{m\text{-}event}(\sigma' p_1) \wedge \ldots \wedge \mathsf{m\text{-}event}(\sigma' p_l) \Rightarrow \sigma' \mathsf{event}(p)$ for some σ'. So, there exists σ' such that for all $k \in \{1,\ldots,l\}$, $\mathsf{m\text{-}event}(\sigma' p_k) \in H'$ and $C' = \sigma' \mathsf{event}(p)$. Hence $\mathsf{event}(E'_1(\sigma M)) = \sigma'' C' = \sigma'' \sigma' \mathsf{event}(p)$ and for all $k \in \{1,\ldots,l\}$, $\mathsf{m\text{-}event}(\sigma'' \sigma' p_k) \in \sigma'' H'$, so $\mathcal{T}'$ executes $\mathsf{event}(\sigma'' \sigma' p_k)$, hence $\mathcal{T}$ executes $\mathsf{event}(M'_k)$ for some M'_k such that $E'_1(M'_k) = \sigma'' \sigma' p_k = \sigma'' \sigma' E'_1(M_k)$. We also have $E'_1(\sigma M) = \sigma'' \sigma' p = \sigma'' \sigma' E'_1(M)$. Since E'_1 defines a suitable relation between terms and patterns (more precisely, using [?, Lemma 9]), there exists a substitution σ_0 such that for all $k \in \{1,\ldots,l\}$, $M'_k = \sigma_0 M_k$ and $\sigma M = \sigma_0 M$. So $\sigma M = \sigma_0 M$ and for all $k \in \{1,\ldots,l\}$, $\mathcal{T}$ executes $\mathsf{event}(M'_k) = \mathsf{event}(\sigma_0 M_k)$, so we have the result. □

Example 4.2 *For the process P of Section 4.3.4.2, letting $P' = \mathrm{instr}(P)$ and $S = \{c\}$, we have $\mathsf{solve}_{\mathcal{R}_{P',S}}(\mathsf{event}(e_B(x))) = \{\mathsf{m\text{-}event}(e_A(\mathsf{pk}(sk_B[\,]))) \Rightarrow \mathsf{event}(e_B(\mathsf{pk}(sk_B[\,])))\}$ so we have proved the correspondence $\mathsf{event}(e_B(x)) \rightsquigarrow \mathsf{event}(e_A(x))$.*

4.6.5 Sketch for injective correspondences

Let P_0 be a closed process and $P'_0 = \mathrm{instr}'(P_0)$. We adapt the generation of clauses as follows: the set of clauses $\mathcal{R}'_{P'_0,S}$ is defined as $\mathcal{R}_{P'_0,S}$ except that

$$\begin{aligned}
&[\![\mathsf{out}(M,N).P]\!]\rho H = [\![P]\!]\rho H \cup \{H\{\rho_{|V_o \cup V_s}/\Box\} \Rightarrow \mathsf{mess}(\rho(M),\rho(N))\}\\
&[\![!^i P]\!]\rho H = [\![P]\!](\rho[i \mapsto i])(H\{\rho_{|V_o \cup V_s}/\Box\})\\
&[\![\mathsf{event}(M,i).P]\!]\rho H = [\![P]\!]\rho(H \wedge \mathsf{m\text{-}event}(\rho(M),\Box)) \cup \{H\{\rho_{|V_o \cup V_s}/\Box\} \Rightarrow \mathsf{event}(\rho(M),i)\}
\end{aligned}$$

where $\Box$ is a special variable. The predicate event has as additional argument the session identifier in which the event is executed. The predicate $\mathsf{m\text{-}event}$ has as additional argument an environment ρ that contains the variables that have a single value for each execution of the considered event $\mathsf{event}(M,i)$ and that are defined when we generate the clause. These variables are the variables that are defined above the first replication that follows $\mathsf{event}(M,i)$ and above the output or event that generates the clause. The special variable $\Box$ is a placeholder for this environment ρ.

Proposition 4.2 (Injective correspondences) *Let P_0 be a closed process, $P'_0 = \mathrm{instr}'(P_0)$. We assume that, in P_0, all events are of the form $\mathsf{event}(f(M_1, \dots, M_n))$ and that different occurrences of* event *have different root function symbols.*

Let M_k ($k \in \{1, \dots, l\}$) and M be terms. Let p_k, p be the patterns obtained by replacing names a with patterns $a[\,]$ in the terms M_k, M respectively.

We also assume that p is of the form $f(\dots)$ for some function symbol f and that, for all k, $p_k = f_k(\dots)$ for some function symbol f_k.

Let $\mathsf{solve}_{\mathcal{R}'_{P'_0,S}}(\mathsf{event}(p, i)) = \{R_r \mid r \in \{1, \dots, n\}\}$. Assume that there exist x_k, i_r, and ρ_{rk} ($r \in \{1, \dots, n\}$, $k \in \{1, \dots, l\}$) such that

- *For all $r \in \{1, \dots, n\}$, there exist H and σ such that $R_r = H \wedge \mathsf{m\text{-}event}(\sigma p_1, \rho_{r1}) \wedge \dots \wedge \mathsf{m\text{-}event}(\sigma p_l, \rho_{rl}) \Rightarrow \mathsf{event}(\sigma p, i_r)$.*
- *For all r and r' in $\{1, \dots, n\}$, for all $k \in \{1, \dots, l\}$, $\rho_{rk}(x_k)\{\lambda/i_r\}$ does not unify with $\rho_{r'k}(x_k)\{\lambda'/i_{r'}\}$ when $\lambda \neq \lambda'$.*

Then P_0 satisfies the correspondence

$$\mathsf{event}(M) \rightsquigarrow \bigwedge_{k=1}^{l} \mathsf{inj}\ \mathsf{event}(M_k)$$

against S-adversaries.

By Theorem 4.7, after deleting session identifiers and environments, the first item shows that P'_0 satisfies the correspondence

$$\mathsf{event}(M) \rightsquigarrow \bigwedge_{k=1}^{l} \mathsf{event}(M_k) \tag{4.9}$$

The environments and session identifiers as well as the second item serve in proving injectivity. Denote by $\_$ an unknown term. If two instances of $\mathsf{event}(p, i)$ are executed in P'_0, by the first item, they are instances of $\mathsf{event}(\sigma_r p, i_r)$ for some r, so they are $\mathsf{event}(\sigma'_1\sigma_{r_1}p, \sigma'_1 i_{r_1})$ and $\mathsf{event}(\sigma'_2\sigma_{r_2}p, \sigma'_2 i_{r_2})$ for some σ'_1 and σ'_2. Furthermore, there is only one occurrence of $\mathsf{event}(f(\dots), i)$ in P'_0, so the event $\mathsf{event}(f(\dots), i)$ can be executed at most once for each value of the session identifier i, so $\sigma'_1 i_{r_1} \neq \sigma'_2 i_{r_2}$. Then, by the first item, corresponding events $\mathsf{event}(\sigma'_1\sigma_{r_1}p_k, \_)$ and $\mathsf{event}(\sigma'_2\sigma_{r_2}p_k, \_)$ have been executed, with associated environments $\sigma'_1\rho_{r_1k}$ and $\sigma'_2\rho_{r_2k}$. By the second item, $\rho_{r_1k}(x_k)\{\lambda_1/i_{r_1}\}$ does not unify with $\rho_{r_2k}(x_k)\{\lambda_2/i_{r_2}\}$ for different values $\lambda_1 = \sigma'_1 i_{r_1}$ and $\lambda_2 = \sigma'_2 i_{r_2}$ of the session identifier. (In this condition, r_1 can be equal to r_2, and when $r_1 = r_2 = r$, the condition simply means that i_r occurs in ρ_{rk}.) So $\sigma'_1\rho_{r_1k}(x_k) \neq \sigma'_2\rho_{r_2k}(x_k)$, so the events $\mathsf{event}(\sigma'_1\sigma_{r_1}p_k, \_)$ and $\mathsf{event}(\sigma'_2\sigma_{jr_2}p_k, \_)$ are distinct, which shows injectivity. This point is very similar to the fact that injective agreement is implied by non-injective agreement when the parameters of events contain nonces generated by the agent to whom authentication is being made, because the event can be executed at most once for each value of the nonce. (The session identifier i_r in our theorem plays the role of the nonce.) [Andrew Gordon, personal communication].

For the detailed proof of this result and more general results on correspondences, we refer to [**?**], in particular Section 7.2.

Example 4.3 *For the process P of Section 4.3.4.2, letting $P' = \mathrm{instr}(P)$ and $S = \{c\}$, we obtain $\mathsf{solve}_{\mathcal{R}'_{P',S}}(\mathsf{event}(e_B(x, i_B))) = \{\mathsf{m\text{-}event}(e_A(\mathsf{pk}(sk_B[\,])), \{x\_b \mapsto b[\mathsf{pk}(sk_A[\,]), i_B], x\_pk_B \mapsto \mathsf{pk}(sk_B[\,])\}) \Rightarrow \mathsf{event}(e_B(\mathsf{pk}(sk_B[\,])), i_B)\}$, so we have proved the correspondence $\mathsf{event}(e_B(x)) \rightsquigarrow \mathsf{inj}\ \mathsf{event}(e_A(x))$ because i_B occurs in $\rho = \{x\_b \mapsto b[\mathsf{pk}(sk_A[\,]), i_B], x\_pk_B \mapsto \mathsf{pk}(sk_B[\,])\}$. (If distinct events e_B are executed, they have different values of i_B, so they correspond to different values of ρ, and to distinct events e_A.)*

4.7 Extension to the Proof of Observational Equivalences

We now present the extension of ProVerif to the proof of observational equivalences, which was introduced in [**?**]. This section is adapted from that paper.

We focus on proving equivalences $P \approx Q$ in which P and Q are two processes that differ only in the choice of some terms. These equivalences arise often in applications. ProVerif proves these equivalences by proving a stronger equivalence that can be expressed as a property on the traces of a process that represents P and Q at the same time.

We first introduce a minor extension of the syntax of processes:

$D ::=$	term evaluations
$\quad M$	term
$\quad f(D_1, \dots, D_n)$	constructor application
$\quad g(D_1, \dots, D_n)$	destructor evaluation
$P, Q ::=$	processes
$\quad \dots$	(as in Figure 4.2)
$\quad \mathsf{let}\ x = D\ \mathsf{in}\ P\ \mathsf{else}\ Q$	term evaluation

This extension allows to evaluate several destructors in one step. This can help the proof of equivalences succeed more often: if a destructor g_1 fails in P but succeeds in Q, and a subsequent destructor g_2 fails in Q, the proof of equivalence will succeed if we evaluate g_1 and g_2 in one step (the term evaluation fails in both P and Q), but fail if we evaluate them separately, even if the processes P and Q are actually equivalent.

The semantics is defined as in 4.3, except that (Red Destr 1) and (Red Destr 2) are replaced with (Red Fun 1) and (Red Fun 2) defined below. They use as auxiliary relation the term evaluation relation $D \Downarrow M$, which means that D evaluates to the term M, and is formally defined as follows:

$M \Downarrow M$
$f(D_1, \dots, D_n) \Downarrow f(N_1, \dots, N_n)$ if for all i, $D_i \Downarrow N_i$
$g(D_1, \dots, D_n) \Downarrow \sigma N$ if $g(N_1, \dots, N_n) \rightarrow N \in \mathrm{def}(g)$ and, for all i, $D_i \Downarrow \sigma N_i$

$\mathsf{let}\ x = D\ \mathsf{in}\ P\ \mathsf{else}\ Q \rightarrow P\{M/x\}$
  if $D \Downarrow M$ (Red Fun 1)
$\mathsf{let}\ x = D\ \mathsf{in}\ P\ \mathsf{else}\ Q \rightarrow Q$
  if there is no M such that $D \Downarrow M$ (Red Fun 2)

Next we introduce a new calculus that can represent pairs of processes that have the same structure and differ only by the terms and term evaluations that they contain. We call such a pair of processes a *biprocess*. The grammar for the calculus is a simple extension of the grammar of Figure 4.2, with additional cases so that $\mathsf{diff}[M, M']$ is a term and $\mathsf{diff}[D, D']$ is a term evaluation. We also extend the definition of contexts to permit the use of diff, and sometimes refer to contexts without diff as plain contexts.

Given a biprocess P, we define two processes $\mathsf{fst}(P)$ and $\mathsf{snd}(P)$, as follows: $\mathsf{fst}(P)$ is obtained by replacing all occurrences of $\mathsf{diff}[M, M']$ with M and $\mathsf{diff}[D, D']$ with D in P, and similarly, $\mathsf{snd}(P)$ is obtained by replacing $\mathsf{diff}[M, M']$ with M' and $\mathsf{diff}[D, D']$ with D' in P. We define $\mathsf{fst}(D)$, $\mathsf{fst}(M)$, $\mathsf{snd}(D)$, and $\mathsf{snd}(M)$ similarly. Our goal is to show that the processes $\mathsf{fst}(P)$ and $\mathsf{snd}(P)$ are observationally equivalent:

Definition 4.16 *Let P be a closed biprocess. We say that P* satisfies observational equivalence *when* $\mathsf{fst}(P) \approx \mathsf{snd}(P)$.

The semantics for biprocesses is defined as in Figure 4.3 with generalized rules (Red I/O), (Red Fun 1), and (Red Fun 2) given in Figure 4.10. Reductions for biprocesses bundle those

$$
\begin{array}{ll}
\mathsf{out}(N,M).Q \parallel \mathsf{in}(N',x).P \;\rightarrow\; Q \parallel P\{M/x\} & \text{(Red I/O)}\\
\quad \text{if } \mathsf{fst}(N) = \mathsf{fst}(N') \text{ and } \mathsf{snd}(N) = \mathsf{snd}(N') & \\
\mathsf{let}\ x = D\ \mathsf{in}\ P\ \mathsf{else}\ Q \rightarrow P\{\mathsf{diff}[M_1, M_2]/x\} & \text{(Red Fun 1)}\\
\quad \text{if } \mathsf{fst}(D) \Downarrow M_1 \text{ and } \mathsf{snd}(D) \Downarrow M_2 & \\
\mathsf{let}\ x = D\ \mathsf{in}\ P\ \mathsf{else}\ Q \rightarrow Q & \text{(Red Fun 2)}\\
\quad \text{if there is no } M_1 \text{ such that } \mathsf{fst}(D) \Downarrow M_1 \text{ and} & \\
\quad \text{there is no } M_2 \text{ such that } \mathsf{snd}(D) \Downarrow M_2 &
\end{array}
$$

Figure 4.10: Generalized rules for biprocesses

for processes: if $P \rightarrow Q$ then $\mathsf{fst}(P) \rightarrow \mathsf{fst}(Q)$ and $\mathsf{snd}(P) \rightarrow \mathsf{snd}(Q)$. Conversely, however, reductions in $\mathsf{fst}(P)$ and $\mathsf{snd}(P)$ need not correspond to any biprocess reduction, in particular when they do not match up. Our first theorem shows that the processes are equivalent when this does not happen.

Definition 4.17 *We say that the biprocess P is* uniform *when $\mathsf{fst}(P) \rightarrow Q_1$ implies that $P \rightarrow Q$ for some biprocess Q with $\mathsf{fst}(Q) \equiv Q_1$, and symmetrically for $\mathsf{snd}(P) \rightarrow Q_2$.*

Theorem 4.8 *Let P_0 be a closed biprocess. If, for all plain evaluation contexts C and reductions $C[P_0] \rightarrow^* P$, the biprocess P is uniform, then P_0 satisfies observational equivalence.*

Proof : Let P be a closed biprocess such that $C[P] \rightarrow^*\equiv Q$ always yields a uniform biprocess Q, and consider the relation

$$\mathcal{R} = \{(Q_1, Q_2) \mid \exists Q, C \text{ such that } Q_1 \equiv \mathsf{fst}(Q), Q_2 \equiv \mathsf{snd}(Q), \text{ and } C[P] \rightarrow^*\equiv Q\}$$

In particular, we have $\mathsf{fst}(P)\ \mathcal{R}\ \mathsf{snd}(P)$, so we can show that P satisfies observational equivalence by establishing that the relation $\mathcal{R}' = \mathcal{R} \cup \mathcal{R}^{-1}$ meets the three conditions of Definition 4.10. By symmetry, we focus on $\mathcal{R}$. Assume $Q_1\ \mathcal{R}\ Q_2$. We have $Q_1 \equiv \mathsf{fst}(Q)$, $Q_2 \equiv \mathsf{snd}(Q)$, and $C[P] \rightarrow^*\equiv Q$ for some C and Q. We show that

1. if $P \downarrow_M$ then $Q \downarrow_M$;
2. if $P \rightarrow P'$ then $Q \rightarrow Q'$ and $P'\ \mathcal{R}\ Q'$ for some Q';
3. $C[P]\ \mathcal{R}\ C[Q]$ for all evaluation contexts C.

It is then easy to conclude the three conditions of Definition 4.10.

1. Assume $Q_1 \downarrow_M$, and let $T_M = \mathsf{in}(M,x).\mathsf{out}(c,c)$ for some fresh name c. As usual in the pi calculus, the predicate $\_\downarrow_M$ tests the ability to send any message on M, hence for any plain process Q_i, we have $Q_i \downarrow_M$ if and only if $Q_i \parallel T_M \rightarrow R_i \parallel \mathsf{out}(c,c)$ for some R_i.

   Here, we have $Q_1 \parallel T_M \rightarrow R_1 \parallel \mathsf{out}(c,c)$ for some R_1. Since $Q_1 \equiv \mathsf{fst}(Q)$, $\mathsf{fst}(Q) \parallel T_M \rightarrow R_1 \parallel \mathsf{out}(c,c)$. The reductions $C[P] \rightarrow^*\equiv Q$ imply $C[P] \parallel T_M \rightarrow^*\equiv Q \parallel T_M$. By hypothesis (with the context $C[\_] \parallel T_M$), $Q \parallel T_M$ is uniform, hence $Q \parallel T_M \rightarrow Q'$ for some Q' with $\mathsf{fst}(Q') \equiv R_1 \parallel \mathsf{out}(c,c)$. Since c does not occur anywhere in Q, by case analysis on this reduction step with our semantics for biprocesses we obtain $Q' \equiv R \parallel \mathsf{out}(c,c)$ for some biprocess R. Thus, we obtain $\mathsf{snd}(Q) \parallel T_M \rightarrow \mathsf{snd}(R) \parallel \mathsf{out}(c,c)$, so $Q_2 \parallel T_M \rightarrow \mathsf{snd}(R) \parallel \mathsf{out}(c,c)$, and finally $\mathsf{snd}(Q) \downarrow_M$.

2. If $Q_1 \rightarrow Q_1'$, then $\mathsf{fst}(Q) \rightarrow Q_1'$, so by uniformity, we have $Q \rightarrow Q'$ with $\mathsf{fst}(Q') \equiv Q_1'$. Thus, $C[P] \rightarrow^*\equiv\rightarrow Q'$ and, by definition of $\mathcal{R}$, we obtain $Q_1'\ \mathcal{R}\ \mathsf{snd}(Q')$. Finally, by definition of the semantics of biprocesses, $Q \rightarrow Q'$ implies $\mathsf{snd}(Q) \rightarrow \mathsf{snd}(Q')$, so $Q_2 \rightarrow \mathsf{snd}(Q')$.

3. Let C' be a plain evaluation context. By definition of the semantics of biprocesses, $C[P] \rightarrow^* \equiv Q$ always implies $C'[C[P]] \rightarrow^* \equiv C'[Q]$. Moreover, $C'[Q_1] \equiv C'[\mathsf{fst}(Q)] = \mathsf{fst}(C'[Q])$ and $C'[Q_2] \equiv C'[\mathsf{snd}(Q)] = \mathsf{snd}(C'[Q])$, hence $C'[Q_1] \mathrel{\mathcal{R}} C'[Q_2]$. □

Our plan is to establish the hypothesis of Theorem 4.8 by automatically verifying that all the biprocesses P in question meet conditions that imply uniformity. The next corollary details those conditions, which guarantee that a communication and an evaluation, respectively, succeed in $\mathsf{fst}(P)$ if and only if they succeed in $\mathsf{snd}(P)$:

Corollary 4.4 *Let P_0 be a closed biprocess. Suppose that, for all plain evaluation contexts C, all evaluation contexts C', and all reductions $C[P_0] \rightarrow^* P$,*

1. *if $P \equiv C'[\mathsf{out}(N, M).Q \parallel \mathsf{in}(N', x).R]$, then $\mathsf{fst}(N) = \mathsf{fst}(N')$ if and only if $\mathsf{snd}(N) = \mathsf{snd}(N')$,*
2. *if $P \equiv C'[\mathsf{let}\ x = D\ \mathsf{in}\ Q\ \mathsf{else}\ R]$, then there exists M_1 such that $\mathsf{fst}(D) \Downarrow M_1$ if and only if there exists M_2 such that $\mathsf{snd}(D) \Downarrow M_2$.*

Then P_0 satisfies observational equivalence.

Proof : We show that P is uniform, then we conclude by Theorem 4.8. Let us show that, if $\mathsf{fst}(P) \rightarrow P'_1$ then there exists a biprocess P' such that $P \rightarrow P'$ and $\mathsf{fst}(P') \equiv P'_1$. The case for $\mathsf{snd}(P) \rightarrow P'_2$ is symmetric.

By induction on the derivation of $\mathsf{fst}(P) \rightarrow P'_1$, we first show that there exist C, Q, and Q'_1 such that $P \equiv C[Q]$, $P'_1 \equiv \mathsf{fst}(C)[Q'_1]$, and $\mathsf{fst}(Q) \rightarrow Q'_1$ using one of the four process rules (Red I/O), (Red Fun 1), (Red Fun 2), or (Red Repl): every step in this derivation trivially commutes with fst, except for structural steps that involve a parallel composition and a restriction, in case $a \in fn(P)$ but $a \notin fn(\mathsf{fst}(P))$. In that case, we use a preliminary renaming from a to some fresh $a' \notin fn(P)$.

For each of these four rules, relying on a hypothesis of Corollary 4.4, we find Q' such that $\mathsf{fst}(Q') = Q'_1$ and $Q \rightarrow Q'$ using the corresponding biprocess rule:

(Red I/O): We have $Q = \mathsf{out}(N, M).R \parallel \mathsf{in}(N', x).R'$ with $\mathsf{fst}(N) = \mathsf{fst}(N')$ and $Q'_1 = \mathsf{fst}(R) \parallel \mathsf{fst}(R')\{\mathsf{fst}(M)/x\}$. For $Q' = R \parallel R'\{M/x\}$, we have $\mathsf{fst}(Q') = Q'_1$ and, by hypothesis 1, $\mathsf{snd}(N) = \mathsf{snd}(N')$, hence $Q \rightarrow Q'$.

(Red Fun 1): We have $Q = \mathsf{let}\ x = D\ \mathsf{in}\ R\ \mathsf{else}\ R'$ with $\mathsf{fst}(D) \Downarrow M_1$ and $Q'_1 = \mathsf{fst}(R)\{M_1/x\}$. By hypothesis 2, $\mathsf{snd}(D) \Downarrow M_2$ for some M_2. We take $Q' = R\{\mathsf{diff}[M_1, M_2]\}$, so that $\mathsf{fst}(Q') = Q'_1$ and $Q \rightarrow Q'$.

(Red Fun 2): We have $Q = \mathsf{let}\ x = D\ \mathsf{in}\ R\ \mathsf{else}\ R'$ with no M_1 such that $\mathsf{fst}(D) \Downarrow M_1$ and $Q'_1 = \mathsf{fst}(R')$. By hypothesis 2, there is no M_2 such that $\mathsf{snd}(D) \Downarrow M_2$. We obtain $Q \rightarrow Q'$ for $Q' = R'$.

(Red Repl): We have $Q = !R$ and $Q'_1 = \mathsf{fst}(R) \parallel !\mathsf{fst}(R)$. We take $Q' = R \parallel !R$, so that $\mathsf{fst}(Q') = Q'_1$ and $Q \rightarrow Q'$.

To conclude, we take the biprocess $P' = C[Q']$ and the reduction $P \rightarrow P'$. □

Thus, we have a sufficient condition for observational equivalence of biprocesses. This condition is essentially a reachability condition on biprocesses. We adapt existing techniques for reasoning about processes in order to prove this condition. The condition is however not necessary: if $P \approx Q$, then $\mathsf{if}\ \mathsf{diff}[\mathsf{true}, \mathsf{false}] = \mathsf{true}\ \mathsf{then}\ P\ \mathsf{else}\ Q$ satisfies observational equivalence, but Theorem 4.8 and Corollary 4.4 will not enable us to prove this fact.

Example 4.4 *Probabilistic encryption is a variant of public-key encryption that further protects the secrecy of the plaintext by embedding some additional, fresh value in each encryption. It can be modeled using three functions for public-key decryption, public-key encryption, and public-key derivation, linked by the equation*

$$\mathsf{adec}(\mathsf{aenc}(x, \mathsf{pk}(y), z), y) = x$$

where z is the additional random parameter for the encryption. A key property of probabilistic encryption is that, without knowledge of the decryption key, ciphertexts appear to be unrelated to the plaintexts, even if the attacker knows the plaintexts and the encryption key. A strong version of this property is that the ciphertexts cannot be distinguished from freshly generated random values. Formally, we state that

$$\mathsf{new}\ s.(\mathsf{out}(c, \mathsf{pk}(s)) \parallel !\mathsf{in}(c', x).\mathsf{new}\ a.\mathsf{out}(c, \mathsf{diff}[\mathsf{aenc}(x, \mathsf{pk}(s), a), a]))$$

satisfies equivalence. This biprocess performs a first output to reveal the public key $\mathsf{pk}(s)$ (but not s!), then repeatedly inputs a term x from the environment and either outputs its encryption under $\mathsf{pk}(s)$ or outputs a fresh, unrelated name. Thus, a single biprocess represents the family of static equivalences that relate a series of probabilistic encryptions for any series of plaintext to a series of fresh, independent names. (Formally, each such equivalence can be obtained as a corollary of this biprocess equivalence, by applying the congruence property of equivalence for the particular context that sends the plaintexts of values on channel c' and reads the encryption key and encryptions on channel c.)

As usual, we can prove the reachability properties that occur as assumptions of Corollary 4.4 by translating the process into Horn clauses. Clauses are built from the following predicates:

$F ::=$	facts
$\mathsf{attacker}'(p, p')$	attacker knowledge
$\mathsf{mess}'(p_1, p_2, p'_1, p'_2)$	output message p_2 on p_1 (resp. p'_2 on p'_1)
$\mathsf{input}'(p, p')$	input on p (resp. p')
$\mathsf{nounif}(p, p')$	impossible unification
bad	bad

Informally, $\mathsf{attacker}'(p, p')$ means that the attacker may obtain p in $\mathsf{fst}(P)$ and p' in $\mathsf{snd}(P)$ by the same operations; $\mathsf{mess}'(p_1, p_2, p'_1, p'_2)$ means that message p_2 may appear on channel p_1 in $\mathsf{fst}(P)$ and that message p'_2 may appear on channel p'_1 in $\mathsf{snd}(P)$ after the same reductions; $\mathsf{input}'(p, p')$ means that an input may be executed on channel p in $\mathsf{fst}(P)$ and on channel p' in $\mathsf{snd}(P)$, thus enabling the attacker to infer whether p (resp. p') is equal to another channel used for output; $\mathsf{nounif}(p, p')$ means that p and p' cannot be unified by substituting elements of $GVar$ with patterns; finally, bad serves in detecting violations of observational equivalence: when bad is not derivable, we have observational equivalence.

An evident difference with respect to previous translations from processes to clauses is that predicates have twice as many arguments: we use the binary predicate $\mathsf{attacker}'$ instead of the unary one $\mathsf{attacker}$ and the 4-ary predicate mess' instead of the binary one mess. This extension allows us to represent information for both variants of a biprocess.

The predicate nounif is not defined by clauses, but by special simplification steps in the solver. We omit the details of the generation of the clauses and of the resolution algorithm, which can be found in [**?**].

Example 4.5 *The biprocess of Example 4.4 yields the clauses:*

$$\mathsf{mess}'(c, \mathsf{pk}(s), c, \mathsf{pk}(s))$$
$$\mathsf{mess}'(c', x, c', x') \Rightarrow \mathsf{mess}'(c, \mathsf{aenc}(x, \mathsf{pk}(s), a[i, x]), c, a[i, x'])$$

The first clause corresponds to the output of the public key $\mathsf{pk}(s)$*. The second clause corresponds to the other output: if a message* x *(resp.* x'*) is received on channel* c'*, then the message* $\mathsf{aenc}(x, \mathsf{pk}(s), a[i, x])$ *in the first variant (resp.* $a[i, x']$ *in the second variant) is sent on channel* c*. The fresh name* a *is encoded as a pattern* $a[i, x]$*.*

In a similar way as for previous security properties, we can prove:

Theorem 4.9 *If* bad *is not a logical consequence of the clauses, then* P_0 *satisfies observational equivalence.*

In order to determine whether bad is a logical consequence of the clauses, we use again an algorithm based on resolution with free selection, adapting the algorithm presented previously, in particular with specific simplification steps for nounif.

4.7.1 Weak Secrets

A *weak secret* represents a secret value with low entropy, such as a human-memorizable password. Protocols that rely on weak secrets are often subject to guessing attacks, whereby an attacker guesses a weak secret, perhaps using a dictionary, and verifies its guess. The guess verification may rely on interaction with protocol participants or on computations on intercepted messages (e.g., [**?**, **?**, **?**]). With some care in protocol design, however, those attacks can be prevented:

- On-line guessing attacks can be mitigated by limiting the number of retries that participants allow. An attacker that repeatedly attempts to guess the weak secret should be eventually detected and stopped if it tries to verify its guesses by interacting with other participants.
- Off-line guessing attacks can be prevented by making sure that, even if the attacker (systematically) guesses the weak secret, it cannot verify whether its guess is correct by computing on intercepted traffic.

Off-line guessing attacks can be explained and modeled in terms of a 2-phase scenario. In phase 0, on-line attacks are possible, but the weak secret is otherwise unguessable. In phase 1, the attacker obtains a possible value for the weak secret (intuitively, by guessing it). The absence of off-line attacks is characterized by an equivalence: the attacker cannot distinguish the weak secret used in phase 0 from an unrelated fresh value.

In our calculus, we arrive at the following definition:

Definition 4.18 (Weak secrecy) *Let* P *be a closed process with no phase prefix. We say that* P *prevents off-line attacks against* w *when* $\mathsf{new}\ w.(\mathsf{phase}\ 0.P \parallel \mathsf{phase}\ 1.\mathsf{new}\ w'.\mathsf{out}(c, \mathsf{diff}[w, w']))$ *satisfies observational equivalence.*

This definition is in line with the work of Cohen, Corin et al., Delaune and Jacquemard, Drielsma et al., and Lowe [**?**, **?**, **?**, **?**, **?**, **?**]. Lowe uses the model-checker FDR to handle a bounded number of sessions, while Delaune and Jacquemard give a decision procedure in this case. Corin et al. give a definition based on equivalence like ours, but do not consider the first, active phase; they analyze only one session.

As a first example, assume that a principal attempts to prove knowledge of a shared password w to a trusted server by sending a hash of this password encrypted under the server's public key. (For simplicity, the protocol does not aim to provide freshness guarantees, so anyone may replay this proof.) Omitting the code for the server, a first protocol may be written:

$$P = \mathsf{new}\ s.\mathsf{out}(c, pk(s)).\mathsf{out}(c, \mathsf{aenc}(h(w), pk(s)))$$

The first output reveals the public key of the server; the second output communicates the proof of knowledge of w. This protocol does not prevent off-line attacks against w. ProVerif finds an attack that corresponds to the following adversary:

$$\begin{aligned} A = {} & \mathsf{phase}\ 0.\mathsf{in}(c, pk).\mathsf{in}(c, e). \\ & \mathsf{phase}\ 1.\mathsf{in}(c, w).\mathsf{if}\ e = \mathsf{aenc}(h(w), pk)\ \mathsf{then}\ \mathsf{out}(\mathrm{Guessed}, ()) \end{aligned}$$

A corrected protocol uses probabilistic encryption (see Example 4.4):

$$P = \mathsf{new}\ s, a.\mathsf{out}(c, pk(s)).\mathsf{out}(c, \mathsf{aenc}(h(w), pk(s), a))$$

ProVerif automatically produces a proof for this corrected protocol.

4.7.2 Authenticity

Abadi and Gordon [?] use equivalences for characterizing authenticity properties, and treat a variant of the Wide-Mouth-Frog protocol as an example. Essentially, authenticity is defined as an equivalence between the protocol and a specification. The technique presented here automatically proves authenticity for the one-session version of this protocol [?, Section 3.2.2], thereby eliminating the need for a laborious manual proof. (Authenticity properties are often formulated as correspondence assertions on behaviors, rather than as equivalences. Those assertions can also be verified with ProVerif, as shown in Section 4.6.)

4.8 Applications

The automatic protocol verifier ProVerif is available at `http://proverif.inria.fr/`. It was successfully applied to many protocols of the literature, to prove secrecy and authentication properties: flawed and corrected versions of the Needham-Schroeder public-key [?, ?] and shared-key [?, ?, ?], Woo-Lam public-key [?, ?] and shared-key [?, ?, ?, ?, ?], Denning-Sacco [?, ?], Yahalom [?], Otway-Rees [?, ?, ?], and Skeme [?] protocols. No false attack occurred in these tests and the only non-termination cases were some flawed versions of the Woo-Lam shared-key protocol. The other protocols were verified in less than one second each on a Pentium M 1.8 GHz [?].

Moreover, ProVerif was also used in more substantial case studies:

- Abadi and Blanchet [?] applied it to the verification of a certified email protocol [?]. They use correspondence properties to prove that the receiver receives the message if and only if the sender has a receipt for the message. (They use simple manual arguments to take into account that the reception of sent messages is guaranteed.) One of the tested versions includes the SSH transport layer in order to establish a secure channel. (Total runtime: 6 min on a Pentium M 1.8 GHz.)

- Abadi, Blanchet, and Fournet [?] studied the JFK protocol (*Just Fast Keying*) [?], which was one of the candidates to the replacement of IKE as key exchange protocol in IPSec. They combined manual proofs and ProVerif to prove correspondences and equivalences. (Total runtime: 3 min on a Pentium M 1.8 GHz.)

- Blanchet and Chaudhuri [?] studied the secure filesystem Plutus [?] with ProVerif, which allowed them to discover and fix weaknesses of the initial system.

- Bhargavan et al. [?, ?, ?] use it to build the Web services verification tool TulaFale: Web services are protocols that send XML messages; TulaFale translates them into the input format of ProVerif and uses ProVerif to prove the desired security properties.

- Bhargavan et al. [?, ?, ?] use ProVerif for verifying implementations of protocols in F# (a functional language of the Microsoft .NET environment): a subset of F# large enough for expressing security protocols is translated into the input format of ProVerif. The TLS protocol, in particular, was studied using this technique [?].

- Canetti and Herzog [?] use ProVerif for verifying protocols in the computational model: they show that, for a restricted class of protocols that use only public-key encryption, a proof in the Dolev-Yao model implies security in the computational model, in the universal composability framework. Authentication is verified using correspondences, while secrecy of keys corresponds to strong secrecy.

- ProVerif was also used for verifying a certified email web service [?], a certified mailing-list protocol [?], e-voting protocols [?, ?], the ad-hoc routing protocol ARAN (*Authenticated Routing for Adhoc Networks*) [?], and zero-knowledge protocols [?].

Finally, Goubault-Larrecq and Parrennes [?] also use the Horn clause method for analyzing implementations of protocols written in C. However, they translate protocols into clauses of the $\mathcal{H}_1$ class and use the $\mathcal{H}_1$ prover by Goubault-Larrecq [?] rather than ProVerif to prove secrecy properties of the protocol.

4.9 Further Readings

The verifier ProVerif has been presented in several research papers. The translation from the spi calculus to Horn clauses for secrecy and its relation with typing was presented in [?]. Its extension to correspondence assertions was presented in [?]. Its extension to the proof of equivalences, the treatment of equational theories, and the extension to scenarios with several phases (named stages in [?]) was presented in [?]. ProVerif can also reconstruct attacks against protocols [?], although we did not detail this point.

4.10 Exercises

Exercice 25
We assume the formal model of cryptography (the so-called Dolev-Yao model) and we consider the following protocol by Otway and Rees (1987).

Message 1. $A \to B : M, A, B, \{N_a, M, A, B\}_{K_{as}}$ $\quad M, N_a$ fresh
Message 2. $B \to S : M, A, B, \{N_a, M, A, B\}_{K_{as}}, \{N_b, M, A, B\}_{K_{bs}}$ $\quad N_b$ fresh
Message 3. $S \to B : M, \{N_a, K_{ab}\}_{K_{as}}, \{N_b, K_{ab}\}_{K_{bs}}$ $\quad K_{ab}$ fresh
Message 4. $B \to A : M, \{N_a, K_{ab}\}_{K_{as}}$
Message 5. $A \to B : \{s\}_{K_{ab}}$

We recall that the notation above describes the messages exchanged in a correct execution of the protocol. Three participants are involved in this protocol: A and B aim at establishing a session, using an authentication server S. In the protocol, A and B are the identities of A and B; K_{as} is a long-term key shared between A and S; similarly, K_{bs} is a key shared between B and S; M, N_a, and N_b are nonces; K_{ab} is the session key between A and B, established by the protocol (it is chosen by the server S); s is a message that A wants to send to B, with the guarantee that the adversary will not be able to obtain it. The notation $\{M\}_K$ represents the shared-key encryption of the message M under the key K.

One shall consider that A is willing to execute the protocol with B, but also with other participants, which are included in the adversary. However, Message 5 is sent by A only if its interlocutor is B. (This message is used in order to test that the key K_{ab} shared between two honest participants A and B is really secret.) B is also willing to execute the protocol not only with A, but also with other participants. The server S is willing to interact with any participants. The server S has a table that relates the identity of each participant X (A, B, ...) to the secret key K_{xs} shared between X and S.

1. Explain the actions of the three participants of the protocol: one shall explain how each participant verifies the messages that he receives and how he constructs the messages that he sends.

2. Code this protocol in the variant of the spi calculus presented during the course. (One shall give both the process and the definition of the destructors used in the protocol.)

   Indication: In order to represent the table that S has and that relates the identity of each participant X with the secret key K_{xs} shared between X and S, one may consider that the identity X is coded by the term $\mathit{host}(K_{xs})$, and that the server uses a function getkey such that $\mathit{getkey}(X) = \mathit{getkey}(\mathit{host}(K_{xs})) = K_{xs}$ in order to reconstruct the key K_{xs} from the identity X.

3. Give the Horn clauses associated to this protocol, in order to show that s is secret. Give a succinct intuitive explanation of each of these clauses (or groups of clauses).

   (Several variants are possible.)

Exercice 26
Similar exercise with the following protocol by Carlsen:

Message 1. $A \to B : A, N_a$ N_a fresh
Message 2. $B \to S : A, N_a, B, N_b$ N_b fresh
Message 3. $S \to B : \{K_{ab}, N_a, A\}_{K_{bs}}, \{N_a, B, K_{ab}\}_{K_{as}}$ K_{ab} fresh
Message 4. $B \to A : \{N_a, B, K_{ab}\}_{K_{as}}, \{N_a\}_{K_{ab}}, N'_b$ N'_b fresh
Message 5. $A \to B : \{N'_b\}_{K_{ab}}$
Message 6. $A \to B : \{s\}_{K_{ab}}$

In this protocol, A and B are the identities of A and B; K_{as} is a long-term key shared between A and S; similarly, K_{bs} is a key shared between B and S; N_a, N_b, and N'_b are nonces; K_{ab} is the session key between A and B, established by the protocol (it is chosen by the server S); s is a message that A wants to send to B, with the guarantee that the adversary will not be able to obtain it. The notation $\{M\}_K$ represents the shared-key encryption of the message M under the key K.

Message 6 is sent by A only if its interlocutor is B. (This message is used in order to test that the key K_{ab} shared between two honest participants A and B is really secret.)

Chapter 5

Static equivalence

5.1 Definitions and Applications

In the first part we have seen how to verify trace properties such as secrecy, formulated as non-deducibility, and authentication properties in a symbolic model. Now we will investigate *equivalence* properties. These properties formalize the notion of indistinguishability in a symbolic model.

5.1.1 Static equivalence

Remember that messages can be modelled as terms over an abstract algebra: given a set of function symbols $\mathcal{F}$, a set of names $\mathcal{N}$ and a set of variables $\mathcal{X}$ we denote the set of all terms constructed over these sets by $\mathcal{T}(\mathcal{F}, \mathcal{N} \uplus \mathcal{X})$. We consider here a slightly simplified setting where $\mathcal{F}$ only contains public symbols, *i.e.*, $\mathcal{F} = \mathcal{F}_{\mathsf{pub}}$. Moreover, the algebra will be equipped with an equational theory. Let us now be a bit more precise.

Definition 5.1 *An equational theory $\mathcal{E}$ is a set of pairs $\{(M,N) | M, N \in T(\mathcal{F}, \mathcal{N} \uplus \mathcal{X})\}$. We define equality modulo $\mathcal{E}$ denoted $=_\mathcal{E}$ to be the smallest equivalence relation such that*

- *$(M,N) \in \mathcal{E}$ implies that $M =_\mathcal{E} N$,*
- *$=_\mathcal{E}$ is closed under substitutions of terms for variables,*
- *$=_\mathcal{E}$ is closed under application of function symbols, i.e. $M_1 =_\mathcal{E} N_1, \ldots, M_k =_\mathcal{E} N_k$ implies that $\mathsf{f}(M_1, \ldots, M_k) =_\mathcal{E} \mathsf{f}(N_1, \ldots, N_k)$ for all $\mathsf{f} \in \mathcal{F}$ of arity k,*
- *$=_\mathcal{E}$ is closed under bijective renaming of names.*

For convenience in examples we will often just say that $\mathcal{E}$ is defined by the equations

$$M_1 = N_1 \quad \ldots \quad M_n = N_n$$

rather than writing $\mathcal{E} = \{(M_1, N_1), \ldots, (M_n, N_n)\}$. So we will for instance define the equational theory enc which models symmetric encryption and pairings by the equations

$$\mathsf{proj}_1(\langle x, y\rangle) = x \quad \mathsf{proj}_2(\langle x, y\rangle) = y \quad \mathsf{sdec}(\{x\}_y, y) = x.$$

Hence we have that $\mathsf{proj}_2(\langle n, \mathsf{sdec}(\{s\}_k, k)\rangle) =_{\mathsf{enc}} s$.

In Section 2.1.4 we have already introduced static equivalence. Let us rephrase this definition now in this slightly simplified setting. We write $\mathsf{new}\ \overline{n}_1.\sigma_1 =_\alpha \mathsf{new}\ \overline{n}_2.\sigma_2$ when these two frames are the same up to α-conversion, *i.e.* up to a bijective renaming of the bound names. Before defining static equivalence between frames we define what it means for two terms to be equal in a frame. Let $phi = \mathsf{new}\ \overline{n}.\sigma$. We say that $(M =_\mathcal{E} N)\phi$ iff $\phi =_\alpha \mathsf{new}\ \overline{n}'.\sigma'$ for some $\overline{n}'$ and σ' such that $names(M,N) \cap \overline{n}' = \emptyset$ and $M\sigma' =_\mathcal{E} N\sigma'$.

Definition 5.2 *Two frames* $\phi_1 = \mathsf{new}\ \overline{n}_1.\sigma_1$ *and* $\phi_2 = \mathsf{new}\ \overline{n}_2\sigma_2$ *are statically equivalent, written* $\phi_1 \sim_{\mathcal{E}} \phi_2$ *iff* $\mathsf{Dom}(\sigma_1) = \mathsf{Dom}(\sigma_2)$ *and for all terms* M, N *we have that* $(M =_{\mathcal{E}} N)\phi_1 \Leftrightarrow (M =_{\mathcal{E}} N)\phi_2$*.*

Note that it follows directly from the definition that static equivalence is closed under α-conversion.

5.1.2 Applications of static equivalence

We have already seen some uses of static equivalence in Section 2.3. There we used static equivalence to model guessing attacks and to define equivalence properties. We will give here some additional examples.

Password protocols. It is not always possible to rely on previously shared keys or on an existing public key infrastructure. In these cases protocols often rely on shared passwords. However, such passwords are *weak secrets* and can be subject to brute-force attacks. Brute-force attacks on passwords are also refered to as *dictionnary attacks* (refering to a dictionnary containing all passwords) or *guessing attacks* (the search space is small enough that the attacker can guess the password). As we have seen before (Definition 2.3) resistance against offline guessing attacks can be modelled by the means of static equivalence.

Example 5.1 *Let us now consider an example of a password protocol, which indeed resists offline guessing attacks. The EKE protocol [?] can be described by the following 5 steps.*

$$\begin{array}{lll}
1.\ \mathrm{A} \rightarrow \mathrm{B}: & \{(\mathsf{pk}(k))\}_w & \textit{(EKE.1)} \\
2.\ \mathrm{B} \rightarrow \mathrm{A}: & \{\mathsf{aenc}(r, \mathsf{pk}(k))\}w & \textit{(EKE.2)} \\
3.\ \mathrm{A} \rightarrow \mathrm{B}: & \{na\}_r & \textit{(EKE.3)} \\
4.\ \mathrm{B} \rightarrow \mathrm{A}: & \{\langle na, nb\rangle\}_r & \textit{(EKE.4)} \\
5.\ \mathrm{A} \rightarrow \mathrm{B}: & \{nb\}_r & \textit{(EKE.5)}
\end{array}$$

This protocol uses both ciphers and public key encryption. It does however not rely on a public key infrastructure, i.e. we do not assume that public keys are known and can be associated to a particular identity. In the first step (EKE.1) A *generates a new private key* k *and sends the corresponding public key* $\mathsf{pk}(k)$ *to* B*, encrypted (using symmetric encryption) with the shared password* w*. Then,* B *generates a fresh session key* r*, which he encrypts (using asymmetric encryption) with the previously received public key* $\mathsf{pk}(k)$*. Finally, he encrypts the resulting ciphertext with the password* w *and sends the result to* A *(EKE.2). The last three steps (EKE.3-5) perform a handshake to avoid replay attacks. One may note that this is a password-only protocol. A new private and public key are used for each session and the only shared secret between different sessions is the password* w*.*

We use the equational theory EKE *defined by the equations*

$$\mathsf{adec}(\mathsf{aenc}(x, \mathsf{pk}(y)), y) = x \quad \mathsf{sdec}(\mathsf{senc}(x, y), y) = x \quad \mathsf{senc}(\mathsf{sdec}(x, y), y) = x$$
$$\mathsf{proj}_1(\langle x_1, x_2\rangle) = x_1 \qquad \mathsf{proj}_2(\langle x_1, x_2\rangle) = x_2$$

Note the modelling of ciphers with the additional equation $\mathsf{senc}(\mathsf{sdec}(x, y), y) = x$*. The fact that decryption always succeeds is one of the differences between a cipher and a symmetric* *encryption. The protocol can be modeled in our process calculus by* $\mathsf{new}\ w.(P_A \mid\mid P_B)$ *where*

$$\begin{array}{lll@{\qquad\qquad}lll}
P_A & = & \mathsf{new}\ k, na. & P_B & = & \mathsf{new}\ r, nb. \\
 & & \mathsf{out}(c, \{\mathsf{pk}(k)\}_w)). & & & \mathsf{in}(c, y_1). \\
 & & \mathsf{in}(c, x_1). & & & \mathsf{out}(c, \{\mathsf{aenc}(r, \mathsf{sdec}(y_1, w))\}_w). \\
 & & \mathsf{let}\ ra = \mathsf{adec}(\mathsf{sdec}(x_1, w), k). & & & \mathsf{in}(c, y_2). \\
 & & \mathsf{out}(c, \{na\}_{ra}) & & & \mathsf{out}(c, \{\langle \mathsf{sdec}(y_2, r), nb\rangle\}_r). \\
 & & \mathsf{in}(c, x_2). & & & \mathsf{in}(c, y_3) \\
 & & \mathsf{if}\ \mathsf{proj}_1(\mathsf{sdec}(x_2, ra)) = na\ \mathsf{then} & & & \mathsf{if}\ \mathsf{sdec}(y_3, r) = nb\ \mathsf{then} \\
 & & \mathsf{out}(c, \{\mathsf{proj}_2(\mathsf{sdec}(x_2, ra))\}_{ra}) & & & 0
\end{array}$$

We use the let construction for readability, with the obvious meaning, i.e., $\mathsf{let}\ x = M.P$ *stands for* $P\{^M/_x\}$*. An honest execution of this protocol yields the frame* $\mathsf{new}\ w, k, r, na, nb.\sigma$ *where*

$$\sigma = \{x_1 \mapsto \{\mathsf{pk}(k)\}_w, x_2 \mapsto \{\mathsf{aenc}(r, \mathsf{pk}(k))\}_w, x_3 \mapsto \{na\}_r, x_4 \mapsto \{\langle na, nb\rangle\}_r, x_5 \mapsto \{nb\}_r\}$$

We indeed have that $\mathsf{new}\ w, k, r, na, nb.\sigma \uplus \{x_w \mapsto w\} \sim_{\mathsf{EKE}} \mathsf{new}\ w, w', k, r, na, nb.\sigma \uplus \{x_w \mapsto w'\}$*. Using the tool ProVerif [**?**] it is possible to show that this equivalence holds for all reachable frames declaring* w *as* weaksecret*.*

Remark 5.1 *We have used static equivalence to model resistance against guessing attacks. One can note that the same modelization captures* real-or-random *properties in general.*

Anonymity in electronic voting. Consider the following toy protocol for electronic voting where a voter sends his vote to an administrator, encrypted with the administrator's public key and signed with his private key. When the administrator has received all the votes he decrypts them and publishes the result. This can be modelled by the following voter and administrator processes. We consider the same equational theory for asymmetric encryption as before, with the slight difference that aenc will be a ternary function symbol, modelling randomization of the encryption.

$$V = \mathsf{new}\ r; \mathsf{out}(c_1, \mathsf{sign}(\mathsf{aenc}(v, r, \mathsf{pk}(skA)), skV)$$

$$A = \mathsf{in}(c_1, y); \mathsf{out}(c_2, \mathsf{adec}(\mathsf{check}(y, \mathsf{pk}(skV)), skA))$$

To model anonymity we consider two situations each involving two voters. In the first situation V_1 votes v_1 and V_2 votes v_2; in the second situation V_1 votes v_2 and V_2 votes v_1, i.e. the two voters swap their vote. The protocol provides anonymity if these two situations are indistinguishable. Let $\sigma = \{z_1 \mapsto \mathsf{pk}(skA), z_2 \mapsto \mathsf{pk}(skV_1), z_3 \mapsto \mathsf{pk}(skV_2)$ model the knowledge of public keys. The previous scenario can then be modelled by the following two general process:

$$\begin{array}{rl}
VP_1 = & \mathsf{new}\ skA, skV_1, skV_2; \\
 & (\sigma \| V\{^{skV_1}/_{skV}\}\{^{v_1}/_v\}\{^{r_1}/_r\} \| V\{^{skV_2}/_{skV}\}\{^{v_2}/_v\}\{^{r_2}/_r\} \| A\{^{skV_1}/_{skV}\} \| A\{^{skV_2}/_{skV}\}) \\
VP_2 = & \mathsf{new}\ skA, skV_1, skV_2; \\
 & (\sigma \| V\{^{skV_1}/_{skV}\}\{^{v_2}/_v\}\{^{r_1}/_r\} \| V\{^{skV_2}/_{skV}\}\{^{v_1}/_v\}\{^{r_2}/_r\} \| A\{^{skV_1}/_{skV}\} \| A\{^{skV_2}/_{skV}\})
\end{array}$$

Two frames which can be derived from this protocol are

$$\begin{array}{rl}
\phi_1 = & \mathsf{new}\ skA, skV_1, skV_2, v_1, r_1, v_2, r_2; \sigma \uplus \\
 & \{x_1 \mapsto \mathsf{sign}(\mathsf{aenc}(v_1, r_1, \mathsf{pk}(skA)), skV_1); x_2 \mapsto \mathsf{sign}(\mathsf{aenc}(v_2, r_2, \mathsf{pk}(skA)), skV_2; \\
 & \ x_3 \mapsto v_1; x_4 \mapsto v_2\}
\end{array}$$

$$\begin{array}{rl}
\phi_2 = & \mathsf{new}\ skA, skV_1, skV_2, v_1, r_1, v_2, r_2; \sigma \uplus \\
 & \{x_1 \mapsto \mathsf{sign}(\mathsf{aenc}(v_2, r_1, \mathsf{pk}(skA)), skV_1); x_2 \mapsto \mathsf{sign}(\mathsf{aenc}(v_1, r_2, \mathsf{pk}(skA)), skV_2; \\
 & \ x_3 \mapsto v_1; x_4 \mapsto v_2\}
\end{array}$$

and anonymity can be modelled as $\phi_1 \sim_{\mathcal{E}} \phi_2$.

5.1.3 Some properties of static equivalence

We will now show some properties of static equivalence.

Proposition 5.1

$$\mathsf{new}\ \overline{n}_1.\sigma_1 \sim_{\mathcal{E}} \mathsf{new}\ \overline{n}_2.\sigma_2 \textit{ iff } \mathsf{new}\ \overline{n}_1 \uplus \overline{m}.(\sigma_1 \uplus \theta) \sim_{\mathcal{E}} \mathsf{new}\ \overline{n}_2 \uplus \overline{m}.(\sigma_2 \uplus \theta)$$

where $\overline{m} \cap (\mathit{names}(\sigma_1) \cup \mathit{names}(\sigma_2)) = \emptyset$ *and* $(\overline{n}_1 \cup \overline{n}_2) \cap \mathit{names}(\theta) = \emptyset$*.*

A direct corollary of this proposition is the following.

Corollary 5.1

$$\begin{gathered}\mathsf{new}\ \overline{n}_1.\sigma_1 \sim_{\mathcal{E}} \mathsf{new}\ \overline{n}_2.\sigma_2 \textit{ and } \mathsf{new}\ \overline{m}_1.\theta_1 \sim_{\mathcal{E}} \mathsf{new}\ \overline{m}_2.\theta_2 \\ \textit{iff} \\ \mathsf{new}\ \overline{n}_1 \uplus \overline{m}_1.(\sigma_1 \uplus \theta_1) \sim_{\mathcal{E}} \mathsf{new}\ \overline{n}_2 \uplus \overline{m}_2.(\sigma_2 \uplus \theta_2)\end{gathered}$$

where $\overline{m}_i \cap \mathit{names}(\sigma_i) = \emptyset$ *and* $\overline{n}_i \cap \mathit{names}(\theta_i) = \emptyset$ $(i \in \{1, 2\})$*.*

The $\Rightarrow$ direction can be seen as a composition result provided the frames do not share any secret name. The $\Leftarrow$ direction states that whenever two frames are statically equivalent then any subframe is also statically equivalent.

When proving results on static equivalence it is often more convenient to use the following equivalent definition of static equivalence.

Definition 5.3 *Two frames* $\phi_1 = \mathsf{new}\ \overline{n}.\sigma_1$ *and* $\phi_2 = \mathsf{new}\ \overline{n}.\sigma_2$ *(having the same set of restricted names) are statically equivalent, written* $\phi_1 \sim_{\mathcal{E}} \phi_2$ *iff* $\mathsf{Dom}(\sigma_1) = \mathsf{Dom}(\sigma_2)$ *and for all terms* M, N*, such that* $(\mathit{names}(M) \cup \mathit{names}(N)) \cap \overline{n} = \emptyset$ *we have that* $M\sigma_1 =_{\mathcal{E}} N\sigma_1 \Leftrightarrow M\sigma_2 =_{\mathcal{E}} N\sigma_2$*.*

We leave it as an exercise to show that these two definitions are equivalent.

Proof : [of Proposition 5.1]

$\boxed{\Rightarrow}$ We will show the contrapositive. Suppose that $\mathsf{new}\ \overline{n}_1 \uplus \overline{m}.(\sigma_1 \uplus \theta) \not\sim_{\mathcal{E}} \mathsf{new}\ \overline{n}_2 \uplus \overline{m}.(\sigma_2 \uplus \theta)$. Hence there exist two terms M, N such that $\mathit{names}(M, N) \cap (\overline{n}_1 \cup \overline{n}_2 \cup \overline{m}) = \emptyset$ and $M(\sigma_1 \uplus \theta) =_{\mathcal{E}} N(\sigma_1 \uplus \theta)$ and $M(\sigma_2 \uplus \theta) \neq_{\mathcal{E}} N(\sigma_2 \uplus \theta)$ (or vice-versa). As σ_i are ground $M(\sigma_i \uplus \theta) = (M\theta)\sigma_i$ and $N(\sigma_i \uplus \theta) = (N\theta)\sigma_i$. Because $\mathit{names}(M, N) \cap (\overline{n}_1 \cup \overline{n}_2) = \emptyset$ and $(\overline{n}_1 \cup \overline{n}_2) \cap \mathit{names}(\theta) = \emptyset$ we have that $\mathit{names}(M\theta, N\theta) \cap (\overline{n}_1 \cup \overline{n}_2) = \emptyset$. Hence, $M\theta, N\theta$ gives a valid test to distinguish $\mathsf{new}\ \overline{n}_1.\sigma_1$ and $\mathsf{new}\ \overline{n}_2.\sigma_2$.

$\boxed{\Leftarrow}$ We again show the contrapositive. Suppose that $\mathsf{new}\ \overline{n}_1.\sigma_1 \not\sim_{\mathcal{E}} \mathsf{new}\ \overline{n}_2.\sigma_2$. Hence there exist two terms M, N such that $\mathit{names}(M, N) \cap (\overline{n}_1 \cup \overline{n}_2) = \emptyset$ and $M\sigma_1 =_{\mathcal{E}} N\sigma_1$ and $M\sigma_2 \neq_{\mathcal{E}} N\sigma_2$ (or vice-versa). Let M' and N' be two terms obtained from M and N by applying a bijective renaming of names in $\mathit{names}(M, N) \cap \overline{m}$ and variables in $\mathsf{Dom}(\theta)$ by fresh names. Hence, $\mathit{names}(M', N') \cap (\overline{m} \cup \overline{n}_1 \cup \overline{n}_2) = \emptyset$ and $\mathit{vars}(M', N') \cap \mathsf{Dom}(\theta) = \emptyset$. Because $=_{\mathcal{E}}$ and $\neq_{\mathcal{E}}$ are closed under bijective renaming and $\overline{m} \cap (\mathit{names}(\sigma_1) \cup \mathit{names}(\sigma_2)) = \emptyset$ we have that $M'\sigma_1 =_{\mathcal{E}} N'\sigma_1$ and $M'\sigma_2 \neq_{\mathcal{E}} N'\sigma_2$. As $\mathit{vars}(M', N') \cap \mathsf{Dom}(\theta) = \emptyset$ and σ_i are ground we obtain that $M'(\sigma_1 \uplus \theta) =_{\mathcal{E}} N'(\sigma_1 \uplus \theta)$ and $M'(\sigma_2 \uplus \theta) \neq_{\mathcal{E}} N'(\sigma_2 \uplus \theta)$. □

While composition with shared secrets does not hold in general, resistance against offline guessing attacks (in the presence of a passive attacker) composes when the same password is used.

Proposition 5.2

$$\begin{array}{ll} & \mathsf{new}\ w.\mathsf{new}\ \overline{n}_1.(\sigma_1 \uplus \{x \mapsto w\}) \sim_{\mathcal{E}} \mathsf{new}\ w.\mathsf{new}\ w'.\mathsf{new}\ \overline{n}_1.(\sigma_1 \uplus \{x \mapsto w'\}) \\ \textit{and} & \mathsf{new}\ w.\mathsf{new}\ \overline{n}_2.(\sigma_2 \uplus \{x \mapsto w\}) \sim_{\mathcal{E}} \mathsf{new}\ w.\mathsf{new}\ w'.\mathsf{new}\ \overline{n}_2.(\sigma_2 \uplus \{x \mapsto w'\}) \\ & \textit{implies that} \\ & \mathsf{new}\ w.\mathsf{new}\ \overline{n}_1 \uplus \overline{n}_2.(\sigma_1 \uplus \sigma_2 \uplus \{x \mapsto w\}) \sim_{\mathcal{E}} \mathsf{new}\ w.\mathsf{new}\ w'.\mathsf{new}\ n_1 \uplus \overline{n}_2.(\sigma_1 \uplus \sigma_2 \uplus \{x \mapsto w'\}) \end{array}$$

where $w' \notin names(\sigma_i)$ *and* $\overline{n}_1 \cap names(\sigma_2) = \overline{n}_2 \cap names(\sigma_1) = \emptyset$.

Exercise 30 will guide us through the proof of this proposition.

5.1.4 Further readings

This section does not give an exhaustive overview of all papers on the subject. The aim is to give some pointers to the interested reader to papers that are closely related (using similar techniques and notations) to the subjects covered in this chapter.

Static equivalence was first introduced in the applied pi calculus [?] which is a process calculus for cryptographic proptocols. In this context it was used to characterize observational equivalence by a labelled bisimulation, relying on static equivalence. Many properties of static equivalence were already introduced in that paper. Deducibility and static equivalence for equational theories are also discussed in detail in [?]. This paper also presents decidability and complexity results, which we will cover in the next section.

Guessing attacks were first formalized using static equivalence in [?] (other previous formalizations not based on static equivalence were presented in [?, ?] but they seem less natural). Composition results for guessing attacks were shown in [?]: this paper includes Proposition 5.2 showing that offline guessing attacks do compose in the presence of a passive adversary, it is shown that offline guessing attacks do not compose in general in the presence of an active adversary, but sufficient conditions are given to achieve this composition.

The modelling of anonymity by the use of static and above all observational equivalence has been adressed for electronic voting in [?, ?, ?]. Anonymity in a different context is studied in [?]: there it is shown how to model anonymity in authentication protocols by the use of observational equivalence.

5.2 Procedure for subterm convergent equational theories

In this chapter we show that for a large class of equational theories, namely *subterm convergent* equational theories, static equivalence can be decided in polynomial time. This part follows closely the paper by Abadi and Cortier [?]. Other results and more practical procedures will be discussed in the further readings section.

5.2.1 Preliminaries

Let us first introduce some preliminary definitions and notations.

Notations. Given a set of function symbols $\mathcal{F}$ we write $\mathsf{ar}(\mathcal{F})$ for the maximal arity of $f \in \mathcal{F}$. Sometimes in this section, we will write $==$ when we want to emphasize that we mean syntactic equality. For a term t we define $|t|$ to be its size as follows: $|t| = 1$ if t is a variable, a name or a constant and $\mathsf{f}(t_1, \ldots, t_n) = 1 + \sum_{1 \leq i \leq n} |t_i|$. We denote by $st(t)$ the set of all subterms of t and by $pos(t)$ the set of its positions. Then, for $p \in pos(t)$ we write $t|_p$ for the subterm of t at position p. $t[u]_p$ is the term obtained by replacing the subterm at position p by u. The notions of size and subterms are naturally lifted to frames: for $\varphi = \mathsf{new}\ \overline{n}.\{x_1 \mapsto M_1, \ldots x_n \mapsto M_n\}$ we define $|\varphi| = \sum_{1 \leq i \leq n} |M_i|$ and $st(\varphi) = \cup_{1 \leq i \leq n} st(M_i)$.

Rewrite systems and equational theories. A rewrite system $\mathcal{R}$ is a set of rewrite rules $\ell \to r$ such that $vars(r) \subseteq vars(\ell)$. We say that a term t rewrites to u by $\mathcal{R}$ if there exists $\ell \to r \in \mathcal{R}$ and $p \in pos(t)$ such that $t|_p = \ell\sigma$ for some substitution σ and $u = t[r\sigma]_p$. A rewrite system $\mathcal{R}$ is terminating if there is no infinite chain $t_1 \to_{\mathcal{R}} t_2 \to_{\mathcal{R}} \ldots$. A rewrite system $\mathcal{R}$ is confluent if for any t_1, t_2 such that $t \to_{\mathcal{R}}^* t_1$, $t \to_{\mathcal{R}} t_2$ there exists u such that $t_1 \to_{\mathcal{R}}^* u$ and $t_2 \to_{\mathcal{R}}^* u$. $\mathcal{R}$ is convergent if it is both confluent and terminating. For a convergent rewrite system $\mathcal{R}$ we denote by $t\downarrow_{\mathcal{R}}$ the unique normal form of t. A rewrite system is subterm convergent if for any rule $\ell \to r \in \mathcal{R}$ we have that r is either a strict subterm of ℓ or r is a constant. Given an equational theory $\mathcal{E}$ we associate a rewrite system $\mathcal{R}_{\mathcal{E}}$ to it by orienting the equations in $\mathcal{E}$. For readability we generally write $\to_{\mathcal{E}}$ instead of $\to_{\mathcal{R}_{\mathcal{E}}}$. We say that an equational theory is subterm convergent if it can be oriented to a subterm convergent rewrite system.

Example 5.2 *The equational theories* enc, cipher *and* EKE *encountered before are subterm convergent. However, the equational theories for homomorphic encryption* homo, *extending* enc *by the equations*

$$\{\langle x,y\rangle\}_z = \langle \{x\}_z, \{y\}_z\rangle \qquad \mathsf{sdec}(\langle x,y\rangle, z) = \langle \mathsf{sdec}(x,z), \mathsf{sdec}(y,z)\rangle$$

and for blind signatures blind *defined by*

$$\begin{array}{rcl} \mathsf{check}(\mathsf{sign}(x,y),\mathsf{pk}(y)) & = & x \\ \mathsf{unblind}(\mathsf{blind}(x,y),y) & = & x \\ \mathsf{unblind}(\mathsf{sign}(\mathsf{blind}(x,y),z),y) & = & \mathsf{sign}(x,z) \end{array}$$

are not subterm convergent.

DAG **representation of terms.** In order to achieve a polynomial time complexity we will use a DAG representation of terms (which can be exponentially more succint than its tree representation). A DAG representation of a term is a directed acyclic graph (V, l, E, v_0) where

- V is the set of vertices;
- $l : V \to \mathcal{F} \cup \mathcal{N} \cup \mathcal{X}$ is a labelling function;
- $E \subseteq (V \times \{1..\mathsf{ar}(\mathcal{F})\}) \times V$ is the edge relation;
- v_0 is the root vertex.

For each vertex $v \in V$ such that $l(v) = \mathsf{f} \in \mathcal{F}$, v has exactly $\mathsf{ar}(\mathsf{f})$ outgoing edges labelled by 1 to $\mathsf{ar}(f)$. Vertices labelled by names and variables have no outgoing edge. We denote by $E(v,i)$ the unique vertex v' such that $(v,i,v') \in E$. A DAG $D = (V,l,E,v_0)$ defines a term $t(D)$ as $t(D) = \mathsf{f}(t(V,l,E,E(v_0,1)),\ldots,t(V,l,E,E(v_0,\mathsf{ar}(\mathsf{f}))))$ if $l(v_0) = f \in \mathcal{F}$ and $t(D) = l(v_0)$ if $l(v_0) \in \mathcal{N} \cup \mathcal{X}$. We say that a DAG representation (V,l,E,v_0) is minimal if there are no two distinct vertices $v_1, v_2 \in V$ such that $t(V,l,E,v_1) = t(V,l,E,v_2)$.

The size of a DAG D denoted $|D|$ is the number of its vertices. We define the DAG size of a term t, denoted $|t|_{\mathsf{DAG}}$, as $|st(t)|$ which coincides with $|D|$ when D is a minimal DAG representation of t.

Given a DAG representation D we can compute its minimal DAG representation in $\mathcal{O}(|D|^3)$. We check whether there are two vertices v_1, v_2 (at most $|D|^2$ possibilities) such that $l(v_1) = l(v_2)$ and $E(v_1,i) = E(v_2,i)$ for $1 \leq i \leq \mathsf{ar}(l(v_1))$. In that case we delete v_2 from V and replace any occurence of v_2 by v_1. We iterate this at most $|D|$ times.

Hence, we can also check whether $t(D_1) == t(D_2)$ in polynomial time in $|D_1| + |D_2|$.

Given a subterm convergent rewrite system $\mathcal{R}$ and a minimal DAG representation $D = (V,l,E,v_0)$ of a term t we can compute the minimal DAG representation of $t\downarrow_{\mathcal{R}}$ in $\mathcal{O}(|D|^4)$. To see this note that each rewrite rule is of the form $C[x_1,\ldots,x_n] \to C'[x_1,\ldots,x_n]$ or $C[x_1,\ldots,x_n] \to c$. Starting from the root we check whether the rewrite rule applies in D (at most $|C||D|$ tests). If it applies to some vertex v, i.e. $t(V,l,E,v) = C[x_1,\ldots,x_n]\theta$ for some θ, replace v by the vertex representing $C'[x_1,\ldots,x_n]\theta$ (if it is not a constant this vertex exists because it is a subterm; otherwise add a vertex labelled by the constant c). Then minimize the DAG in $\mathcal{O}(|D|^3)$. At each step (except for a constant number of cases) we delete one vertex. Hence the procedure stops after at most $|D|$ iterations and we compute the DAG representation of $t\downarrow_{\mathcal{R}}$ in $\mathcal{O}(|D|^4)$.

5.2.2 Deciding $\sim_{\mathcal{E}}$ in polynomial time for subterm convergent equational theories

Let $\mathcal{E}$ be a subterm convergent equational theory defined by the axioms $\cup_{1\leq i\leq n}\{(\ell_i, r_i)\}$. We define the theory constant $c_{\mathcal{E}} = \max_{1\leq i\leq n}(|\ell_i|, \mathsf{ar}(\mathcal{F})+1)$. By convention, we define $c_{\mathcal{E}}$ to be 1 in the case where $\mathcal{E}$ and $\mathcal{F}$ are empty.

Theorem 5.1 *Let φ and φ' be two frames. We can decide whether $\varphi \sim_{\mathcal{E}} \varphi'$ in polynomial time in $|\varphi| + |\varphi'|$.*

The remaining of this section will be dedicated to the proof of this result. The proof can be split into 3 steps:

1. frame saturation,
2. caracterization of a frame by a finite set of equalities,
3. decidability of $\sim_{\mathcal{E}}$.

We now detail each of these steps.

5.2.2.1 Frame saturation

For each frame φ we can compute a set of terms $\mathsf{sat}(\varphi)$. This set contains all terms that are deducible from the frame by applying only "small" contexts.

Definition 5.4 *Given a frame $\varphi = \mathsf{new}\ \overline{n}.\{x_1 \mapsto M_1, \dots, x_k \mapsto M_k\}$ we define $\mathsf{sat}(\varphi)$ to be the smallest set such that*

1. *$\{M_1, \dots, M_k\} \subseteq \mathsf{sat}(\varphi)$,*
2. *if $M_1, \dots, M_n \in \mathsf{sat}(\varphi)$ and $\mathsf{f}(M_1, \dots, M_n) \in st(\varphi)$ then $\mathsf{f}(M_1, \dots, M_n) \in \mathsf{sat}(\varphi)$,*
3. *if $M_1, \dots, M_n \in \mathsf{sat}(\varphi)$ and C is a context such that $C[M_1, \dots, M_n] \rightarrow_{\mathcal{E}} M$ and $|C| \leq c_{\mathcal{E}}$, $names(C) \cap \overline{n} = \emptyset$ and $M \in st(\varphi)$ then $M \in \mathsf{sat}(\varphi)$.*

Example 5.3 *We again consider the equational theory enc which can be oriented into the following subterm convergent equational theory:*

$$\begin{array}{rl} \mathsf{sdec}(\{x\}_y, y) \rightarrow_{\mathsf{enc}} x & \\ \mathsf{proj}_i(\langle x_1, x_2\rangle) \rightarrow_{\mathsf{enc}} x_i & \qquad (i \in \{1,2\}) \end{array}$$

We have that $c_{\mathsf{enc}} = 5$. Consider the frame

$$\varphi = \mathsf{new}\ s_1, s_2, k_1, k_2.\{x_1 \mapsto \{\langle s_1, s_2\rangle\}_{\langle k_1,k_2\rangle}, x_2 \mapsto k_1, x_3 \mapsto k_2\}$$

We have that $\mathsf{sat}(\varphi) = \{\{\langle s_1, s_2\rangle\}_{\langle s_1,s_2\rangle}, k_1, k_2, \langle k_1, k_2\rangle, \langle s_1, s_2\rangle, s_1, s_2\}$. The first three terms result directly from the frame (rule 1). The term $\langle k_1, k_2\rangle$ can be added by applying the pairing function symbol (rule 2). Term $\langle s_1, s_2\rangle$ can be added in two different ways using rule 3: either apply the context $\mathsf{sdec}(\_,\_)$ on $\{\langle s_1, s_2\rangle\}_{\langle s_1,s_2\rangle}$ and $\langle k_1, k_2\rangle$ or $\mathsf{sdec}(\_, \langle\_,\_\rangle)$ on $\{\langle s_1, s_2\rangle\}_{\langle k_1,k_2\rangle}$, k_1 and k_2. s_i can be added by applying $\mathsf{proj}_i(\_)$ on $\langle s_1, s_2\rangle$. Note that we are not allowed to directly use the context $C = \mathsf{proj}_i(\mathsf{sdec}(\_, \langle\_,\_\rangle))$ (corresponding to the actual recipe for deducing s_i directly from φ) because $|C| > 5$.

We now show that this set can be computed in polynomial time in $|\varphi|$

Proposition 5.3 *The set $\mathsf{sat}(\varphi)$ can be computed in $\mathcal{O}(|\varphi|^{\max(\mathsf{ar}(\mathcal{F}), c_{\mathcal{E}})+2})$.*

Proof : We initialize the set with the elements $\{M_1, \ldots, M_k\}$ (rule 1) and then saturate the set using rules 2 and 3. We notice that $\mathsf{sat}(\varphi) \subseteq st(\varphi)$. Hence, we have that $\mathsf{sat}(\varphi)$ contains at most $|\varphi|$ elements and the saturation will take at most $|\varphi|$ steps.

- If the step is an application of rule 2 we have to construct at most $|\mathcal{F}||\varphi|^{\mathsf{ar}(\mathcal{F})}$ terms. For each of these terms we check whether it is in $\mathsf{sat}(\varphi)$ which can be done in linear time in $|\varphi|$. Hence, this step can be computed in $\mathcal{O}(|\varphi|^{\mathsf{ar}(\mathcal{F})+1})$.
- If the step is an application of rule 3 we have to compare whether any context of size $\leq c_\mathcal{E}$ applied to some M_is in $\mathsf{sat}(\varphi)$ is an instance of the lhs of a rewrite rule and check whether the resulting term is in $\mathsf{sat}(\varphi)$. This can be computed in $\mathcal{O}(|\varphi|^{c_\mathcal{E}+1})$. To see this note that the number of contexts is constant (as $c_\mathcal{E}$ is fixed) and that a context has at most $c_\mathcal{E}$ holes. Hence, we need to consider at most $\mathcal{O}(|\varphi|^{c_\mathcal{E}})$ terms for which we need to check whether they are in $\mathsf{sat}(\varphi)$.

Hence each step can be computed in $\mathcal{O}(|\varphi|^{\max(\mathsf{ar}(\mathcal{F}), c_\mathcal{E})+1})$. As there are at most $|\varphi|$ steps we can compute $\mathsf{sat}(\varphi)$ in $\mathcal{O}(|\varphi|^{\max(\mathsf{ar}(\mathcal{F}), c_\mathcal{E})+2})$.

Moreover any element of $\mathsf{sat}(\varphi)$ can be deduced using a recipe of small DAG size.

Proposition 5.4 *Let $\varphi = \mathsf{new}\ \overline{n}.\sigma$. For all $M \in \mathsf{sat}(\varphi)$ there exists a recipe R_M such that $|R_M|_{\mathsf{DAG}} \leq c_\mathcal{E} \cdot |\varphi|$, names$(R_M) \cap \overline{n} = \emptyset$ and $R_M\sigma =_\mathcal{E} M$.*

Proof : For each term $M \in \mathsf{sat}(\varphi)$ we can give a recipe R_M according to the rule of Definition 5.4 which added M to $\mathsf{sat}(\varphi)$.

1. Let $R_M = x_i$ if $x_i\sigma = M$ and M is added by rule 1.
2. Let $R_M = \mathsf{f}(R_{M_1}, \ldots R_{M_n})$ if M is added by rule 2.
3. Let $R_M = C(R_{M_1}, \ldots R_{M_n})$ if M is added by rule 3.

We can now construct a graph which contains each DAG corresponding to R_M for $M \in \mathsf{sat}(\varphi)$. The graph is constructed as follows.

1. For each recipe R_M constructed by rule 1 add a vertex labelled by x_i.
2. For each recipe $R_M = \mathsf{f}(R_{M_1}, \ldots R_{M_n})$ constructed by rule 2 add a vertex labelled by f and connect it to the vertices corresponding to the terms $R_{M_1}, \ldots R_{M_n}$.
3. For each recipe $R_M = C(R_{M_1}, \ldots R_{M_n})$ constructed by rule 3 construct the minimal DAG corresponding to C and connect it to the vertices corresponding to the terms $R_{M_1}, \ldots R_{M_n}$.

Steps 1 and 2 add 1 vertex to the graph. As $|C| \leq c_\mathcal{E}$ each step 3 adds at most $c_\mathcal{E}$ vertices to the graph. Moreover, we know that the graph can be constructed in $|\varphi|$ steps. Hence its size is bounded by $c_\mathcal{E} \cdot |\varphi|$.

Example 5.4 *Continuing Example 5.3, let*

$$\begin{array}{rcl} M_1 &=& \{\langle s_1, s_2\rangle\}_{\langle k_1,k_2\rangle} \\ M_2 &=& k_1 \\ M_3 &=& k_2 \end{array} \qquad \begin{array}{rcl} M_4 &=& \langle k_1, k_2\rangle \\ M_5 &=& \langle s_1, s_2\rangle \\ M_6 &=& s_1 \\ M_7 &=& s_2 \end{array}$$

denote the elements of $\mathsf{sat}(\varphi)$. *We can define the following recipes*

$$\begin{array}{rcl} R_{M_i} &=& x_i \qquad (1 \leq i \leq 3) \\ R_{M_4} &=& \langle x_2, x_3\rangle \end{array} \qquad \begin{array}{rcl} R_{M_5} &=& \mathsf{sdec}(x_1, R_{M_4}) \\ R_{M_6} &=& \mathsf{proj}_1(R_{M_5}) \\ R_{M_7} &=& \mathsf{proj}_2(R_{M_6}) \end{array}$$

5.2.2.2 Caracterization of a frame by a finite set of equalities

To each frame φ, we can associate a set $\mathsf{Eq}(\varphi)$ which is finite (up to renaming) and caracterizes all equalities which hold in φ.

Definition 5.5 *Given a frame φ we define $\mathsf{Eq}(\varphi)$ to be the set of equalities*

$$C_1[R_{M_1}, \dots R_{M_k}] = C_2[R_{N_1}, \dots R_{N_\ell}]$$

such that $|C_1|, |C_2| \leq c_{\mathcal{E}}$, $M_{1 \leq i \leq k}, N_{1 \leq j \leq \ell} \in \mathsf{sat}(\varphi)$ and $(C_1[R_{M_1}, \dots R_{M_k}] =_{\mathcal{E}} C_2[R_{N_1}, \dots R_{N_\ell}])\varphi$. We write $\varphi' \models \mathsf{Eq}(\varphi)$ iff $(M =_{\mathcal{E}} N)\varphi'$ for all $(M = N) \in \mathsf{Eq}(\varphi)$.

Example 5.5 *Let us continue with Example 5.4. Omitting redundant and trivial equations we have that $\mathsf{Eq}(\varphi) = \{(\{R_{M_5}\}_{R_{M_4}} = R_{M_1}), \langle R_{M_6}, R_{M_7}\rangle = R_{M_5})\}$. Intuitively these equalities state that M_1 is indeed an encryption with the key M_4 and M_1 is the encryption of a pair.*

We now show two crucial properties of $\mathsf{Eq}(\varphi)$.

Lemma 5.1 *Let $\varphi = \mathsf{new}\ \overline{n}.\sigma$ and φ' be two frames such that $\varphi' \models \mathsf{Eq}(\varphi)$. For all contexts C_1 and C_2 such that $names(C_1, C_2) \cap \overline{n} = \emptyset$ and for all terms $M_{1 \leq i \leq k}, N_{1 \leq j \leq \ell} \in \mathsf{sat}(\varphi)$ if $C_1[M_1, \dots, M_k] == C_2[N_1, \dots, N_\ell]$ then $(C_1[R_{M_1}, \dots, R_{M_k}] =_{\mathcal{E}} C_2[R_{N_1}, \dots, R_{N_\ell}])\varphi'$.*

Proof : The proof is done by induction on $|C_1| + |C_2|$.

Base case. $|C_1|, |C_2| \leq c_{\mathcal{E}}$. We have that

$$C_1[M_1, \dots, M_k] == C_2[N_1, \dots, N_\ell]$$

As $M_i, N_j \in \mathsf{sat}(\varphi)$ we have by Proposition 5.4 that R_{M_i}, resp. R_{N_j}, are recipes for M_i, resp. N_j, in φ. Hence,

$$(C_1[R_{M_1}, \dots, R_{M_k}] =_{\mathcal{E}} C_2[R_{N_1}, \dots, R_{N_\ell}])\varphi$$

As $|C_1|, |C_2| \leq c_{\mathcal{E}}$, we have that $(C_1[R_{M_1}, \dots, R_{M_k}] = C_2[R_{N_1}, \dots, R_{N_\ell}]) \in \mathsf{Eq}(\varphi)$ and hence

$$(C_1[R_{M_1}, \dots, R_{M_k}] =_{\mathcal{E}} C_2[R_{N_1}, \dots, R_{N_\ell}])\varphi'.$$

Inductive case. We consider two cases.

- $C_1 \neq \_$ and $C_2 \neq \_$.
  In that case, we have that $C_1 == \mathsf{f}(C_1^1, \dots C_1^n)$ and $C_2 == \mathsf{f}(C_2^1, \dots C_2^n)$ and for $1 \leq i \leq n$

  $$C_1^i[M_1, \dots, M_k] == C_2^i[N_1, \dots, N_\ell].$$

  By induction hyptothesis, we have that $(C_1^i[R_{M_1}, \dots, R_{M_k}] =_{\mathcal{E}} C_2^i[R_{N_1}, \dots, R_{N_\ell}])\varphi'$. Hence, as $=_{\mathcal{E}}$ is closed under application of function symbols we also have that

  $$(C_1[R_{M_1}, \dots, R_{M_k}] =_{\mathcal{E}} C_2[R_{N_1}, \dots, R_{N_\ell}])\varphi'.$$

- Either $C_1 == \_$ or $C_2 == \_$. Let us suppose that $C_1 == \mathsf{f}(C_1^1, \dots C_1^n)$ and $C_2 == \_$.
  Let $M, M_1, \dots, M_k \in \mathsf{sat}(\varphi)$ such that $C_1[M_1, \dots, M_k] == M$. Hence we have

  $$\mathsf{f}(C_1^1[M_1, \dots, M_k], \dots, C_1^n[M_1, \dots, M_k]) == M.$$

  Let $N_i = C_1^i[M_1, \dots, M_k]$ for $1 \leq i \leq n$. As $M \in \mathsf{sat}(\varphi)$ and $N_i \in st(M)$ we have that $N_i \in st(\mathsf{sat}(\varphi))$. Applying iteratively rule 2 of Definition 5.4 we obtain that $N_i \in \mathsf{sat}(\varphi)$. We can apply the induction hypothesis on $N_i == C_1^i[M_1, \dots, M_k]$ and obtain that $(R_{N_i} =_{\mathcal{E}} C_1^i[R_{M_1}, \dots, R_{M_k}])\varphi'$. Moreover, $M == \mathsf{f}(N_1, \dots, N_n)$. Applying the base case (with contexts $\_$ and $\mathsf{f}(\_, \dots, \_)$) we obtain that $(R_M =_{\mathcal{E}} \mathsf{f}(R_{N_1}, \dots, R_{N_n}))\varphi'$. From $(R_{N_i} =_{\mathcal{E}} C_1^i[R_{M_1}, \dots, R_{M_k}])\varphi'$ and $(R_M =_{\mathcal{E}} \mathsf{f}(R_{N_1}, \dots, R_{N_n}))\varphi'$ we conclude that

$$(C_1[R_{M_1}, \dots, R_{M_k}] =_{\mathcal{E}} R_M)\varphi'.$$

Lemma 5.2 *Let $\varphi = \mathsf{new}\ \overline{n}.\sigma$ and C_1 a context such that $names(C_1) \cap \overline{n} = \emptyset$. For all $M_1, \dots, M_k \in \mathsf{sat}(\varphi)$ and T such that $C_1[M_1, \dots, M_k] \to_{\mathcal{E}}^* T$ there exist C_2 and $M_1', \dots, M_\ell' \in \mathsf{sat}(\varphi)$ such that $names(C_2) \cap \overline{n} = \emptyset$ and $T == C_2[M_1', \dots M_\ell']$. Moreover, if $\varphi' \models \mathsf{Eq}(\varphi)$ then $(C_1[R_{M_1}, \dots R_{M_k}] =_{\mathcal{E}} C_2[R_{M_1'}, \dots R_{M_k'}])\varphi'$.*

5.2.2.3 Decidability of $\sim_{\mathcal{E}}$

We now show the key proposition of the decidability proof.

Proposition 5.5

$$\varphi \sim_{\mathcal{E}} \varphi' \ \Leftrightarrow\ \varphi \models \mathsf{Eq}(\varphi') \text{ and } \varphi' \models \mathsf{Eq}(\varphi)$$

Proof :
$\boxed{\Rightarrow}$ It follows directly from the definition of static equivalence that $\varphi \sim_{\mathcal{E}} \varphi' \quad \Rightarrow \quad \varphi \models \mathsf{Eq}(\varphi')$ and $\varphi' \models \mathsf{Eq}(\varphi)$.

$\boxed{\Leftarrow}$ We have that $\varphi' \models \mathsf{Eq}(\varphi)$. Let M, N be such that $(M =_{\mathcal{E}} N)\varphi$, i.e., $\varphi =_\alpha \mathsf{new}\ \overline{n}.\sigma$ such that $names(M, N) \cap \overline{n} = \emptyset$ and $M\sigma =_{\mathcal{E}} N\sigma$. Hence $M\sigma\downarrow_{\mathcal{E}} == N\sigma\downarrow_{\mathcal{E}}$. Let $T = M\sigma\downarrow_{\mathcal{E}}$. By Lemma 5.2 we have that there exist C_M and $M_1, \dots, M_k \in \mathsf{sat}(\varphi)$ such that $names(C_M) \cap \overline{n} = \emptyset$ and

$$T == C_M[M_1, \dots M_k] \text{ and } (M =_{\mathcal{E}} C_M[R_{M_1}, \dots R_{M_k}])\varphi'.$$

Similarly, as $T == N\sigma\downarrow_{\mathcal{E}}$ there exist C_N and $N_1, \dots, N_\ell \in \mathsf{sat}(\varphi)$ such that $names(C_N) \cap \overline{n} = \emptyset$ and

$$T == C_N[N_1, \dots N_\ell] \text{ and } (N =_{\mathcal{E}} C_N[R_{N_1}, \dots R_{N_\ell}])\varphi'.$$

We obtain that $C_M[M_1, \dots M_k] == C_N[N_1, \dots N_\ell]$ and by Lemma 5.1 we obtain that

$$(C_M[R_{M_1}, \dots R_{M_k}] =_{\mathcal{E}} C_N[R_{N_1}, \dots R_{N_\ell}])\varphi'$$

and hence $(M =_{\mathcal{E}} N)\varphi'$.

Conversely, assuming $\varphi \models \mathsf{Eq}(\varphi')$ and $(M =_{\mathcal{E}} N)\varphi'$ we obtain that $(M =_{\mathcal{E}} N)\varphi$. We conclude that $\varphi \sim_{\mathcal{E}} \varphi'$.

From this and previous propositions follows a polynomial time decision procedure. To decide $\varphi \sim_{\mathcal{E}} \varphi'$ construct $\mathsf{sat}(\varphi)$ and $\mathsf{sat}(\varphi')$. These constructions can be done in polynomial time (Proposition 5.3). Moreover, each term in $\mathsf{sat}(\varphi) \cup \mathsf{sat}(\varphi')$ has polynomial DAG size (Proposition 5.4). Because of renamings, $\mathsf{Eq}(\varphi)$ and $\mathsf{Eq}(\varphi')$ may be infinite. However, as each of the equalities in $\mathsf{Eq}(\varphi)$ and $\mathsf{Eq}(\varphi')$ is of the form $C_1[R_{M_1}, \dots, R_{M_k}] = C_2[R_{N_1}, \dots, R_{N_\ell}]$ with $|C_i| \leq c_{\mathcal{E}}$ each equality contains at most $2 \cdot c_{\mathcal{E}}$ distinct names which can be fixed. There are at most $\mathcal{O}((|\varphi|^{c_{\mathcal{E}}})^2)$ equalities in $\mathsf{Eq}(\varphi)$ each having a polynomial DAG size. Checking whether two terms in DAG representation are equal can be done in polynomial time as well. Hence, we can check in polynomial time that $\varphi \models \mathsf{Eq}(\varphi')$ and $\varphi' \models \mathsf{Eq}(\varphi)$.

5.2.3 Deciding $\sim_{\mathcal{E}}$ vs deciding $\vdash_{\mathcal{E}}$

An interesting question is what is the relation between the decidability problems for $\sim_{\mathcal{E}}$ and $\vdash_{\mathcal{E}}$. In general, these problems cannot be reduced to each other: there exist an equational theory, such that $\vdash_{\mathcal{E}}$ is decidable and $\sim_{\mathcal{E}}$ is not; there also exists (more surprisingly) an equational theory, such that $\sim_{\mathcal{E}}$ is decidable and $\vdash_{\mathcal{E}}$ is not. Indeed for many equational theories deciding $\vdash_{\mathcal{E}}$ can be reduced to $\sim_{\mathcal{E}}$. In particular, suppose that $\sim_{\mathcal{E}}$ is decidable over a signature $\mathcal{F} \uplus \{\mathsf{h}\}$ where h is a free symbol. Then we can decide $\vdash_{\mathcal{E}}$ over the signature $\mathcal{F}$ by the following encoding:

$$\mathsf{new}\ \overline{n}.\sigma \vdash_{\mathcal{E}} t \text{ iff } \mathsf{new}\ \overline{n}.\sigma \uplus \{x \mapsto \mathsf{h}(t)\} \sim_{\mathcal{E}} \mathsf{new}\ \overline{n}, a.\sigma \uplus \{x \mapsto a\}$$

where $a \notin \mathit{names}(\sigma)$. In particular this implies that $\vdash_{\mathcal{E}}$ is decidable in polynomial time for subterm convergent equational theories.

5.2.4 Further readings

While the here described procedure (from [?]) is indeed effective a direct implementation would not be efficient. Procedures for deciding static equivalence for subterm convergent equational theories have been proposed in [?, ?] and have been implemented in the tools YAPA and KISS. Both procedures can also be used to decide the theories blind and homo presented in Example 5.2. (The procedure presented in [?] and its implementation in the tool KISS is actually the outcome of an M2 internship following this course.) In [?], decidability for blind and homo (and a more general class of equational theories) was already shown, however, no generic procedure was presented.

In [?] it is shown that static equivalence is also decidable for a class of monoidal theories (which are theories over associative commutative operators including theories for exclusive or and abelian groups). Another important result [?] is that (under some mild assumptions) decision procedures for disjoint equational theories can be combined to obtain a procedure for deciding the joint equational theory.

While decidability of static equivalence has been extensively studied, the current knowledge on decidability of obervational equivalence (the generalization to the active case) is very partial. We currently only have approximate procedures for an unboundned number of sessions [?] (implemented in the tool ProVerif), approximate procedures for a bounded number of sessions [?, ?, ?] and a decision procedure for a restricted class of processes [?].

The relationship between deciding deducibility and static equivalence has been first investigated in [?]. There the above encoding of deducibility into static equivalence is proposed and its correctness proven in detail. An equational theory which is decidable for static equivalence but not for deducibility is also presented. In [?] another example of such a theory is given with detailed proofs.

5.3 Exercises

Exercice 27
Consider the equational theory enc defined by the equations

$$\mathsf{sdec}(\{x\}_y, y) = x \quad \mathsf{proj}_1(\langle x_1, x_2\rangle) = x_1 \quad \mathsf{proj}_2(\langle x_1, x_2\rangle) = x_2$$

and cipher which extends enc by the equation $\{\mathsf{sdec}(x,y)\}_y = x$. Which of the following pairs of frames are statically equivalent? Whenever applicable give the distinguishing test.

$$\begin{array}{rcl}
\{x \mapsto a\} & \overset{?}{\sim}_{\mathsf{enc}} & \{x \mapsto b\} \\
\{x \mapsto \{0\}_k\} & \overset{?}{\sim}_{\mathsf{enc}} & \{x \mapsto \{1\}_k\} \\
\mathsf{new}\ k.\{x \mapsto \{0\}_k\} & \overset{?}{\sim}_{\mathsf{enc}} & \mathsf{new}\ k.\{y \mapsto \{1\}_k\} \\
\mathsf{new}\ k.\{x \mapsto \{0\}_k\} & \overset{?}{\sim}_{\mathsf{enc}} & \mathsf{new}\ k.\{x \mapsto \{1\}_k\} \\
\mathsf{new}\ n,k.\{x \mapsto \{n\}_k, y \mapsto k\} & \overset{?}{\sim}_{\mathsf{enc}} & \mathsf{new}\ n,k,k'.\{x \mapsto \{n\}_k, y \mapsto k'\} \\
\mathsf{new}\ n,k.\{x \mapsto \{n\}_k, y \mapsto k\} & \overset{?}{\sim}_{\mathsf{cipher}} & \mathsf{new}\ n,k,k'.\{x \mapsto \{n\}_k, y \mapsto k'\}
\end{array}$$

Exercice 28
In Section 5.1.2 we considered a toy voting protocol. However, the anonymity relied on the fact that the administrator waits to have both votes before publishing the result (to model this we could enhance the language with a synchronization construct). Show that as specified above $VP_1 \not\approx_\ell VP_2$.

Exercice 29
Proposition 5.1 does not hold any more if we drop one of the condition $\overline{m} \cap (names(\sigma_1) \cup names(\sigma_2)) = \emptyset$ and $(\overline{n}_1 \cup \overline{n}_2) \cap names(\theta) = \emptyset$. Give a counter-example in each case when a condition is omitted.

Exercice 30
The aim of this exercise is to guide us through a proof of Proposition 5.2. This proof requires us to prove several properties of static equivalence. Suppose $\mathsf{new}\ \overline{n}_1.\sigma_1 \sim_{\mathcal{E}} \mathsf{new}\ \overline{n}_2.\sigma_2$. Show that

$$\mathsf{new}\ n.\mathsf{new}\ \overline{n}_1.\sigma_1 \sim_{\mathcal{E}} \mathsf{new}\ n.\mathsf{new}\ \overline{n}_2.\sigma_2 \text{ where } n \notin (\overline{n}_1 \cup \overline{n}_2), \tag{5.1}$$

$$\mathsf{new}\ \overline{n}_1.\sigma_1\{^s/_n\} \sim_{\mathcal{E}} \mathsf{new}\ \overline{n}_2.\sigma_2\{^s/_n\} \text{ where } n \notin (\overline{n}_1 \cup \overline{n}_2) \text{ and } s \text{ is a fresh name.} \tag{5.2}$$

Let $s \notin (\overline{n}_1 \cup \overline{n}_2)$. Show that

$$\mathsf{new}\ \overline{n}_1.\sigma_1 \sim_{\mathcal{E}} \mathsf{new}\ \overline{n}_2.\sigma_2 \text{ iff } \mathsf{new}\ s.\mathsf{new}\ \overline{n}_1.\sigma_1 \uplus \{x \mapsto s\} \sim_{\mathcal{E}} \mathsf{new}\ s.\mathsf{new}\ \overline{n}_2.\sigma_2 \uplus \{x \mapsto s\} \tag{5.3}$$

Use these properties (as well as Proposition 5.1) to show that the following 3 statements are equivalent.

1. $\mathsf{new}\ w.\mathsf{new}\ \overline{n}.\sigma \uplus \{x \mapsto w\} \sim_{\mathcal{E}} \mathsf{new}\ w.\mathsf{new}\ w'.\mathsf{new}\ \overline{n}.\sigma \uplus \{x \mapsto w\}$
2. $\mathsf{new}\ \overline{n}.\sigma \sim_{\mathcal{E}} \mathsf{new}\ w.\mathsf{new}\ \overline{n}.\sigma$
3. $\mathsf{new}\ \overline{n}.\sigma \sim_{\mathcal{E}} \mathsf{new}\ \overline{n}.\sigma\{^{w'}/_w\}$ where w' is a fresh name.

From these results show Proposition 5.2.

Exercice 31
Give an example illustrating that the procedure given in Section 5.2 is wrong if we would define the theory constant $c_{\mathcal{E}}$ to be $\max_{1\leq i\leq n}(|\ell_i|)$ instead of $\max_{1\leq i\leq n}(|\ell_i|, \mathsf{ar}(\mathcal{F})+1)$.

Exercice 32
We discussed the modelling of guessing attacks by static equivalence. Use the procedure of Section 5.2 to show that

$$\mathsf{new}\ w.\mathsf{new}\ n.\{z_1 \mapsto \{n\}_w; x_w \mapsto w\} \not\sim_{\mathsf{enc}} \mathsf{new}\ w.\mathsf{new}\ w'.\mathsf{new}\ n.\{z_1 \mapsto \{n\}_w; x_w \mapsto w'\}$$

and

$$\mathsf{new}\ w.\mathsf{new}\ n.\{z_1 \mapsto \{n\}_w; x_w \mapsto w\} \sim_{\mathsf{cipher}} \mathsf{new}\ w.\mathsf{new}\ w'.\mathsf{new}\ n.\{z_1 \mapsto \{n\}_w; x_w \mapsto w'\}$$

Exercice 33

The here presented procedure terminates in polynomial time given that terms are implemented by DAGs rather than trees. Give an example of two frames φ_1, φ_2 such that $\varphi_1 \not\sim_{\mathsf{enc}} \varphi_2$ but every distinguishing test is of exponential size in $|\varphi_1| + |\varphi_2|$.

Exercice 34

Give an example of two frames φ_1 and φ_2 such that $\varphi_1 \not\sim_{\mathsf{homo}} \varphi_2$ (for the theory of homomorphic encryption introduced in Example 5.2) for which the above procedure fails to distinguish the frames.

Chapter 6

Composition Results

This chapter is related to the work of R. Canetti *et al.* who study universal composability of protocols [?] in a different context (cryptographic model). They consider composition of a protocol with arbitrary other protocols and composition of a protocol with copies of itself. However, in the initial approach [?], they do not allow shared data between protocols, meaning that new encryption and signing keys have to be generated for each component. Note that in the symbolic setting, this kind of composition trivially holds since the definition of security properties always involved an arbitrary context. Indeed, if $\mathsf{new}\ \overline{n}.P_1$ satisfies a secrecy property, this means that $(\mathsf{new}\ \overline{n}.P_1) \parallel A$ satisfies the secrecy property for any attacker A. Hence, in particular, we have that the secrecy property holds if we consider a protocol P_2 running in parallel.

Composition theorems that allow joint states between protocols are proposed in further work (see *e.g.* [?, ?, ?]). Such composition theorems are more useful. They allows one to compose different protocols that share some keying material.

"if $\mathsf{new}\ \overline{n}.P_1$ is secure, then $\mathsf{new}\ \overline{n}.(P_1 \parallel P_2)$ is secure."

In this chapter, we prove composability for tagged protocols and secrecy property only. Many parts of this chapter is borrowed from [?] and [?].

6.1 Motivation

When a protocol has been proved secure for an unbounded number of sessions, against a fully active adversary that can intercept, block and send new messages, there is absolutely no guarantee if the protocol is executed in an environment where other protocols are executed, possibly sharing some common keys like public keys or long-term symmetric keys. The interaction with the other protocols may dramatically damage the security of a protocol. Consider for example the two following naive protocols.

$$P_1 : \quad A \to B : \ \mathsf{aenc}(s, \mathsf{pk}(b)) \qquad\qquad P_2 : \quad \begin{array}{ll} A \to B : & \mathsf{aenc}(N_a, \mathsf{pk}(b)) \\ B \to A : & N_a \end{array}$$

In protocol P_1, the agent A simply sends a secret s encrypted under B's public key. In protocol P_2, the agent sends some fresh nonce to B encrypted under B's public key. The agent B acknowledges A's message by forwarding A's nonce. While P_1 executed alone easily guarantees the secrecy of s, even against active adversaries, the secrecy of s is no more guaranteed when the protocol P_2 is executed. Indeed, an adversary may use the protocol P_2 as an oracle to decrypt any message. More realistic examples illustrating interactions between protocols can be found in *e.g.* [?].

The purpose of this chapter is to investigate sufficient conditions for a protocol to be safely used in an environment where other protocols may be executed as well. We show that whenever a protocol is proved secure when it is executed alone, its security is not compromised by the interactions with any other protocol, provided each protocol is given an identifier (e.g. the protocol's name) that should appear in any encrypted message. Continuing our example, let us consider the two slightly modified protocols.

$$P_1': \quad A \to B: \ \mathsf{aenc}(\langle 1, s\rangle, \mathsf{pk}(b)) \qquad\qquad P_2': \quad \begin{array}{ll} A \to B: & \mathsf{aenc}(\langle 2, N_a\rangle, \mathsf{pk}(b)) \\ B \to A: & N_a \end{array}$$

The composition theorem (formally stated in Section 6.2) will ensure that P_1' can be safely executed together with P_2', without compromising the secrecy of s.

Being able to share keys between protocols is a very desirable property as it allows to save both memory (for storing keys) and time since generating keys is time consuming in particular in the case of public key encryption. We provide in Section 6.2.3 examples of protocols that share secret materials. For security reasons however, most protocols currently make use of different keys.

The idea of adding an identifier in encrypted messages follows the spirit of the rules proposed in the paper of M. Abadi and R. Needham on prudent engineering practice for cryptographic protocols [?] (Principle 10). The use of unique protocol identifiers is also recommended in [?, ?] and has also been used in the design of fail-stop protocols [?]. In this chapter, we prove that it is sufficient for securely executing several protocols in the same environment. We will see in Section 6.3 how to extend this technique to compose a protocol with copies of itself.

6.2 Parallel Composition under Shared Keys

For sake of simplicity, we consider symmetric encryption only. We consider protocol written with pattern matching (no conditionals) and we do not use explicitly destructors.

6.2.1 Theorem

Even if a protocol is secure for an unbounded number of sessions, its security may collapse if the protocol is executed in an environment where other protocols sharing some common keys are executed. We have seen a first example in the previous section. We need to consider some restrictions.

Hypothesis 1. To avoid a ciphertext from a protocol P_1 to be decrypted in an another protocol P_2, we introduce the notion of *well-tagged* protocol. This notion is formally defined below and relies on the following notion of *encrypted subterms.*

Let α be a term. We say that a term t is α-tagged if for every $\mathsf{senc}(t_1, t_2) \in st(t)$, we have that $t_1 = \langle \alpha, t_1' \rangle$ for some term t_1'. A term is said *well-tagged* if it is α-tagged for some term α. A protocol P is α-tagged if any term occurring in the role of the protocol is α-tagged. A protocol is said *well-tagged* if it is α-tagged for some term α.

Requiring that a protocol is well-tagged can be very easily achieved in practice: it is sufficient for example to add the name of the protocol in each encrypted term. Moreover, note that (as opposite to [?]) this does not require that the agents check that nested encrypted terms are correctly tagged. For example, let P be the following protocol:

$$P = \mathsf{in}(c, \mathsf{senc}(\langle \alpha, x\rangle, k)).\ \mathsf{out}(c, \mathsf{senc}(\langle \alpha, x\rangle, k')).$$

The protocol P is α-tagged. Note that the message $\mathsf{senc}(\langle \alpha, \mathsf{senc}(a, k)\rangle, k)$ (which is not α-tagged) would be accepted by P.

Hypothesis 2. Tagging protocols is not sufficient, indeed critical long-term keys should not be revealed in clear. Consider for example the two well-tagged protocols

$$P_3: \quad A \to B: \ \mathsf{senc}(\langle \alpha, s\rangle, k_{ab}) \qquad\qquad P_4: \quad A \to B: \ k_{ab}$$

The security of protocol P_3 is again compromised by the execution of P_4. Thus we will require that long-term keys do not occur in plaintext in the protocol. The set *plaintext*(t) of *plaintext* of a term t is the set of names and variables, that is recursively defined as follows:

- *plaintext*$(u) = \{u\}$ when u is a variable or a name,
- *plaintext*$(\langle u_1, u_2\rangle) =$ *plaintext*$(u_1) \cup$ *plaintext*(u_2), and
- *plaintext*$(\mathsf{senc}(u_1, u_2)) =$ *plaintext*(u_1).

This notation is extended to set of terms and protocols as expected.

Some weird protocols may still reveal critical keys in a hidden way. Consider for example the following (α-tagged) protocol.

$$P = \mathsf{new}\ k_{ab}.(\mathsf{out}(c, \mathsf{senc}(\langle \alpha, a\rangle, k_{ab})) \parallel \mathsf{in}(c, \mathsf{senc}(\langle \alpha, a\rangle, x)).\ \mathsf{out}(c, x))$$

While the long-term key k_{ab} does not appear in plaintext, the key k_{ab} is revealed after simply one normal execution of the protocol. This protocol is however not realistic since an unknown value cannot be learned (and sent) if it does not appear previously in plaintext. Thus we will further require (Condition 2 of Theorem 6.1) that a variable occurring in plaintext in a sent message, has to previously occur in plaintext in a received message.

We show that two well-tagged protocols can be safely composed as soon as they use different tags and that critical long-term keys do not appear in plaintext. Here, we assume that names are restricted. In other words, names are not known by the attacker unless they are explicitly given in his initial knowledge. Sometimes, we write $\mathsf{new}\ \overline{n}.(P \parallel Q)$ to insist on the fact that those names $\overline{n}$ are not known by the attacker or shared between the processes P and Q.

Theorem 6.1 *Let P_1 and P_2 be two well-tagged protocols such that P_1 is α-tagged and P_2 is β-tagged with $\alpha \neq \beta$. Let T_0 (intuitively the initial knowledge of the intruder) be a set of names. Let $\overline{n} = (fn(P_1) \cup fn(P_2)) \smallsetminus T_0$ be the set of* critical names. *Let m be a term constructed from P_1 such that m is α-tagged and $fv(m) \subseteq bv(P_1)$. Moreover, we assume that*

1. *critical names do not appear in plaintext, that is*
$$\overline{n} \cap (plaintext(P_1) \cup plaintext(P_2)) = \emptyset.$$
2. *any $x \in bv(P_1)$ (resp. $bv(P_2)$) "occurs first" at a plaintext position.*

Then $\mathsf{new}\ \overline{n}.P_1$ preserves the secrecy of m for the initial knowledge T_0 if, and only if, $\mathsf{new}\ \overline{n}.(P_1 \parallel P_2)$ preserves the secrecy of m for T_0.

We require that terms from P_1 and P_2 are tagged with distinct tags for simplicity. The key condition is actually that for any encrypted subterm t_1 of P_1 and for any encrypted subterm t_2 of P_2, the terms t_1 and t_2 cannot be unified.

Theorem 6.1 is proved by contradiction. Assume that $\mathsf{new}\ \overline{n}.(P_1 \parallel P_2)$ does not preserve the secrecy of m for T_0. It means that there exists a scenario sce for $\mathsf{new}\ \overline{n}.(P_1 \mid P_2)$ such that the constraint system associated to this scenario, T_0 and m is satisfiable. In this case, the scenario can be turned into a scenario for P_1 such that the constraint system associated to this new scenario, T_0 and m is satisfiable, which means that P_1 does not preserve the secrecy of m for some initial knowledge T_0, contradiction.

To prove this composition result, we first need to show that messages from two combined protocols do not need to be mixed up to mount an attack. For this purpose, we refine the decision procedure presented in Chapter 3 that allows us to control the form of the execution traces.

6.2.2 Main Steps of the Proof

To prove our composition result, we first refine the decision procedure presented in Chapter 3 for solving constraint systems. The *simplification rules* we consider are the following ones:

$$\begin{array}{llcll}
\mathsf{R}_1: & \mathcal{C} \wedge T \overset{?}{\vdash} u & \rightsquigarrow & \mathcal{C} & \text{if } T \cup \{x \mid T' \overset{?}{\vdash} x \in \mathcal{C}, T' \subsetneq T\} \vdash u \\
\mathsf{R}_2: & \mathcal{C} \wedge T \overset{?}{\vdash} u & \rightsquigarrow_\sigma & \mathcal{C}\sigma \wedge T\sigma \overset{?}{\vdash} u\sigma & \text{if } \sigma = \mathsf{mgu}(t,u) \text{ where } t \in st(T),\ t \neq u, \\
 & & & & \text{and } t, u \text{ are neither variables nor pairs} \\
\mathsf{R}_3: & \mathcal{C} \wedge T \overset{?}{\vdash} u & \rightsquigarrow_\sigma & \mathcal{C}\sigma \wedge T\sigma \overset{?}{\vdash} u\sigma & \text{if } \sigma = \mathsf{mgu}(t_1,t_2),\ t_1, t_2 \in st(T),\ t_1 \neq t_2, \\
 & & & & \text{and } t_1, t_2 \text{ are neither variables nor pairs} \\
\mathsf{R}_4: & \mathcal{C} \wedge T \overset{?}{\vdash} u & \rightsquigarrow & \bot & \text{if } vars(T,u) = \emptyset \text{ and } T \not\vdash u \\
\mathsf{R}_\mathsf{f}: & \mathcal{C} \wedge T \overset{?}{\vdash} \mathsf{f}(u,v) & \rightsquigarrow & \mathcal{C} \wedge T \overset{?}{\vdash} u \wedge T \overset{?}{\vdash} v & \text{for } f \in \{\langle\rangle, \mathsf{senc}\}
\end{array}$$

Our rules are the same than those presented in Chapter 3 (or more precisely those given in Exercise 17) except that we forbid unification of terms headed by $\langle \_, \_ \rangle$. Correction and termination are still ensured and we show that they still form a complete decision procedure. Intuitively, unification between pairs is useless since pairs can be decomposed in order to perform unification on its components. Then, it is possible to build again the pair if necessary. Note that this is not always possible for encryption since the key used to decrypt or encrypt may be unknown by the attacker. Proving that forbidding unification between pairs still leads to a complete decision procedure required in particular to introduce a new notion of minimality for tree proofs for deduction (see Exercises 35 and 36). Note that this result is of independent interest. Indeed, we provide a more efficient decision procedure for solving constraint systems, thus for deciding secrecy for a bounded number of sessions. Of course, the theoretical complexity remains the same (NP).

Theorem 6.2 *Let $\mathcal{C}$ be an unsolved constraint system.*

1. *(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 $\mathcal{C}$.*

2. *(Completeness) If θ is a solution of $\mathcal{C}$, then there exist a solved constraint system $\mathcal{C}'$ and substitutions σ, θ' such that $\theta = \sigma\theta'$, $C \rightsquigarrow_\sigma^* \mathcal{C}'$ and θ' is a solution of $\mathcal{C}'$.*

3. *(Termination) There is no infinite chain $\mathcal{C} \rightsquigarrow_{\sigma_1} \mathcal{C}_1 \ldots \rightsquigarrow_{\sigma_n} \mathcal{C}_n$.*

Theorem 6.1 is then proved by contradiction in three main steps. First, Theorem 6.2 serves as a key result for proving that if $\mathcal{C}$ is satisfiable, then there exists a solution θ where messages from P_1 and P_2 are not mixed up. This is obtained by observing that the simplification rules enable us to build θ step by step through unification of subterms of P_1 and P_2. Now, since unification between pairs is forbidden, the rules R_2 and R_3 only involve subterms that convey the same tag, *i.e* subterms issued from the same protocol. Second, conditions 1 and 2 ensure that for any solution θ of $\mathcal{C}$, the critical names of $\overline{n}$ do not appear in plaintext in $\mathcal{C}\theta$. Third, thanks to the two previous steps, we prove that β-tagged terms (intuitively messages from P_2) are not useful for deducing α-tagged terms. For this, we establish that $T \vdash u$ implies $\overline{T} \vdash \overline{u}$ where $\overline{\cdot}$ is a function that keep the terms issued from P_1 unchanged and projects the terms issued from P_2 on a public name. The proof is done by induction on the proof tree witnessing $T \vdash u$. It requires in particular the introduction of a new locality lemma for deduction of ground terms (see Exercise 37). Then, we deduce that, removing from $\mathcal{C}$ all constraints inherited from P_2 and all β-tagged terms, we obtain a satisfiable constraint $\mathcal{C}'$ that is associated to a scenario of P_1.

6.2.3 Applications

Security protocols can be analysed using several existing tools, *e.g.* [?, ?]. The security of a protocol P is however guaranteed provided that no other protocols share any of the private data of P. Our result shows that, once the security of an isolated protocol has been established, this protocol can be safely executed in environments that may use some common data provided that a tag is added each time a party performs an encryption.

For security reasons however, most protocols actually make use of different keys. In this section, we provide a simple criteria for safely composing protocols that share keys. This would allow to save both memory (for storing keys) and time since generating keys is time consuming in particular in the case of public key encryption. For example, the SSL protocol should contain the name "ssl2.0" in any of its encrypted messages. This would ensure that no armful interaction can occur with any other protocols even if they share some data with the SSL protocol, provided that these other protocols are also tagged. In other words, to avoid armful interaction between protocols, one should simply use a tagged version of them.

There are also situations were running different protocols with common keys already occur. We provide three examples of such cases.

- It is often the case that several versions of a protocol can be used concurrently. In this case and for backward compatibility reasons, the same keys can be used in the different versions of the protocol, which may lead to potentially dangerous interactions.

- When encrypting emails the same public key can be used for two distinct encryption protocols: the PGP protocol (Pretty Good Privacy) and its open source version OpenPGP. Note that the PGP protocol contains also other sub-protocols such as digitally signing message, all sharing the same public-key infrastructure.

- In the slightly different context of APIs, J. Clulow [?] discovered an attack when the VISA PIN verification values (PVV) protocol and the IBM CCA API share the same verification key.

6.3 From One Session to Many

In this section we briefly present a protocol transformation which maps a protocol that is secure for a single session to a protocol that is secure for an unbounded number of sessions. This is another kind of parallel composition. Moreover, this provides an effective strategy to design secure protocols: (i) design a protocol intended to be secure for one protocol session (this can be efficiently verified with existing automated tools); (ii) apply our transformation and obtain a protocol which is secure for an unbounded number of sessions.

6.3.1 Protocol Transformation

Suppose that Π is a protocol between k participants $A_1, \ldots, A_k$. Our transformation adds to Π a preamble in which each participant sends a freshly generated nonce N_i together with his identity to all other participants. This allows each participant to compute a dynamic, session dependent *tag* $\langle A_1, N_1\rangle, \ldots, \langle A_k, N_k\rangle$ that will be used to tag each encryption and signature in Π. Our transformation is surprisingly simple and does not require any cryptographic protection of the preamble. Intuitively, the security relies on the fact that the participant A_i decides on a given tag for a given session which is ensured to be fresh as it contains his own freshly generated nonce N_i. The transformation is computationally light as it does not add any cryptographic application; it may merely increase the size of messages to be encrypted or signed. The transformation applies to a large class of protocols, which may use symmetric and asymmetric encryption, digital signature and hash function.

We may note that, *en passant*, we identify a class of tagged protocols for which security is decidable for an unbounded number of sessions. This directly follows from our main result as it stipulates that verifying security for a single protocol session is sufficient to conclude security for an unbounded number of sessions.

Given an input protocol Π, our transformation will compute a new protocol $\tilde{\Pi}$ which consists of two phases. During the first phase, the protocol participants try to agree on some common, dynamically generated, session identifier τ. For this, each participant sends a freshly generated nonce N_i together with his identity A_i to all other participants. (Note that if broadcast is not practical or if not all identities are known to each participant, the message can be sent to some of the participants who forward the message.) At the end of this preamble, each participant computes a session identifier: $\tau = \langle\langle A_1, N_1\rangle, \ldots, \langle A_k, N_k\rangle\rangle$. Note that an active attacker may interfere with this initialisation phase and may intercept and replace some of the nonces. Hence, the protocol participants do not necessarily agree on the same session identifier τ after this preamble. In fact, each participant computes his own session identifier, say τ_j. During the second phase, each participant j executes the original protocol in which the dynamically computed identifier is used for tagging each application of a cryptographic primitive. In this phase, when a participant opens an encryption, he will check that the tag is in accordance with the nonces he received during the initialisation phase. In particular he can test the presence of his own nonce.

The transformation, using the informal Alice-Bob notation, is described below and relies on the tagging operation that is formally defined in Definition **??**.

$$\Pi = \left\{ \begin{array}{ll} A_{i_1} \to A_{j_1} : & m_1 \\ \vdots & \\ A_{i_\ell} \to A_{j_\ell} : & m_\ell \end{array} \right. \qquad \tilde{\Pi} = \left\{ \begin{array}{ll} \text{Phase 1} & \text{Phase 2} \\ A_1 \to All : \ \langle A_1, N_1\rangle & A_{i_1} \to A_{j_1} : \ [m_1]_\tau \\ \vdots & \vdots \\ A_k \to All : \ \langle A_k, N_k\rangle & A_{i_\ell} \to A_{j_\ell} : \ [m_\ell]_\tau \\ \text{where } \tau = \langle \mathsf{tag}_1, \ldots, \mathsf{tag}_k\rangle \text{ with } \mathsf{tag}_i = \langle A_i, N_i\rangle & \end{array} \right.$$

Note that, the Alice-Bob notation only represents what happens in a normal execution, *i.e.* with no intervention of the attacker. Of course, in such a situation, the participants agree on the same session identifier τ used in the second phase.

A *k-tag* is a term $\langle\langle a_1, v_1\rangle, \ldots, \langle a_k, v_k\rangle\rangle$ where each a_i is a name of an agent and each v_i is a term. Let u be a term and tag be a k-tag. The k-tagging of u with tag, denoted $[u]_{\mathsf{tag}}$, is inductively defined as follows:

$$\begin{array}{lcll} [\langle u_1, u_2\rangle]_{\mathsf{tag}} & = & \langle [u_1]_{\mathsf{tag}}, [u_2]_{\mathsf{tag}}\rangle & \\ [\mathsf{senc}(u_1, u_2)]_{\mathsf{tag}} & = & \mathsf{senc}(\langle \mathsf{tag}, [u_1]_{\mathsf{tag}}\rangle, [u_2]_{\mathsf{tag}}) & \\ [u]_{\mathsf{tag}} & = & u & \text{otherwise} \end{array}$$

6.3.2 Composition Result

We only state the composition result informally.

> “if the compiled protocol admits an attack that may involve several honest and dishonest sessions, then there exists an attack which only requires one honest session of each role (and no dishonest sessions). “

The situation is however slightly more complicated than it may seem at first sight since there is an infinite number of honest sessions, which one would need to verify separately. Actually we can avoid this combinatorial explosion thanks to the following well-known result [**?**]: when verifying secrecy properties it is sufficient to consider one single honest agent (which is allowed to "talk to herself"). Hence we can instantiate all the parameters with the same honest agent name.

Our dynamic tagging is useful to avoid interaction between different sessions of the same role in a protocol execution and allows us for instance to prevent man-in-the-middle attacks. We need also to forbid long-term secrets in plaintext position (even under an encryption). Note that this condition is generally satisfied by protocols and considered as a prudent engineering practice.

6.3.3 Other ways of tagging

We can also considered an alternative, slightly different transformation that does not include the identities in the tag, *i.e.*, the tag is simply the sequence of nonces. In that case we obtain a different result: if a protocol admits an attack then there exists an attack which only requires one (not necessarily honest) session for each role. In this case, we need to additionally check for attacks that involve a session engaged with the attacker. On the example of the Needham-Schroeder protocol the man-in-the-middle attack is not prevented by this weaker tagging scheme. However, the result requires one to also consider one dishonest session for each role, hence including the attack scenario. In both cases, it is important for the tags to be *collaborative*, *i.e.* all participants do contribute by adding a fresh nonce.

6.4 Further Readings

First, the results presented in this chapter are still true in a more general setting [**?**, **?**], *e.g.* asymmetric encryption, authentication properties, ... Some other results have been obtained for a broader class of composition operations [**?**, **?**]. In particular, their results allow sequential composition. This is useful to do refinement. For instance, we may study a protocol where a key is magically shared between two agents. Then, we want to ensure that the security property still hold when a real key establishment protocol is used.

Finally, different kinds of tags have also been considered in [**?**, **?**, **?**]. However these tags are *static* and have a different aim. While our dynamic tagging scheme presented in Section 6.3 avoids confusing messages from different sessions, these static tags avoid confusing different messages inside a same session and do not prevent that a same message is reused in two different sessions. Under some additional assumptions (e.g. no temporary secret, no ciphertext forwarding), several decidability results [**?**, **?**] have been obtained by showing that it is sufficient to consider one session per role. But those results can not deal with protocols such as the Yahalom protocol or protocols which rely on a temporary secret. In the framework we consider here, the question whether such static tags would be sufficient to obtain decidability is still an open question (see [**?**]). In a similar way, static tags have also been used by Heather *et al.* [**?**] to avoid type confusion attacks.

Following the approach presented in this chapter, it has been shown [**?**] that tagging hashes enable to preserve resistance against guessing attacks under parallel composition. This allows one to safely reuse the same passwords for different applications.

A. Datta *et al.* (*e.g.* [**?**]) have also studied secure protocol composition in a broader sense: protocols can be composed in parallel, sequentially or protocols may use other protocols as components. However, they do not provide any syntactic conditions for a protocol P to be safely executed in parallel with other protocols. For any protocol P' that might be executed in parallel, they have to prove that the two protocols P and P' satisfy each other invariants. Their approach is thus rather designed for component-based design of protocols.

6.5 Exercises

Exercice 35 ($\star\star$)
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 for any $j < i$ such that $T_j \vdash u$, Π is also a proof of $T_j \vdash u$.
We say that a proof Π is *simple* if

1. any subproof of Π is left-minimal,
2. a composition rule is not directly followed by a decomposition rule,
3. any term of the form $\langle u_1, u_2\rangle$ obtained by application of a decomposition rule or an axiom rule is followed by a projection rule.

Questions:

1. Let $T_1 = \{a\}$ and $T_2 = \{a, \mathsf{senc}(\langle a, b\rangle, k), k\}$. We have that $T_2 \vdash \langle a, b\rangle$. Give a proof of $T_2 \vdash \langle a, b\rangle$ that is not simple.
2. Give a simple proof of $T_2 \vdash \langle a, b\rangle$.
3. Show that if there exists a proof of $T_i \vdash u$ then there exists a simple proof of it.

Exercice 36 ($\star\star$)
We consider symmetric encryption/decryption and pairing/projection (again no asymmetric encryption). Show that the procedure made up of the rules R_1, R_2, R'_3, and R_f is still complete if we do not perform unification on pairs. This result is stated in Theorem 6.2.
hint: use the notion of simple proof introduced in Exercise 35.

Exercice 37 ($\star$)
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 be a set of ground terms and u be a ground term such that $T \vdash u$. Then, we have that $\mathit{plaintext}(u) \subseteq \mathit{plaintext}(T)$.

Exercice 38 ($\star$)
Apply the transformation described in Section 6.3 on the Needham-Schroeder protocol.

1. Check that the man-in-the-middle attack does not exist anymore.
2. Remove the identity of the agent from the tag. Is this tag sufficient to avoid the man-in-the-middle attack?

Exercice 39 ($\star$)
In this chapter, we consider static tags to compose different protocols, and dynamic tags to compose different sessions coming from the same protocol. Show that dynamic tags, as described in Section 6.3 are not sufficient to compose different protocols.

Part III

Verification in the Computational Setting

Chapter 7

The Computational Model

7.1 Introduction

Since the beginning of public-key cryptography, with the seminal Diffie-Hellman paper [?], many suitable algorithmic problems for cryptography have been proposed and many cryptographic schemes have been designed, together with more or less heuristic proofs of their security. However, as already explained, most of those schemes have thereafter been broken, even when some kind of proofs existed. The simple fact that a cryptographic algorithm withstood cryptanalytic attacks for several years has often been considered as a kind of validation procedure, but some schemes take a long time before being broken. An example is the Chor-Rivest cryptosystem [?, ?], based on the knapsack problem, which took more than 10 years to be totally broken [?], whereas before this attack it was believed to be strongly secure. As a consequence, the lack of attacks at some time should never be considered as a security validation of the proposal.

A completely different paradigm is provided by the concept of "provable" security in the computational model. A significant line of research has tried to provide proofs in the framework of complexity theory (*a.k.a.* "reductionist" security proofs [?]): the proofs provide reductions from a well-studied computational problem (RSA or the discrete logarithm) to an attack against a cryptographic protocol. Adversaries, and any player, are now modeled by probabilistic Turing machines.

At the beginning, people just tried to define the security notions required by actual cryptographic schemes, and then to design protocols which achieve these notions. The techniques were directly derived from the complexity theory, providing polynomial reductions. However, their aim was essentially theoretical. They were indeed trying to minimize the required assumptions on the primitives (one-way functions or permutations, possibly trapdoor, etc) [?, ?, ?, ?] without considering practicality. Therefore, they just needed to design a scheme with polynomial algorithms, and to exhibit polynomial reductions from the basic assumption on the primitive into an attack of the security notion, in an asymptotic way.

However, as just said, such a result has no practical impact on actual security. Indeed, even with a polynomial reduction, one may be able to break the cryptographic protocol within a few hours, whereas the reduction just leads to an algorithm against the underlying problem which requires many years. Therefore, those reductions only prove the security when very huge (and thus maybe unpractical) parameters are in use, under the assumption that no polynomial time algorithm exists to solve the underlying problem.

7.1.1 Exact Security and Practical Security

For more than 10 years now, more efficient reductions have been expected, under the denominations of either "exact security" [?] or "concrete security" [?], which provide more practical security results. The perfect situation is reached when one manages to prove that, from an attack, one can describe an algorithm against the underlying problem, with almost the same success probability within almost the same amount of time. We have then achieved "practical security".

Unfortunately, in many cases, even just provable security is at the cost of an important loss in terms of efficiency for the cryptographic protocol. Thus some models have been proposed, trying to deal with the security of efficient schemes: some concrete objects are identified with ideal (or black-box) ones.

For example, it is by now usual to identify hash functions with ideal random functions, in the so-called "random-oracle model", informally introduced by Fiat and Shamir [**?**], and formalized by Bellare and Rogaway [**?**]. Similarly, block ciphers are identified with families of truly random permutations in the "ideal cipher model" [**?**].

From a more algebraic point of view, another kind of idealization was introduced in cryptography: the black-box group [**?**, **?**]. Here, the group operation is defined by a black-box: a new element necessarily comes from the addition (or the subtraction) of two already known elements. It is by now called the "generic model". It has been more recently extended to bilinear groups [**?**]. Some works [**?**, **?**] even require several ideal models together to provide some validations.

7.2 Security Proofs and Security Arguments

7.2.1 Computational Assumptions

In both symmetric and asymmetric scenarios, many security notions can not be unconditionally guaranteed (whatever the computational power of the adversary). Therefore, security generally relies on a computational assumption: the existence of one-way functions, or permutations, possibly trapdoor. A one-way function is a function f which anyone can easily compute, but given $y = f(x)$ it is computationally intractable to recover x (or any pre-image of y). A one-way permutation is a bijective one-way function. For encryption, one would like the inversion to be possible for the recipient only: a trapdoor one-way permutation is a one-way permutation for which a secret information (the trapdoor) helps to invert the function on any point.

Given such objects, and thus computational assumptions about the intractability of the inversion without possible trapdoors, we would like that security could be achieved without extra assumptions. The only way to formally prove such a fact is by showing that an attacker against the cryptographic protocol can be used as a sub-part in an algorithm that can break the basic computational assumption.

A partial order therefore exists between computational assumptions (and intractable problems too): if a problem P is more difficult than the problem P' (P' reduces to P, see below) then the assumption of the intractability of the problem P is weaker than the assumption of the intractability of the problem P'. The weaker the required assumption is, the more secure the cryptographic scheme is.

7.2.2 Complexity

In complexity theory, such an algorithm which uses the attacker as a sub-part in a global algorithm is called a reduction. If this reduction is polynomial, we can say that the attack of the cryptographic protocol is at least as hard as inverting the function: if one has a polynomial algorithm to solve the latter problem, one can polynomially solve the former one. In the complexity theory framework, a *polynomial* algorithm is the formalization of *efficiency*.

Therefore, in order to prove the security of a cryptographic protocol, one first needs to make precise the security notion one wants the protocol to achieve: which adversary's goal one wants to be intractable, under which kind of attack. At the beginning of the 1980's, such security notions have been defined for encryption [?] and signature [?, ?], and provably secure schemes have been suggested. However, those proofs had a theoretical impact only, because both the proposed schemes and the reductions were completely unpractical, yet polynomial. The reductions were indeed efficient (*i.e.* polynomial), and thus a polynomial attack against a cryptosystem would have led to a polynomial algorithm that broke the computational assumption. But the latter algorithm, even polynomial, may require hundreds of years to solve a small instance.

For example, let us consider a cryptographic protocol based on integer factoring. Let us assume that one provides a polynomial reduction from the factorization into an attack. But such a reduction may just lead to a factorization algorithm with a complexity in $2^{25}k^{10}$, where k is the bit-size of the integer to factor. This indeed contradicts the assumption that no-polynomial algorithm exists for factoring. However, on a 1024-bit number ($k = 2^{10}$), it provides an algorithm that requires 2^{125} basic operations, which is much more than the complexity of the best current algorithm, such as NFS [?], which needs less than 2^{100} (see Section 8.2). Therefore, such a reduction would just be meaningful for numbers above 4096 bits (since with $k = 2^{12}$, $2^{145} < 2^{149}$, where 2^{149} is the estimate effort for factoring a 4096-bit integer with the best algorithm.) Concrete examples are given later.

7.2.3 Practical Security

Moreover, most of the proposed schemes were unpractical as well. Indeed, the protocols were polynomial in time and memory, but not efficient enough for practical implementation.

For a few years, people have tried to provide both practical schemes, with practical reductions and exact complexity, which prove the security for realistic parameters, under a well-defined assumption: exact reduction in the standard model (which means in the complexity-theoretic framework). For example, under the assumption that a 1024-bit integer cannot be factored with less than 2^{70} basic operations, the cryptographic protocol cannot be broken with less than 2^{60} basic operations. We will see such an example later.

Unfortunately, as already remarked, practical or even just efficient reductions in the standard model can rarely be conjugated with practical schemes. Therefore, one needs to make some hypotheses on the adversary: the attack is generic, independent of the actual implementation of some objects

- hash functions, in the "random-oracle model";
- symmetric block ciphers, in the "ideal-cipher model";
- algebraic groups, in the "generic model".

The "random-oracle model" was the first to be introduced in the cryptographic community [?, ?], and has already been widely accepted. By the way, flaws have been shown in the "generic model" [?] on practical schemes, and the "random-oracle model" is not equivalent to the standard one either. Several gaps have already been exhibited [?, ?, ?]. However, all the counter-examples in the random-oracle model are pathological, counter-intuitive and not natural. Therefore, in the sequel, we focus on security analyses in this model, for real and natural constructions. A security proof in the random-oracle model will at least give a strong argument in favor of the security of the scheme. Furthermore, proofs in the random-oracle model under a weak computational assumption may be of more practical interest than proofs in the standard model under a strong computational assumption.

7.2.4 The Random-Oracle Model

As said above, efficiency rarely meets with provable security. More precisely, none of the most efficient schemes in their category have been proven secure in the standard model. However, some of them admit security validations under ideal assumptions: the random-oracle model is the most widely accepted one.

Many cryptographic schemes use a hash function $\mathcal{H}$ (such as MD5 [?], SHA-1 [?] or more recent functions). This use of hash functions was originally motivated by the wish to sign long messages with a single short signature. In order to achieve *non-repudiation*, a minimal requirement on the hash function is the impossibility for the signer to find two different messages providing the same hash value. This property is called *collision-resistance*.

It was later realized that hash functions were an essential ingredient for the security of, first, signature schemes, and then of most cryptographic schemes. In order to obtain security arguments, while keeping the efficiency of the designs that use hash functions, a few authors suggested using the hypothesis that $\mathcal{H}$ behaves like a random function. First, Fiat and Shamir [?] applied it heuristically to provide a signature scheme "as secure as" factorization. Then, Bellare and Rogaway [?, ?, ?] formalized this concept for cryptography, and namely for signature and public-key encryption.

In this model, the so-called "random-oracle model", the hash function can be formalized by an oracle which produces a truly random value for each new query. Of course, if the same query is asked twice, identical answers are obtained. This is precisely the context of relativized complexity theory with "oracles," hence the name.

About this model, no one has ever been able to provide a convincing contradiction to its practical validity, but just theoretical counter-examples on either clearly wrong designs for practical purpose [?], or artificial security notions [?, ?]. Therefore, this model has been strongly accepted by the community, and is considered as a good one, in which security analyses give a good taste of the actual security level. Even if it does not provide a formal proof of security (as in the standard model, without any ideal assumption), it is argued that proofs in this model ensure security of the overall design of the scheme provided that the hash function has no weakness, hence the name "security arguments".

This model can also be seen as a restriction on the adversary's capabilities. Indeed, it simply means that the attack is generic without considering any particular instantiation of the hash functions. Therefore, an actual attack would necessarily use a weakness or a specific feature of the hash function. The replacement of the hash function by another one would rule out this attack.

On the other hand, assuming the tamper-resistance of some devices, such as smart cards, the random-oracle model is equivalent to the standard model, which simply requires the existence of pseudo-random functions [?, ?].

As a consequence, almost all the standards bodies by now require designs provably secure, at least in that model, thanks to the security validation of very efficient protocols.

7.2.5 The General Framework

Before going into more details of this kind of proofs, we would like to insist on the fact that in the current general framework, we model all the players by (interactive) probabilistic polynomial Turing machines, and we give the adversary complete access to the cryptographic primitive, but as a black-box. It can ask any query of its choice, and the box always answers correctly, in constant time. Such a model does not consider timing attacks [?], where the adversary tries to extract the secrets from the computational time. Some other attacks analyze the electrical energy required by a computation to get the secrets [?], or to make the primitive fail on some computation [?, ?]. They are not captured either by this model.

Chapter 8

Provable Security

8.1 Security Notions

In this section we describe more formally what a signature scheme and an encryption scheme are. Moreover, we make precise their goals, and thus the security notions one wants the schemes to achieve. This is the first imperative step towards provable security.

8.1.1 Public-Key Encryption

The aim of a public-key encryption scheme is to allow anybody who knows the public key of Alice to send her a message that she will be the only one able to recover, granted her private key.

8.1.1.1 Definitions

A public-key encryption scheme $\mathsf{S} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ is defined by the three following algorithms:

- The *key generation algorithm* $\mathcal{K}$. On input 1^k where k is the security parameter, the algorithm $\mathcal{K}$ produces a pair $(\mathsf{pk}, \mathsf{sk})$ of matching public and private keys. Algorithm $\mathcal{K}$ is probabilistic.

- The *encryption algorithm* $\mathcal{E}$. Given a message m and a public key pk, $\mathcal{E}$ produces a ciphertext c of m. This algorithm may be probabilistic. In the latter case, we write $\mathcal{E}_{\mathsf{pk}}(m; r)$ where r is the random input to $\mathcal{E}$.

- The *decryption algorithm* $\mathcal{D}$. Given a ciphertext c and the private key sk, $\mathcal{D}_{\mathsf{sk}}(c)$ gives back the plaintext m. This algorithm is necessarily deterministic.

8.1.1.2 Security Notions

The **goals** of the adversary may be various. The first common security notion that one would like for an encryption scheme is *one-wayness* (OW): with just public data, an attacker cannot get back the whole plaintext of a given ciphertext. More formally, this means that for any adversary $\mathcal{A}$, its success in inverting $\mathcal{E}$ without the private key should be negligible over the probability space $\mathbf{M} \times \Omega$, where $\mathbf{M}$ is the message space and Ω is the space of the random coins r used for the encryption scheme, and the internal random coins of the adversary:

$$\mathsf{Succ}^{\mathsf{ow}}_{\mathsf{S}}(\mathcal{A}) = \Pr_{m,r}[(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathcal{K}(1^k) : \mathcal{A}(\mathsf{pk}, \mathcal{E}_{\mathsf{pk}}(m; r)) = m].$$

However, many applications require more from an encryption scheme, namely the *semantic security* (IND) [?], *a.k.a. polynomial security/indistinguishability of encryptions*: if the attacker has some information about the plaintext, for example that it is either "yes" or "no" to a crucial query, any adversary should not learn more with the view of the ciphertext. This security notion requires computational impossibility to distinguish between two messages, chosen by the adversary, which one has been encrypted, with a probability significantly better than one half: its advantage $\mathsf{Adv}^{\mathsf{ind}}_{\mathsf{S}}(\mathcal{A})$, formally defined as

$$2 \times \Pr_{b,r}\left[\begin{array}{l}(\mathsf{pk},\mathsf{sk}) \leftarrow \mathcal{K}(1^k), (m_0, m_1, s) \leftarrow \mathcal{A}_1(\mathsf{pk}),\\ c = \mathcal{E}_{\mathsf{pk}}(m_b; r) : \mathcal{A}_2(m_0, m_1, s, c) = b\end{array}\right] - 1,$$

where the adversary $\mathcal{A}$ is seen as a 2-stage attacker $(\mathcal{A}_1, \mathcal{A}_2)$, should be negligible.

A later notion is *non-malleability* (NM) [?]. To break it, the adversary, given a ciphertext, tries to produce a new ciphertext such that the plaintexts are meaningfully related. This notion is stronger than the above semantic security, but it is equivalent to the latter in the most interesting scenario [?] (the CCA attacks, see below). Therefore, we will just focus on one-wayness and semantic security.

On the other hand, an attacker can play many kinds of attacks, according to the **available information**: since we are considering asymmetric encryption, the adversary can encrypt any plaintext of its choice, granted the public key, hence the *chosen-plaintext attack* (CPA). It may furthermore have access to additional information, modeled by partial or full access to some oracles:

- A validity-checking oracle which, on input a ciphertext c, answers whether it is a valid ciphertext or not. Such a weak oracle, involved in the so-called reaction attacks [?] or *Validity-Checking Attack* (VCA), had been enough to break some famous encryption schemes [?, ?].

- A plaintext-checking oracle which, on input a pair (m, c), answers whether c encrypts the message m. This attack has been termed the *Plaintext-Checking Attack* (PCA) [?].

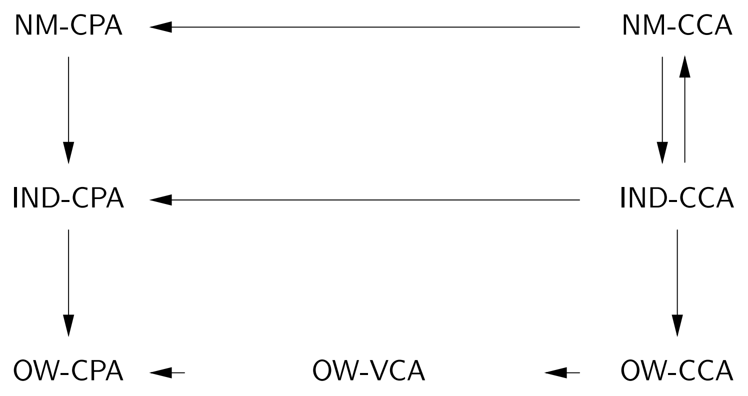
- The decryption oracle itself, which on any ciphertext answers the corresponding plaintext. There is of course the natural restriction not to ask the challenge ciphertext to that oracle.

For all these oracles, access may be restricted as soon as the challenge ciphertext is known, the attack is thus said *non-adaptive* since oracle queries cannot depend on the challenge ciphertext, while they depend on previous answers. On the opposite, access can be unlimited and attacks are thus called *adaptive attacks* (w.r.t. the challenge ciphertext). This distinction has been widely used for the chosen-ciphertext attacks, for historical reasons: the *non-adaptive chosen-ciphertext attacks* (CCA1) [?], *a.k.a.* lunchtime attacks, and *adaptive chosen-ciphertext attacks* (CCA2) [?]. The latter scenario which allows adaptively chosen ciphertexts as queries to the decryption oracle is definitely the strongest attack, and will be named the *chosen-ciphertext attack* (CCA).

Furthermore, multi-user scenarios can be considered where related messages are encrypted under different keys to be sent to many people (*e.g.* broadcast of encrypted data). This may provide many useful data for an adversary. For example, RSA is well-known to be weak in such a scenario [?, ?], namely with a small encryption exponent, because of the Chinese Remainders Theorem. But once again, semantic security has been shown to be the appropriate security level, since it automatically extends to the multi-user setting: if an encryption scheme is semantically secure in the classical sense, it is also semantically secure in multi-user scenarios, against both passive [?] and active [?] adversaries.

A general study of these security notions and attacks was conducted in [?], we therefore refer the reader to this paper for more details. See also the summary diagram on Figure 8.1. However, we can just review the main scenarios we will consider in the following:

- one-wayness under chosen-plaintext attacks (OW-CPA) – where the adversary wants to recover the whole plaintext from just the ciphertext and the public key. This is the weakest scenario.



OW	– One-Wayness	CPA	– Chosen-Plaintext Attack
IND	– Indistinguishability (*a.k.a.* Semantic Security)	VCA	– Validity-Checking Attack (*a.k.a.* Reaction Attack)
NM	– Non-Malleability	CCA	– Chosen-Ciphertext Attack

Figure 8.1: Relations between the Security Notions for Asymmetric Encryption

- semantic security under adaptive chosen-ciphertext attacks (IND-CCA) – where the adversary just wants to distinguish which plaintext, between two messages of its choice, has been encrypted, while it can ask any query it wants to a decryption oracle (except the challenge ciphertext). This is the strongest scenario one can define for encryption (still in our general framework.) Thus, this is our goal when we design a cryptosystem.

8.1.2 Digital Signature Schemes

Digital signature schemes are the electronic version of handwritten signatures for digital documents: a user's signature on a message m is a string which depends on m, on public and secret data specific to the user and —possibly— on randomly chosen data, in such a way that anyone can check the validity of the signature by using public data only. The user's public data are called the *public key*, whereas his secret data are called the *private key*. The intuitive security notion would be the impossibility to forge user's signatures without the knowledge of his private key. In this section, we give a more precise definition of signature schemes and of the possible attacks against them (most of those definitions are based on [**?**]).

8.1.2.1 Definitions

A signature scheme $\mathsf{S} = (\mathcal{K}, \mathcal{S}, \mathcal{V})$ is defined by the three following algorithms:

- The *key generation algorithm* $\mathcal{K}$. On input 1^k, which is a formal notation for a machine with running time polynomial in k (1^k is indeed k in basis 1), the algorithm $\mathcal{K}$ produces a pair $(\mathsf{pk}, \mathsf{sk})$ of matching public and private keys. Algorithm $\mathcal{K}$ is probabilistic. The input k is called the security parameter. The sizes of the keys, or of any problem involved in the cryptographic scheme, will depend on it, in order to achieve an appropriate security level (the expected minimal time complexity of any attack).

- The *signing algorithm* $\mathcal{S}$. Given a message m and a pair of matching public and private keys $(\mathsf{pk}, \mathsf{sk})$, $\mathcal{S}$ produces a signature σ. The signing algorithm might be probabilistic.

- The *verification algorithm* $\mathcal{V}$. Given a signature σ, a message m and a public key pk, $\mathcal{V}$ tests whether σ is a valid signature of m with respect to pk. In general, the verification algorithm need not be probabilistic.

8.1.2.2 Forgeries and Attacks

In this subsection, we formalize some security notions which capture the main practical situations. On the one hand, the **goals** of the adversary may be various:

- Disclosing the private key of the signer. It is the most serious attack. This attack is termed *total break*.
- Constructing an efficient algorithm which is able to sign messages with good probability of success. This is called *universal forgery*.
- Providing a new message-signature pair. This is called *existential forgery*. The corresponding security level is called *existential unforgeability* (EUF).

In many cases the latter forgery, the *existential forgery*, is not dangerous because the output message is likely to be meaningless. Nevertheless, a signature scheme which is existentially forgeable does not guarantee by itself the identity of the signer. For example, it cannot be used to certify randomly looking elements, such as keys. Furthermore, it cannot formally guarantee the non-repudiation property, since anyone may be able to produce a message with a valid signature.

On the other hand, various **means** can be made available to the adversary, helping it into its forgery. We focus on two specific kinds of attacks against signature schemes: the *no-message attacks* and the *known-message attacks* (KMA). In the former scenario, the attacker only knows the public key of the signer. In the latter, the attacker has access to a list of valid message-signature pairs. According to the way this list was created, we usually distinguish many subclasses, but the strongest is definitely the *adaptive chosen-message attack* (CMA), where the attacker can ask the signer to sign any message of its choice, in an adaptive way: it can adapt its queries according to previous answers.

When signature generation is not deterministic, there may be several signatures corresponding to a given message. And then, some notions defined above may become ambiguous [**?**]. First, in known-message attacks, an existential forgery becomes the ability to forge a fresh message/signature pair that has not been obtained during the attack. There is a subtle point here, related to the context where several signatures may correspond to a given message. We actually adopt the stronger rule that the attacker needs to forge the signature of message, whose signature was not queried. The more liberal rule, which makes the attacker successful when it outputs a second signature of a given message different from a previously obtained signature of the same message, is called *malleability*, while the corresponding security level is called *non-malleability* (NM). Similarly, in adaptive chosen-message attacks, the adversary may ask several times the same message, and each new answer gives it some information. A slightly weaker security model, by now called *single-occurrence adaptive chosen-message attack* (SO-CMA), allows the adversary at most one signature query for each message. In other words the adversary cannot submit the same message twice for signature.

When one designs a signature scheme, one wants to computationally rule out at least existential forgeries, or even achieve non-malleability, under adaptive chosen-message attacks. More formally, one wants that the success probability of any adversary $\mathcal{A}$ with a reasonable time is small, where

$$\mathsf{Succ}_{\mathsf{S}}^{\mathsf{euf}}(\mathcal{A}) = \Pr\left[\ (\mathsf{pk},\mathsf{sk}) \leftarrow \mathcal{K}(1^k), (m,\sigma) \leftarrow \mathcal{A}^{\mathcal{S}_{\mathsf{sk}}}(\mathsf{pk}) : \mathcal{V}(\mathsf{pk},m,\sigma) = 1\ \right].$$

We remark that since the adversary is allowed to play an adaptive chosen-message attack, the signing algorithm is made available, without any restriction, hence the oracle notation $\mathcal{A}^{\mathcal{S}_{\mathsf{sk}}}$. Of course, in its answer, there is the natural restriction that, at least, the returned message-signature has not been obtained from the signing oracle $\mathcal{S}_{\mathsf{sk}}$ itself (non-malleability) or even the output message has not been queried (existential unforgeability).

8.2 The Computational Assumptions

There are two major families in number theory-based public-key cryptography:

1. the schemes based on integer factoring, and on the RSA problem [?];

2. the schemes based on the discrete logarithm problem, and on the Diffie-Hellman problems [?], in any "suitable" group. The first groups in use were cyclic subgroups of $\mathbb{Z}_p^\star$, the multiplicative group of the modular quotient ring $\mathbb{Z}_p = \mathbb{Z}/p\mathbb{Z}$. But many schemes are now converted on cyclic subgroups of elliptic curves, or of the Jacobian of hyper-elliptic curves, with namely the so-called ECDSA [?], the US Digital Signature Standard [?] on elliptic curves. The pairing-based cryptography also works in groups (elliptic curves) where the discrete logarithm problem is difficult, but a bilinear map provides variable intractability levels for the Diffie-Hellman problems, according to the actual curve.

8.2.1 Integer Factoring and the RSA Problem

The most famous intractable problem is factorization of integers: while it is easy to multiply two prime integers p and q to get the product $n = p \cdot q$, it is not simple to decompose n into its prime factors p and q.

Currently, the most efficient algorithm is based on sieving on number fields. The Number Field Sieve (NFS) method [?] has a super-polynomial, but sub-exponential, complexity in $\mathcal{O}(\exp((1.923 + o(1))(\ln n)^{1/3}(\ln \ln n)^{2/3}))$. It has been used to establish the main records, in august 1999 (by factoring a 155-digit integer, 512-bit long [?]) and in december 2009 (by factoring a 232-digit integer, 768-bit long [?]). The factored numbers, called RSA-155 and RSA-768, were taken from the "RSA Challenge List", which is used as a yardstick for the security of the RSA cryptosystem (see later). The latter is used extensively in hardware and software to protect electronic data traffic such as in the SSL (Security Sockets Layer) Handshake Protocol.

```
RSA-155 =
  109417386415705274218097073220403576120037329454492\
  599091384213147634998428893478471799725789126733249\
  62575289978183379707653724402714674353159335433897
= 102639592829741105772054196573991675900\
  716567808038066803341933521790711307779
* 106603488380168454820927220360012878679\
  207958575989291522270608237193062808643

RSA-232 =
  1230186684530117755130494958384962720772853569595334792197\
  3224521517264005072636575187452021997864693899564749427740\
  6384592519255732630345373154826850791702612214291346167042\
  921431160222124047927473779408066535141959745985690214341 3
= 3347807169895689878604416984821269081770479498371376856891\
  2431388982883793878002287614711652531743087737814467999489
* 3674604366679959042824463379962795263227915816434308764267\
  6032283815739666511279233373417143396810270092798736308917
```

Unfortunately, integer multiplication just provides a one-way function, without any possibility to invert the process. No information is known to make factoring easier. However, some algebraic structures are based on the factorization of an integer n, where some computations are difficult without the factorization of n, but easy with it: the finite quotient ring $\mathbb{Z}_n$ which is isomorphic to the product ring $\mathbb{Z}_p \times \mathbb{Z}_q$ if $n = p \cdot q$.

Year	Required Complexity	modulus bitlength
before 2000	64	768
before 2010	80	1024
before 2020	112	2048
before 2030	128	3072
	192	7680
	256	15360

Figure 8.2: Bitlength of RSA Moduli

For example, the e-th power of any element x can be easily computed using the *square-and-multiply* method. It consists in using the binary representation of the exponent $e = e_k e_{k-1} \dots e_0$, computing the successive 2 powers of x ($x^{2^0}, x^{2^1}, \dots, x^{2^k}$) and eventually to multiply altogether the ones for which $e_i = 1$. However, to compute e-th roots, it seems that one requires to know an integer d such that $ed = 1 \bmod \varphi(n)$, where $\varphi(n)$ is the totient Euler function which denotes the cardinality of the multiplicative subgroup $\mathbb{Z}_n^\star$ of $\mathbb{Z}_n$. In the particular case where $n = pq$, $\varphi(n) = (p-1)(q-1)$. And therefore, $ed - 1$ is a multiple of $\varphi(n)$ which is equivalent to the knowledge of the factorization of n [**?**]. In 1978, Rivest, Shamir and Adleman [**?**] defined the following problem:

> **The RSA Problem.** Let $n = pq$ be the product of two large primes of similar size and e an integer relatively prime to $\varphi(n)$. For a given $y \in \mathbb{Z}_n^\star$, compute the modular e-th root x of y (*i.e.* $x \in \mathbb{Z}_n^\star$ such that $x^e = y \bmod n$.)

The Euler function can be easily computed from the factorization of n, since for any $n = \prod p_i^{v_i}$,

$$\varphi(n) = n \times \prod \left(1 - \frac{1}{p_i}\right).$$

Therefore, with the factorization of n (the trapdoor), the RSA problem can be easily solved. But nobody knows whether the factorization is required, and how to do without it either:

> **The RSA Assumption.** For any product of two primes, $n = pq$, large enough, the RSA problem is intractable (presumably as hard as the factorization of n).

More precisely, the intractability level is assumed as depicted on Figure 8.2. Until 2000, a security level of 2^{64} was considered enough since the computational power that an adversary could collect within a reasonable time was bounded by 2^{64}. Since 2010, 2^{80} is no longer considered enough, and thus 2048-bit RSA moduli are recommended. However, this is under the assumption that the RSA problem can be efficiently reduced to an attack, without any loss (practical security).

8.2.2 The Discrete Logarithm and the Diffie-Hellman Problems

The setting is quite general: one is given

- a cyclic group $\mathcal{G}$ of prime order q (such as the finite group $(\mathbb{Z}_q, +)$, a subgroup of $(\mathbb{Z}_p^\star, \times)$ for $q | p-1$, of an elliptic curve, etc);
- a generator $\mathbf{g}$ (*i.e.* $\mathcal{G} = \langle \mathbf{g} \rangle$).

We note in bold (such as $\mathbf{g}$) any element of the group $\mathcal{G}$, to distinguish it from a scalar $x \in \mathbb{Z}_q$. But such a $\mathbf{g}$ could be an element in $\mathbb{Z}_p^\star$ or a point of an elliptic curve, according to the setting. Above, we talked about a "suitable" group $\mathcal{G}$. In such a group, some of the following problems have to be hard to solve (using the additive notation).

- the **Discrete Logarithm** problem (**DL**): given $\mathbf{y} \in \mathcal{G}$, compute $x \in \mathbb{Z}_q$ such that $\mathbf{y} = x \cdot \mathbf{g} = \mathbf{g} + \ldots + \mathbf{g}$ (x times), then one writes $x = \log_{\mathbf{g}} \mathbf{y}$.

- the **Computational Diffie-Hellman** problem (**CDH**): given two elements in the group $\mathcal{G}$, $\mathbf{a} = a \cdot \mathbf{g}$ and $\mathbf{b} = b \cdot \mathbf{g}$, compute $\mathbf{c} = ab \cdot \mathbf{g}$. Then one writes $\mathbf{c} = \mathbf{DH}(\mathbf{a}, \mathbf{b})$.

- the **Decisional Diffie-Hellman** Problem (**DDH**): given three elements in the group $\mathcal{G}$, $\mathbf{a} = a \cdot \mathbf{g}$, $\mathbf{b} = b \cdot \mathbf{g}$ and $\mathbf{c} = c \cdot \mathbf{g}$, decide whether $\mathbf{c} = \mathbf{DH}(\mathbf{a}, \mathbf{b})$ (or equivalently, whether $c = ab \bmod q$).

It is clear that they are sorted from the strongest problem to the weakest one. Furthermore, one may remark that they all are "random self-reducible", which means that any instance can be reduced to a uniformly distributed instance: for example, given a specific element $\mathbf{y}$ for which one wants to compute the discrete logarithm x in basis $\mathbf{g}$, one can choose a random $z \in \mathbb{Z}_q$, and compute $\mathbf{z} = z \cdot \mathbf{y}$. The element $\mathbf{z}$ is therefore uniformly distributed in the group, and the discrete logarithm $\alpha = \log_{\mathbf{g}} \mathbf{z}$ leads to $x = \alpha / z \bmod q$. As a consequence, there are only average complexity cases. Thus, the ability to solve a problem for a non-negligible fraction of instances in polynomial time is equivalent to solve any instance in expected polynomial time.

A variant of the Diffie-Hellman problem has been defined by Tatsuaki Okamoto and David Pointcheval [**?**], the so-called *Gap Diffie-Hellman Problem* (**GDH**), where one wants to solve the **CDH** problem with an access to a **DDH** oracle. One may easily remark the following properties about above problems: $\mathbf{DL} \geq \mathbf{CDH} \geq \{\mathbf{DDH}, \mathbf{GDH}\}$, where $A \geq B$ means that the problem A is at least as hard as the problem B. However, in practice, no one knows how to solve any of them without breaking the **DL** problem itself. On pairing-friendly elliptic curves, the **DDH** can be easy to decide.

Currently, the most efficient algorithms to solve the latter problem depend on the underlying group. For generic groups (for which no specific algebraic property can be used), algorithms have a complexity in the square root of q, the order of the generator $\mathbf{g}$ [**?**, **?**]. For example, on well-chosen elliptic curves only these algorithms can be used.

However, for subgroups of $\mathbb{Z}_p^\star$, some better techniques can be applied. The best algorithm is based on sieving on number fields, as for the factorization. The General Number Field Sieve method [**?**] has a super-polynomial, but sub-exponential, complexity in $\mathcal{O}(\exp((1.923 + o(1))(\ln p)^{1/3}(\ln \ln p)^{2/3}))$.

For signature applications, one only requires groups where the **DL** problem is hard, whereas encryption needs trapdoor problems and therefore requires groups where some of the **DH**'s problems are also hard to solve.

8.3 Proof Methodology

The actual security proof consists of a reduction of the underlying computational problem to an attack against the cryptographic scheme: if there exist an adversary $\mathcal{A}$ able to win the security game, we can use this adversary $\mathcal{A}$ as a subroutine to solve the computational problem, as shown of Figure 8.3. More precisely, a proof consists in building a simulator. In order to be able to evaluate the success probability of the simulator in solving the computational problem, we have to show that the view of the adversary remains unchanged from the one its has during a real attack.

Until the early 2000's, a proof consisted in exhibiting the simulator, and then to directly analyze the success probability. This analysis was intricate and error-prone. Victor Shoup introduced in [**?**, **?**, **?**] a game-based approach, also revisited by Bellare and Rogaway [**?**], that we will extensively use in these notes.

In this technique, we define a sequence $\mathbf{G}_1$, $\mathbf{G}_2$, etc., of modified attack games starting from the actual security game $\mathbf{G}_0$. Each of the games operates on the same underlying probability

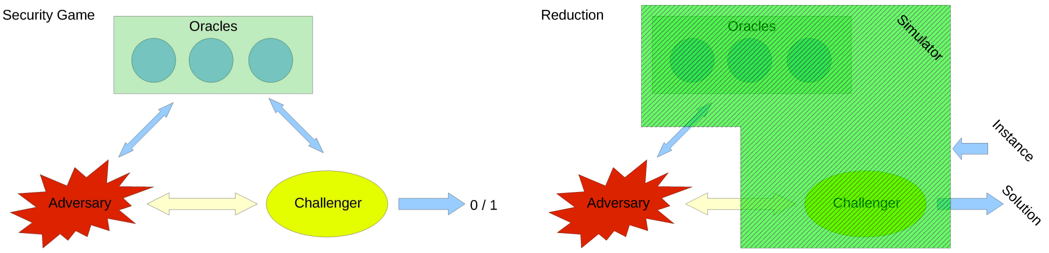


Figure 8.3: Proof by Reduction

space: the public and private keys of the cryptographic scheme, the coin tosses of the adversary $\mathcal{A}$ and the various oracles. Only the rules defining how the view is computed differ from game to game: we modify the behavior of the oracles and of the challenger. The view of the adversary (similar to the trace of an execution in the symbolic model) can be seen as a random variable following a distribution probability $\mathcal{D}_i$ in the game $\mathbf{G}_i$.

Then, several situations can appear:

- the distribution remains perfectly unchanged, then the distance between the two distributions is 0;
- the distribution remains statistically unchanged, then the distance between the two distributions is negligible;
- the distributions are computationally indistinguishable, then a decisional problem is involved;

In these three cases, the probability of any event is almost the same in the two games: the difference is bounded by the distance between the two distributions.

We can also modify the distributions, but in specific cases only: unless an event is raised, the two games run identically. As a consequence, the distributions of outputs of the two games are related by the event. We can indeed apply the following Shoup lemma:

Lemma 8.1 *Let* E_1*,* E_2 *and* F_1*,* F_2 *be events defined on a probability space*

$$\Pr[\mathsf{E}_1 \,|\, \neg\mathsf{F}_1] = \Pr[\mathsf{E}_2 \,|\, \neg\mathsf{F}_2] \textit{ and } \Pr[\mathsf{F}_1] = \Pr[\mathsf{F}_2] = \varepsilon \;\Rightarrow\; |\Pr[\mathsf{E}_1] - \Pr[\mathsf{E}_2]| \leq \varepsilon.$$

Proof : The proof follows from easy computations:

$$\begin{aligned} |\Pr[\mathsf{E}_1] - \Pr[\mathsf{E}_2]| &= |\Pr[\mathsf{E}_1 \,|\, \mathsf{F}_1]\cdot\Pr[\mathsf{F}_1] + \Pr[\mathsf{E}_1 \,|\, \neg\mathsf{F}_1]\cdot\Pr[\neg\mathsf{F}_1] \\ &\quad - \Pr[\mathsf{E}_2 \,|\, \mathsf{F}_2]\cdot\Pr[\mathsf{F}_2] - \Pr[\mathsf{E}_2 \,|\, \neg\mathsf{F}_2]\cdot\Pr[\neg\mathsf{F}_2]| \\ &= |(\Pr[\mathsf{E}_1 \,|\, \mathsf{F}_1] - \Pr[\mathsf{E}_2 \,|\, \mathsf{F}_2])\cdot\varepsilon \\ &\quad + (\Pr[\mathsf{E}_1 \,|\, \neg\mathsf{F}_1] - \Pr[\mathsf{E}_2 \,|\, \neg\mathsf{F}_2])\cdot(1-\varepsilon)| \\ &= |(\Pr[\mathsf{E}_1 \,|\, \mathsf{F}_1] - \Pr[\mathsf{E}_2 \,|\, \mathsf{F}_2])\cdot\varepsilon| \leq \varepsilon. \end{aligned}$$

8.4 Exercises

Exercice 40 (Amplification of RSA)

Let us consider the RSA problem, with modulus n and exponent e, prime to $\varphi(n)$. We say that the algorithm $\mathcal{A}$ is an (ε, t)-adversary against RSA with parameters (n, e) if, within time t, its success probability is greater than ε:

$$\mathsf{Succ}(\mathcal{A}) = \Pr_{x \in \mathbb{Z}_n^\star}[\mathcal{A}(n, e, x^e \bmod n) = x] \geq \varepsilon.$$

Show that if there exists an (ε, t)-adversary $\mathcal{A}$ against RSA with parameters (n, e), then for any $0 < \eta < 1$, there exists an (η, t')-adversary $\mathcal{B}$ against RSA with the same parameters (n, e), within a reasonable time bound t'.

Exercice 41 (Amplification of Diffie-Hellman)
Let us consider the Diffie-Hellman problem in a cyclic group $\mathcal{G}$ of prime order q, with a generator g: given $X = g^x$ and $Y = g^y$, one has to compute $Z = g^{xy}$.

We say that the algorithm $\mathcal{A}$ is an (ε, t)-adversary against DH with parameters $(\mathcal{G}, g)$ if, within time t, its success probability is greater than ε:

$$\mathsf{Succ}(\mathcal{A}) = \Pr_{x,y \in \mathbb{Z}_q^\star}[\mathcal{A}(g^x, g^y) = g^{xy}] \geq \varepsilon.$$

1. Can we use the same amplification method as above to build an (η, t')-adversary for any $0 < \eta < 1$?

   From an instance $(A = g^a, B = g^b)$, let us derive two instances: $(X_i = A^{\alpha_i} g^{u_i}, Y_i = B^{\beta_i} g^{v_i})$, for $i = 0, 1$, with random exponents $\alpha_i, \beta_i, u_i, v_i$. We then run twice our adversary $\mathcal{A}$ on each derived instance.

2. Show that we can detect if $\mathcal{A}$ succeeded on the two instances with negligible error

3. Show that we can detect if $\mathcal{A}$ failed for at least one instance, with negligible error

4. Describe the amplified algorithm B, and estimate the time and the success probability

5. Show that u_i and v_i are important in the randomization: zero values could lead to a wrong algorithm $\mathcal{B}$.

Chapter 9

Public-Key Encryption Schemes

9.1 Introduction

9.1.1 The RSA Encryption Scheme

In their seminal paper [**?**], Rivest, Shamir and Adleman proposed both signature and public-key encryption schemes, thanks to the "trapdoor one-way permutation" property of the RSA function: the generation algorithm produces a large composite number $N = pq$, a public key e, and a private key d such that $e \cdot d = 1 \bmod \varphi(N)$. The encryption of a message m, encoded as an element in $\mathbb{Z}_N^\star$, is simply $c = m^e \bmod N$. This ciphertext can be easily decrypted thanks to the knowledge of d, $m = c^d \bmod N$. Clearly, this encryption is $\mathsf{OW\text{-}CPA}$, relative to the RSA problem. The determinism makes a plaintext-checking oracle useless. Indeed, the encryption of a message m, under a public key pk is always the same, and thus it is easy to check whether a ciphertext c really encrypts m, by re-encrypting it. Therefore the RSA-encryption scheme is $\mathsf{OW\text{-}PCA}$ relative to the RSA problem as well.

Because of this determinism, it cannot be semantically secure: given the encryption c of either m_0 or m_1, the adversary simply computes $c' = m_0^e \bmod N$ and checks whether $c' = c$. Furthermore, with a small exponent e (*e.g.* $e = 3$), any security vanishes under a multi-user attack: given $c_1 = m^3 \bmod N_1$, $c_2 = m^3 \bmod N_2$ and $c_3 = m^3 \bmod N_3$, one can easily compute $m^3 \bmod N_1N_2N_3$ thanks to the Chinese Remainders Theorem, which is exactly m^3 in $\mathbb{Z}$ and therefore leads to an easy recovery of m.

9.1.2 The El Gamal Encryption Scheme

In 1985, El Gamal [**?**] also designed both signature and public-key encryption schemes. The latter is based on the Diffie-Hellman key exchange protocol [**?**]: given a cyclic group $\mathcal{G}$ of order prime q and a generator $\mathbf{g}$, the generation algorithm produces a random element $x \in \mathbb{Z}_q^\star$ as private key, and a public key $\mathbf{y} = x \cdot \mathbf{g}$. The encryption of a message m, encoded as an element $\mathbf{m}$ in $\mathcal{G}$, is a pair $(\mathbf{c} = a \cdot \mathbf{g}, \mathbf{d} = a \cdot \mathbf{y} + \mathbf{m})$, for a random $a \in \mathbb{Z}_q$. This ciphertext can be easily decrypted thanks to the knowledge of x, since

$$a \cdot \mathbf{y} = ax \cdot \mathbf{g} = x \cdot \mathbf{c},$$

and thus $\mathbf{m} = \mathbf{d} - x \cdot \mathbf{c}$. This encryption scheme is well-known to be $\mathsf{OW\text{-}CPA}$ relative to the Computational Diffie-Hellman problem. It is also semantically secure (against chosen-plaintext attacks) relative to the Decisional Diffie-Hellman problem [**?**].

As we have seen above, the expected security level is $\mathsf{IND\text{-}CCA}$, whereas the RSA encryption just reaches $\mathsf{OW\text{-}CPA}$ under the RSA assumption, and the El Gamal encryption achieves $\mathsf{IND\text{-}CPA}$ under the **DDH** assumption. Can we achieve $\mathsf{IND\text{-}CCA}$ for practical encryption schemes?

9.2 The Cramer-Shoup Encryption Scheme

As we will see later, the scheme using hash functions, modeled as random oracles in the security analyses, are definitely the most efficient schemes, but let us first start with a nice variation of the ElGamal encryption scheme, proposed by Cramer and Shoup [**?**], that is both rather efficient and IND-CCA secure under the **DDH** assumption.

9.2.1 Description

For the description of this scheme, we will use the multiplicative notation, as in the original paper: We thus work in a cyclic group $\mathcal{G}$ of order prime q, with two independent generators g_1 and g_2. We will also need a hash function $\mathcal{H}$, that will be assumed to be second-preimage resistant. Then the encryption scheme $\mathsf{CS} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ can be described as follows:

- $\mathcal{K}(1^k)$: produces a random element $z \in \mathbb{Z}_q^\star$, as in the above ElGamal scheme, but also four additional scalars $x_1, x_2, y_1, y_2 \in \mathbb{Z}_q^\star$ as private key. The public key is defined by the values $h = g_1^z$ and $c = g_1^{x_1} g_2^{x_2}$, $d = g_1^{y_1} g_2^{y_2}$. The public key pk is therefore (c, d, h) and the private key sk is (x_1, x_2, y_1, y_2, z).
- $\mathcal{E}_{\mathsf{pk}}(m; r)$: given a message $m \in \mathcal{G}$ and a random scalar $r \in \mathbb{Z}_q$, one computes
$$u_1 = g_1^r, u_2 = g_2^r, e = m \times h^r, v = (cd^\alpha)^r \quad \text{where } \alpha = \mathcal{H}(u_1, u_2, e).$$
- $\mathcal{D}_{\mathsf{sk}}(u_1 \| u_2 \| e \| v)$: thanks to the private key, the decryption algorithm $\mathcal{D}_{\mathsf{sk}}$ first checks the validity of the ciphertext:
$$v \overset{?}{=} u_1^{x_1 + \alpha y_1} u_2^{x_2 + \alpha y_2} \quad \text{where } \alpha = \mathcal{H}(u_1, u_2, e).$$
If the equality holds, the plaintext is $m = e/u_1^z$, otherwise it returns "Reject."

9.2.2 Security Analysis

About this construction, one can prove:

Theorem 9.1 *Let $\mathcal{A}$ be a* CCA-adversary against the semantic security of the above encryption scheme CS. Assume that $\mathcal{A}$ has advantage ε and running time τ and makes q_d queries to the decryption oracle. Then

$$\mathsf{Adv}^{\mathsf{ddh}}(T) \quad \geq \quad \frac{\varepsilon}{2} - \frac{1}{2} \times \mathsf{Succ}^{\mathcal{H}}(T) - \frac{3q_d}{2q},$$

where $T = t + (10 + 4q_d)T_{\mathsf{exp}}$, and T_{exp} the time for one exponentiation.

Proof : In the following we use starred letters ($r^\star$, $u_1^\star$, $u_2^\star$, $e^\star$, $v^\star$, $\alpha^\star$, and $c^\star$) to refer to the challenge ciphertext, whereas unstarred letters (r, u_1, u_2, e, v, α and c) refer to the ciphertext asked to the decryption oracle.

Game $\mathbf{G}_0$: A pair of keys $(\mathsf{pk}, \mathsf{sk})$ is generated using $\mathcal{K}(1^k)$. Adversary A_1 is fed with pk, and outputs a pair of messages (m_0, m_1). Next a challenge ciphertext is produced by flipping a coin b and producing a ciphertext $c^\star = u_1^\star \| u_2^\star \| e^\star \| v^\star$ of m_b. This ciphertext comes from a random $r^\star \overset{R}{\leftarrow} \mathbb{Z}_q$ and $u_1^\star = g_1^r, u_2^\star = g_2^r, e^\star = m_b \times h^r, v^\star = (cd^{\alpha^\star})^r$ where $\alpha^\star = \mathcal{H}(u_1^\star, u_2^\star, e^\star)$. On input $c^\star$, A_2 outputs bit b'. In both stages, the adversary is given additional access to the decryption oracle $\mathcal{D}_{\mathsf{sk}}$. The only requirement is that the challenge ciphertext $c^\star$ cannot be queried from the decryption oracle.

$\mathcal{K}$ Oracle

One generates four random scalars $x_1, x_2, y_1, y_2 \in \mathbb{Z}_q^\star$.

▶Rule GenSK$^{(0)}$

> One chooses a random scalar $z \in \mathbb{Z}_q$, and sets $h = g_1^z$.

The private consists of all the random scalars. The public key is defined by h and $c = g_1^{x_1} g_2^{x_2}$, $d = g_1^{y_1} g_2^{y_2}$.

$\mathcal{D}$ Oracle

Query $\mathcal{D}_{\mathsf{sk}}(u_1 \| u_2 \| e \| v)$:

▶Rule DecM$^{(0)}$

> One computes $m = e/h^z$.

▶Rule CheckV$^{(0)}$

> One computes $\alpha = \mathcal{H}(u_1, u_2, e)$, and $v' = u_1^{x_1+\alpha y_1} u_2^{x_2+\alpha y_2}$.
> If $v \neq v'$ then one returns "Reject."

One returns m.

Challenger

For two messages (m_0, m_1), flip a coin b and set $m^\star = m_b$.

▶Rule Chal−Output$^{(0)}$

> Choose randomly $r^\star$, then set
> $u_1^\star = g_1^{r^\star}$, $u_2^\star = g_2^{r^\star}$, $e^\star = m^\star \times h^{r^\star}$, $v^\star = (cd^{\alpha^\star})^{r^\star}$
> where $\alpha^\star = \mathcal{H}(u_1^\star, u_2^\star, e^\star)$.

Then, output $c^\star = u_1^\star \| u_2^\star \| e^\star \| v^\star$.

Figure 9.1: Formal Simulation of the $\mathsf{IND\text{-}CCA}$ Game against the CS Construction

We denote by S_0 the event $b' = b$ and use a similar notation S_i in any $\mathbf{G}_i$ below. By definition, we have

$$\Pr[\mathsf{S}_0] = \frac{1}{2} + \frac{\varepsilon}{2}. \tag{9.1}$$

Game $\mathbf{G}_1$: In the previous game, we define an event CBad_0 that is raised during a decryption querying $(u_1 \| u_2 \| e \| v)$ if $v = u_1^{x_1+\alpha y_1} u_2^{x_2+\alpha y_2}$, but $u_1 = g_1^{r_1}$ and $u_2 = g_2^{r_2}$ with $r_1 \neq r_2$. In this new game, when this event is raised, one stops the game and outputs a random bit bit b'. Then, clearly,

$$\begin{aligned}
\Pr[\mathsf{CBad}_1] = \Pr[\mathsf{CBad}_0] \qquad & \Pr[\mathsf{S}_1 \,|\, \neg\mathsf{CBad}_1] = \Pr[\mathsf{S}_0 \,|\, \neg\mathsf{CBad}_0] \quad \Pr[\mathsf{S}_1 \,|\, \mathsf{CBad}_1] = \frac{1}{2} \\
|\Pr[\mathsf{S}_1] - \Pr[\mathsf{S}_0]\,| \quad = \quad & |\Pr[\mathsf{S}_1 \,|\, \neg\mathsf{CBad}_1]\Pr[\neg\mathsf{CBad}_1] + \Pr[\mathsf{S}_1 \,|\, \mathsf{CBad}_1]\Pr[\mathsf{CBad}_1] \\
& - \Pr[\mathsf{S}_0 \,|\, \neg\mathsf{CBad}_0]\Pr[\neg\mathsf{CBad}_0] - \Pr[\mathsf{S}_0 \,|\, \mathsf{CBad}_0]\Pr[\mathsf{CBad}_0]\,| \\
= \quad & |\Pr[\mathsf{S}_1 \,|\, \mathsf{CBad}_1] - \Pr[\mathsf{S}_0 \,|\, \mathsf{CBad}_0]\,| \times \Pr[\mathsf{CBad}_0] \\
= \quad & \left|\frac{1}{2} - \Pr[\mathsf{S}_0 \,|\, \mathsf{CBad}_0]\right| \times \Pr[\mathsf{CBad}_0]
\end{aligned}$$

As a consequence,

$$|\Pr[\mathsf{S}_1] - \Pr[\mathsf{S}_0]\,| \leq \frac{1}{2} \times \Pr[\mathsf{CBad}_1] \tag{9.2}$$

Let us now evaluate $\Pr[\mathsf{CBad}_1]$. This event is raised if, knowing the public parameters $c = g_1^{x_1} g_2^{x_2}$ and $d = g_1^{y_1} g_2^{y_2}$ and the challenge ciphertext $u_1^\star = g_1^{r^\star}$, $u_2^\star = g_2^{r^\star}$, and $v^\star = (cd^{\alpha^\star})^{r^\star}$ where

$\alpha^\star = \mathcal{H}(u_1^\star, u_2^\star, e^\star)$, the adversary generates $u_1 = g_1^{r_1}$, $u_2 = g_2^{r_2}$, and v such that $r_1 \neq r_2$ and $v = u_1^{x_1+\alpha y_1} u_2^{x_2+\alpha y_2}$.

If we denote (formally), $g_2 = g_1^{\beta}$, $c = g_1^{\gamma}$, and $d = g_1^{\delta}$, together with $v = g_1^{\mu}$ and $chv = g_1^{\mu^\star}$ we have

$$\begin{array}{rcl} x_1 + \beta x_2 & = & \gamma \\ y_1 + \beta y_2 & = & \delta \\ r^\star(x_1 + \alpha y_1) + \beta r^\star(x_2 + \alpha y_2) & = & \mu^\star \\ r_1(x_1 + \alpha y_1) + \beta r_2(x_2 + \alpha y_2) & = & \mu \end{array}$$

By construction, the third equation is a linear combination of the two first one ($\mu^\star$ can thus be uniquely determined). However, the fourth is linearly independent ($r_1 \neq r_2$), μ is unpredictable, and thus v is so too. As a consequence, even a powerful adversary has no chance to raise CBad_1 but by chance:

$$\Pr[\mathsf{CBad}_0] = \Pr[\mathsf{CBad}_1] \leq \frac{q_d}{q} \tag{9.3}$$

Game G$_2$: In this game we modify the key generation, with

▶Rule GenSK$^{(2)}$

> One chooses two random scalars $z_1, z_2 \in \mathbb{Z}_q$, and sets $h = g_1^{z_1} g_2^{z_2}$.

and thus the decryption procedure becomes:

▶Rule DecM$^{(2)}$

> One computes $m = e / u_1^{z_1} u_2^{z_2}$.

Since we still abort when an acceptable ciphertext satisfies $r_1 \neq r_2$, where $u_1 = g_1^{r_1}$ and $u_2 = g_2^{r_2}$, then necessarily, a decryption only happens when $r = r_1 = r_2$: $u_1^{z_1} u_2^{z_2} = (g_1^{z_1} g_2^{z_2})^r = h^r$. If we formally denote $h = g_1^z$, which means that $z = z_1 + \beta z_2$, then $u_1^{z_1} u_2^{z_2} = (g_1^r)^{z_1 + \beta z_2} = u_1^z$, and thus the decryption is identical to the previous game.

$$\Pr[\mathsf{S}_2] = \Pr[\mathsf{S}_1] \tag{9.4}$$

Game G$_3$: We now want to involve a **DDH** instance. Let us be given a tuple $(g_1, g_2, U = g_1^{r^\star{}_1}, V = g_2^{r^\star{}_2})$, with $r^\star{}_1 = r^\star{}_2$. We use it for the challenge ciphertext generation:

▶Rule Chal−Output$^{(3)}$

> Set $u_1^\star = U$, $u_2^\star = V$, $e^\star = m^\star \times U^{z_1} V^{z_2}$, $v^\star = U^{x_1+\alpha^\star y_1} V^{x_2+\alpha^\star y_2}$
> where $\alpha^\star = \mathcal{H}(u_1^\star, u_2^\star, e^\star)$.

Since we assume that $r^\star = r^\star{}_1 = r^\star{}_2$, $u_1^\star = g_1^{r^\star}$, $u_2^\star = g_2^{r^\star}$, $e^\star = m^\star \times (g_1^{z_1} g_2^{z_2})^{r^\star} = h^{r^\star}$, and $v^\star = g_1^{r^\star x_1 + \alpha^\star r^\star y_1} g_2^{r^\star x_2 + \alpha^\star r^\star y_2} = c^{r^\star} d^{\alpha^\star r^\star}$ where $\alpha^\star = \mathcal{H}(u_1^\star, u_2^\star, e^\star)$. Hence, the challenge generation is identical to the previous game:

$$\Pr[\mathsf{S}_3] = \Pr[\mathsf{S}_2] \tag{9.5}$$

Game G$_4$: Now, we would like to replace the Diffie-Hellman tuple by a random tuple, and use the **DDH** assumption. However, our current simulation is not polynomial: in order to detect the event CBad and then abort, one needs to be able to compute discrete logarithms (or at least to make the **DDH** decision). We thus forget this event, and do not abort anymore, even in case of wrong acceptable ciphertexts. Since $v^\star$ still leads to a linear combination of the exponents of the public key, we can make exactly the same analysis as in game $\mathbf{G}_1$:

$$|\Pr[\mathsf{S}_4] - \Pr[\mathsf{S}_3]| \leq \frac{q_d}{2q} \tag{9.6}$$

Game $\mathbf{G}_5$: We are now given a random input tuple $(g_1, g_2, U = g_1^{r^\star{}_1}, V = g_2^{r^\star{}_2})$. By simply outputting the boolean $b' = b$, we have a distinguisher against the **DDH** problem, within time T, which takes into account the time complexity t of the adversary, and the simulations of the key generation (6 exponentiations), the decryption oracle ($4q_d$ exponentiations) and the challenger (4 exponentiations):

$$|\Pr[\mathsf{S}_5] - \Pr[\mathsf{S}_4]| \leq \mathsf{Adv}^{\mathsf{ddh}}(t + (10 + 4q_d)T_{\mathsf{exp}}) \tag{9.7}$$

Game $\mathbf{G}_6$: In order to be sure that no additional information is revealed about the secret key, we again abort the simulation when the event CBad is detected. Now, $v^\star$ does no longer leads to a linear combination of the exponents of the public key. We can thus take the first part of the analysis of the game $\mathbf{G}_1$:

$$|\Pr[\mathsf{S}_6] - \Pr[\mathsf{S}_5]| \leq \frac{1}{2} \times \Pr[\mathsf{CBad}_6] \tag{9.8}$$

The analysis of the event CBad is a bit more intricate: the adversary knows $c = g_1^{x_1} g_2^{x_2}$ and $d = g_1^{y_1} g_2^{y_2}$ and the challenge ciphertext $u_1^\star = g_1^{r^\star{}_1}$, $u_2^\star = g_2^{r^\star{}_2}$, and $v^\star = g_1^{r^\star{}_1 x_1 + \alpha^\star r^\star{}_1 y_1} g_2^{r^\star{}_2 x_2 + \alpha^\star r^\star{}_2 y_2}$ where $\alpha^\star = \mathcal{H}(u_1^\star, u_2^\star, e^\star)$ and wants to generate $u_1 = g_1^{r_1}$, $u_2 = g_2^{r_2}$, and v such that $r_1 \neq r_2$ (to learn something) and $v = u_1^{x_1 + \alpha y_1} u_2^{x_2 + \alpha y_2}$ (to be accepted).

If we denote (formally), $g_2 = g_1^\beta$, $c = g_1^\gamma$, and $d = g_1^\delta$, together with $v = g_1^\mu$ and $chv = g_1^{\mu^\star}$ we have

$$\begin{aligned} x_1 + \beta x_2 &= \gamma \\ y_1 + \beta y_2 &= \delta \\ r^\star{}_1(x_1 + \alpha^\star y_1) + \beta r^\star{}_2(x_2 + \alpha^\star y_2) &= \mu^\star \\ r_1(x_1 + \alpha y_1) + \beta r_2(x_2 + \alpha y_2) &= \mu \end{aligned}$$

A powerful adversary is able to compute α, β, γ, δ, $r^\star{}_1$, $r^\star{}_2$ and $\mu^\star$. Is goal is to generate $r_1 \neq r_2$ and μ that satisfies the appropriate equation with the variables “x_1, x_2, y_1, y_2 used by the simulator. However, for any $r_1 \neq r_2$, we can show that the following determinant is

$$\begin{vmatrix} 1 & \beta & 0 & 0 \\ 0 & 0 & 1 & \beta \\ r^\star{}_1 & \beta r^\star{}_2 & r^\star{}_1 \alpha^\star & \beta r^\star{}_2 \alpha^\star \\ r_1 & \beta r_2 & r_1 \alpha & \beta r_2 \alpha \end{vmatrix} = \beta^2 \times (r^\star{}_2 - r^\star{}_1) \times (r_2 - r_1) \times (\alpha^\star - \alpha).$$

But for a decryption query (u_1, u_2, e, α), three cases appear:

- $(u_1, u_2, e) = (u_1^\star, chu_2, e^\star)$, then necessarily $chv \neq v$ otherwise the query is exactly the challenge ciphertext, and this is not allowed. Then, we know that the verification check will not pass, the decryption will be rejected;

- $(u_1, u_2, e) \neq (u_1^\star, chu_2, e^\star)$, but $\alpha^\star = \alpha$, which means that $\mathcal{H}(u_1, u_2, e) = \mathcal{H}(u_1^\star, chu_2, e^\star)$, and then the adversary has found a second pre-image for $\mathcal{H}$;

- $(u_1, u_2, e) \neq (u_1^\star, chu_2, e^\star)$, and $\alpha^\star \neq \alpha$. If $r^\star{}_1 \neq r^\star{}_2$, then the above determinant is non-zero, and thus the value μ is unpredictable.

As a consequence, even a powerful adversary has no chance to raise CBad_6 but by chance:

$$\Pr[\mathsf{CBad}_5] = \Pr[\mathsf{CBad}_6] \leq \frac{q_d}{q} + \mathsf{Succ}^{\mathcal{H}}(t + (10 + 4q_d)T_{\mathsf{exp}}) \tag{9.9}$$

Furthermore, in this last game the challenge ciphertext contains $e^\star = m^\star \times U^{z_1}V^{z_2}$, and the public key $h = g_1^{z_1}g_2^{z_2}$ just reveals a linear relation between z_1 and z_2, and any decryption queries for valid ciphertexts where $r_1 = r_2$. On the opposite, since $U = g_1^{r^\star{}_1}$ and $V = g_2^{r^\star{}_2}$ with $r^\star{}_1 \neq r^\star{}_2$, even a powerful adversary has no information about the value of $U^{z_1}V^{z_2}$, hence b' is independent of b:

$$\Pr[\mathsf{S}_6] = \frac{1}{2} \tag{9.10}$$

As a conclusion, one can see that

$$\begin{array}{rcl}
\Pr[\mathsf{S}_0] & = & \frac{1}{2} + \frac{\varepsilon}{2} \\
|\Pr[\mathsf{S}_1] - \Pr[\mathsf{S}_0]| & \leq & \frac{q_d}{2q} \\
|\Pr[\mathsf{S}_2] - \Pr[\mathsf{S}_1]| & = & 0 \\
|\Pr[\mathsf{S}_3] - \Pr[\mathsf{S}_2]| & = & 0 \\
|\Pr[\mathsf{S}_4] - \Pr[\mathsf{S}_3]| & \leq & \frac{q_d}{2q} \\
|\Pr[\mathsf{S}_5] - \Pr[\mathsf{S}_4]| & \leq & \mathsf{Adv}^{\mathsf{ddh}}(T) \\
|\Pr[\mathsf{S}_6] - \Pr[\mathsf{S}_5]| & \leq & \frac{q_d}{2q} + \frac{1}{2} \times \mathsf{Succ}^{\mathcal{H}}(T) \\
\Pr[\mathsf{S}_6] & = & \frac{1}{2}
\end{array}$$

where $T = t + (10 + 4q_d)T_{\mathsf{exp}}$, and thus

$$\frac{\varepsilon}{2} = |\Pr[\mathsf{S}_6] - \Pr[\mathsf{S}_0]| \leq \frac{3q_d}{2q} + \mathsf{Adv}^{\mathsf{ddh}}(T) + \frac{1}{2} \times \mathsf{Succ}^{\mathcal{H}}(T)$$

9.3 A Generic Construction

In [?], Bellare and Rogaway proposed the first generic construction which applies to any trapdoor one-way permutation f onto X.

9.3.1 Description

We need two hash functions $\mathcal{G}$ and $\mathcal{H}$:

$$\mathcal{G} : X \longrightarrow \{0,1\}^n \quad \text{and} \quad \mathcal{H} : \{0,1\}^\star \longrightarrow \{0,1\}^{k_1},$$

where n is the bit-length of the plaintexts, and k_1 a security parameter. Then the encryption scheme $\mathsf{BR} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ can be described as follows:

- $\mathcal{K}(1^k)$: specifies an instance of the function f, and of its inverse f^{-1}. The public key pk is therefore f and the private key sk is f^{-1}.

- $\mathcal{E}_{\mathsf{pk}}(m; r)$: given a message $m \in \{0,1\}^n$, and a random value $r \overset{R}{\leftarrow} X$, the encryption algorithm $\mathcal{E}_{\mathsf{pk}}$ computes

  $$a = f(r), \qquad b = m \oplus \mathcal{G}(r) \quad \text{and} \quad c = \mathcal{H}(m, r),$$

  and outputs the ciphertext $y = a\|b\|c$.

- $\mathcal{D}_{\mathsf{sk}}(a\|b\|c)$: thanks to the private key, the decryption algorithm $\mathcal{D}_{\mathsf{sk}}$ extracts

  $$r = f^{-1}(a), \quad \text{and next} \quad m = b \oplus \mathcal{G}(r).$$

  If $c = \mathcal{H}(m, r)$, the algorithm returns m, otherwise it returns "Reject."

9.3.2 Security Analysis

About this construction, one can prove:

Theorem 9.2 *Let $\mathcal{A}$ be a* CCA-adversary against the semantic security of the above encryption scheme BR. Assume that $\mathcal{A}$ has advantage ε and running time τ and makes q_d, q_g and q_h queries to the decryption oracle, and the hash functions $\mathcal{G}$ and $\mathcal{H}$, respectively. Then

$$\begin{array}{rcl} \mathsf{Succ}_f^{\mathsf{ow}}(\tau') & \geq & \dfrac{\varepsilon}{2} - \dfrac{2q_d}{2^{k_1}}, \\ \text{with} \quad \tau' & \leq & \tau + (q_g + q_h) \cdot T_f, \end{array}$$

where T_f denotes the time complexity for evaluating f.

Proof : In the following we use starred letters ($r^\star$, $a^\star$, $b^\star$, $c^\star$ and $y^\star$) to refer to the challenge ciphertext, whereas unstarred letters (r, a, b, c and y) refer to the ciphertext asked to the decryption oracle.

Game $\mathbf{G}_0$: A pair of keys $(\mathsf{pk}, \mathsf{sk})$ is generated using $\mathcal{K}(1^k)$. Adversary A_1 is fed with pk, the description of f, and outputs a pair of messages (m_0, m_1). Next a challenge ciphertext is produced by flipping a coin b and producing a ciphertext $y^\star = a^\star\|b^\star\|c^\star$ of m_b. This ciphertext comes from a random $r^\star \stackrel{R}{\leftarrow} X$ and $a^\star = f(r^\star)$, $b^\star = m_b \oplus \mathcal{G}(r^\star)$ and $c^\star = \mathcal{H}(m_b, r^\star)$. On input $y^\star$, A_2 outputs bit b'. In both stages, the adversary is given additional access to the decryption oracle $\mathcal{D}_{\mathsf{sk}}$. The only requirement is that the challenge ciphertext $y^\star$ cannot be queried from the decryption oracle.

We denote by S_0 the event $b' = b$ and use a similar notation S_i in any $\mathbf{G}_i$ below. By definition, we have

$$\Pr[\mathsf{S}_0] = \frac{1}{2} + \frac{\varepsilon}{2}. \tag{9.11}$$

Game $\mathbf{G}_1$: In this game, one makes the classical simulation of the random oracles, with random answers for any new query, as shown on Figure 9.2. This game is clearly identical to the previous one.

Game $\mathbf{G}_2$: In this game, one randomly chooses $h^+ \stackrel{R}{\leftarrow} \{0,1\}^{k_1}$, and uses it instead of $\mathcal{H}(m^\star, r^\star)$.

►Rule Chal–Hash$^{(2)}$

> The value $h^+ \stackrel{R}{\leftarrow} \{0,1\}^{k_1}$ has been chosen ahead of time, choose randomly $r^\star$, then set $a^\star = f(r^\star)$, $g^\star = \mathcal{G}(r^\star)$, $b^\star = m^\star \oplus g^\star$, and $c^\star = h^+$.

The two games $\mathbf{G}_2$ and $\mathbf{G}_1$ are perfectly indistinguishable unless $(m^\star, r^\star)$ is asked for $\mathcal{H}$, either by the adversary or the decryption oracle. But the latter case is not possible, otherwise the decryption query would be the challenge ciphertext. More generally, we denote by AskR_2 the event that $r^\star$ has been asked to $\mathcal{G}$ or to $\mathcal{H}$, by the adversary. We have:

$$|\Pr[\mathsf{S}_2] - \Pr[\mathsf{S}_1]| \leq \Pr[\mathsf{AskR}_2]. \tag{9.12}$$

Game $\mathbf{G}_3$: We start modifying the simulation of the decryption oracle, by rejecting any ciphertext $(a\|b\|c)$ for which the corresponding (m, r) has not been queried to $\mathcal{H}$:

$\mathcal{G}$, $\mathcal{H}$ Oracles

Query $\mathcal{G}(r)$: if a record (r, g) appears in G-List, the answer is g. Otherwise the answer g is chosen randomly: $g \in \{0,1\}^n$ and the record (r, g) is added in G-List.

Query $\mathcal{H}(m, r)$: if a record (m, r, h) appears in H-List, the answer is h. Otherwise the answer h is chosen randomly: $h \in \{0,1\}^{k_1}$ and the record (m, r, h) is added in H-List.

$\mathcal{D}$ Oracle

Query $\mathcal{D}_{\mathsf{sk}}(a\|b\|c)$: one applies the following rules:

▶Rule Decrypt−R$^{(1)}$

> Compute $r = f^{-1}(a)$;

Then, compute $m = b \oplus \mathcal{G}(r)$, and finally,

▶Rule Decrypt−H$^{(1)}$

> If $c = \mathcal{H}(m, r)$, one returns m, otherwise one returns "Reject."

Challenger

For two messages (m_0, m_1), flip a coin b and set $m^\star = m_b$.

▶Rule Chal−Hash$^{(1)}$

> Choose randomly $r^\star$, then set
> $a^\star = f(r^\star)$,
> $g^\star = \mathcal{G}(r^\star)$, $\quad b^\star = m^\star \oplus g^\star$,
> $c^\star = \mathcal{H}(m^\star, r^\star)$.

Then, output $y^\star = a^\star\|b^\star\|c^\star$.

Figure 9.2: Formal Simulation of the IND-CCA Game against the BR Construction

▶Rule Decrypt−H$^{(3)}$

> Look up in H-List for (m, r, c). If such a triple does not exist, then output "Reject", otherwise output m.

Such a simulation differs from the previous one if the value c has been correctly guessed, by chance:

$$|\Pr[\mathsf{S}_3] - \Pr[\mathsf{S}_2]| \le \frac{q_d}{2^{k_1}} \qquad |\Pr[\mathsf{AskR}_3] - \Pr[\mathsf{AskR}_2]| \le \frac{q_d}{2^{k_1}}. \tag{9.13}$$

Game $\mathbf{G}_4$: In this game, one randomly chooses $r^+ \stackrel{R}{\leftarrow} X$ and $g^+ \stackrel{R}{\leftarrow} \{0,1\}^n$, and uses r^+ instead of $r^\star$, as well as g^+ instead of $\mathcal{G}(r^\star)$.

▶Rule Chal−Hash$^{(4)}$

> The three values $r^+ \stackrel{R}{\leftarrow} X$, $g^+ \stackrel{R}{\leftarrow} \{0,1\}^n$ and $h^+ \stackrel{R}{\leftarrow} \{0,1\}^{k_1}$ have been chosen ahead of time, then set $a^\star = f(r^+)$, $\quad b^\star = m^\star \oplus g^+$, $\quad c^\star = h^+$.

The two games $\mathbf{G}_4$ and $\mathbf{G}_3$ are perfectly indistinguishable unless $r^\star$ is asked for $\mathcal{G}$, either by the adversary or the decryption oracle. The former case has already been cancelled in the previous game, in AskR_3. The latter case does not make any difference since either $\mathcal{H}(m, r^\star)$ has been queried by the adversary, which falls in AskR_3, or the ciphertext is rejected in both games. We have:

$$\Pr[\mathsf{S}_4] = \Pr[\mathsf{S}_3] \qquad \Pr[\mathsf{AskR}_4] = \Pr[\mathsf{AskR}_3]. \tag{9.14}$$

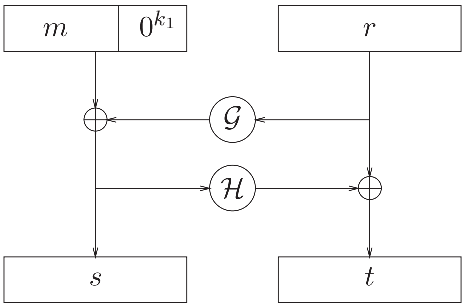


Figure 9.3: Optimal Asymmetric Encryption Padding

In this game, $m^\star$ is masked by g^+, a random value which never appears anywhere else. Thus, the input to $\mathcal{A}_2$ follows a distribution that does not depend on b. Accordingly:

$$\Pr[\mathsf{S}_4] = \frac{1}{2}. \tag{9.15}$$

Game G$_5$: Finally, one randomly chooses $a^+ \overset{R}{\leftarrow} X$, which implicitly defines a random r^+ in X. Actually, a^+ is the given random challenge for which one is looking for the pre-image r^+.

▶**Rule Chal–Hash**$^{(5)}$

> The three values $a^+ \overset{R}{\leftarrow} X$, $g^+ \overset{R}{\leftarrow} \{0,1\}^n$ and $h^+ \overset{R}{\leftarrow} \{0,1\}^{k_1}$ have been chosen/given ahead of time, then set $a^\star = a^+, \quad b^\star = m^\star \oplus g^+, \quad c^\star = h^+$.

The two games $\mathbf{G}_5$ and $\mathbf{G}_4$ are perfectly indistinguishable, thanks to the permutation property of f.

One may now note that the event AskR_5 leads to the pre-image of a^+ by f in the queries asked to $\mathcal{G}$ and $\mathcal{H}$, by the adversary. By checking all of them, one gets it:

$$\Pr[\mathsf{AskR}_5] \leq \mathsf{Succ}_f^{\mathsf{ow}}(\tau + (q_g + q_h)T_f). \tag{9.16}$$

9.4 OAEP: the Optimal Asymmetric Encryption Padding

9.4.1 Description

The problem with the above generic construction is the high over-head. When one encrypts with a trapdoor one-way permutation onto X, one could hope the ciphertext to be an element in X, without anything else. In 1994, Bellare and Rogaway proposed such a more compact generic conversion [?], in the random-oracle model, the "Optimal Asymmetric Encryption Padding" (OAEP, see Figure 9.3), obtained from a trapdoor one-way permutation f onto $\{0,1\}^k$, whose inverse is denoted by f^{-1}. We need two hash functions $\mathcal{G}$ and $\mathcal{H}$:

$$\mathcal{G} : \{0,1\}^{k_0} \longrightarrow \{0,1\}^{k-k_0} \quad \text{and} \quad \mathcal{H} : \{0,1\}^{k-k_0} \longrightarrow \{0,1\}^{k_0},$$

for some k_0. We also need n and k_1 which satisfy $k = n + k_0 + k_1$. Then the encryption scheme $\mathsf{OAEP} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ can be described as follows:

- $\mathcal{K}(1^k)$: specifies an instance of the function f, and of its inverse f^{-1}. The public key pk is therefore f and the private key sk is f^{-1}.

- $\mathcal{E}_{\mathsf{pk}}(m;r)$: given a message $m \in \{0,1\}^n$, and a random value $r \overset{R}{\leftarrow} \{0,1\}^{k_0}$, the encryption algorithm $\mathcal{E}_{\mathsf{pk}}$ computes

  $$s = (m\|0^{k_1}) \oplus \mathcal{G}(r) \quad \text{and} \quad t = r \oplus \mathcal{H}(s),$$

  and outputs the ciphertext $c = f(s,t)$.

- $\mathcal{D}_{\mathsf{sk}}(c)$: thanks to the private key, the decryption algorithm $\mathcal{D}_{\mathsf{sk}}$ extracts

  $$(s,t) = f^{-1}(c), \quad \text{and next} \quad r = t \oplus \mathcal{H}(s) \quad \text{and} \quad M = s \oplus \mathcal{G}(r).$$

  If $[M]_{k_1} = 0^{k_1}$, the algorithm returns $[M]^n$, otherwise it returns "Reject."

In the above description, $[M]_{k_1}$ denotes the k_1 least significant bits of M, while $[M]^n$ denotes the n most significant bits of M.

9.4.2 About the Security

Paper [?] includes a proof that, provided f is a one-way trapdoor permutation, the resulting OAEP encryption scheme is both semantically secure and weakly plaintext-aware. This implies the semantic security against indifferent chosen-ciphertext attacks, also called security against lunchtime attacks (IND-CCA1). Indeed, the *Weak Plaintext-Awareness* means that the adversary cannot produce a new valid ciphertext, until it has seen any valid one, without knowing (awareness) the plaintext. This is more formally defined by the existence of a plaintext-extractor which, on input a ciphertext and the list of the query-answers of the random oracles, outputs the corresponding plaintext. This plaintext-extractor is thus enough for simulating the decryption oracle, but in the first step of the attack only. We briefly comment on the intuition behind (weak) plaintext-awareness. When the plaintext-extractor receives a ciphertext c, then:

- either s has been queried to $\mathcal{H}$ and r has been queried to $\mathcal{G}$, in which case the extractor finds the cleartext by inspecting the two query lists G-List and H-List,
- or else the decryption of (s,t) remains highly random and there is little chance to meet the redundancy 0^{k_1}: the plaintext extractor can safely declare the ciphertext invalid.

The argument collapses when the plaintext-extractor receives additional valid ciphertexts, since this puts additional implicit constraints on $\mathcal{G}$ and $\mathcal{H}$. These constraints cannot be seen by inspecting the query lists. Hence the requirement of a stronger notion of *plaintext-awareness.* In [?], Bellare, Desai, Pointcheval, and Rogaway defined such a stronger notion which extends the previous *awareness* of the plaintext even after having seen valid ciphertexts. But such a plaintext-awareness notion had never been studied for OAEP, while it was still widely admitted.

9.4.2.1 Shoup's Counter-Example

In his papers [?, ?], Shoup showed that it was quite unlikely to extend the results of [?] to obtain adaptive chosen-ciphertext security, under the sole one-wayness of the permutation. His counter-example made use of the ad hoc notion of an *XOR-malleable* trapdoor one-way permutation: for such permutation f_0, one can compute $f_0(x \oplus a)$ from $f_0(x)$ and a, with non-negligible probability.

Let f_0 be such an XOR-malleable permutation. Define f by $f(s\|t) = s\|f_0(t)$. Clearly, f is also a trapdoor one-way permutation. However, it leads to a malleable encryption scheme as we now show. Start with a challenge ciphertext $y = f(s\|t) = s\|u$, where $s\|t$ is the output of the OAEP transformation on the redundant message $m\|0^{k_1}$ and the random string r (see Figure 9.4),

$$s = \mathcal{G}(r) \oplus (m\|0^{k_1}), \qquad t = \mathcal{H}(s) \oplus r \quad \text{and} \quad u = f_0(t).$$

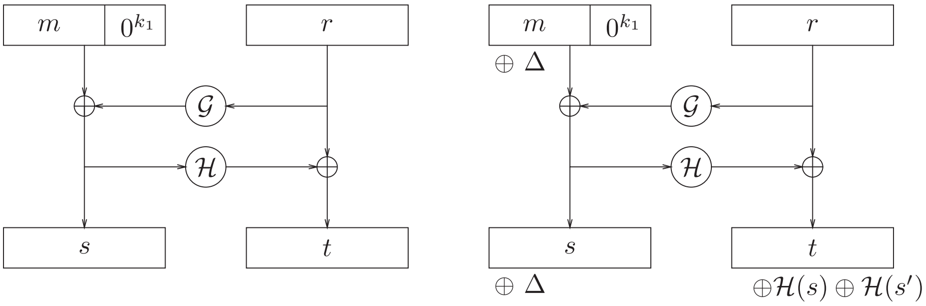


Figure 9.4: Shoup's attack.

Since f is the identity on its leftmost part, we know s, and can define $\Delta = \delta \| 0^{k_1}$, for any random string δ, and $s' = s \oplus \Delta$. We then set $t' = \mathcal{H}(s') \oplus r = t \oplus (\mathcal{H}(s) \oplus \mathcal{H}(s'))$. The XOR-malleability of f_0 allows one to obtain $u' = f_0(t')$ from $u = f_0(t)$ and $\mathcal{H}(s) \oplus \mathcal{H}(s')$, with significant probability. Finally, $y' = s' \| u'$ is a valid ciphertext of $m' = m \oplus \delta$, built from $r' = r$, since:

$$t' = f_0^{-1}(u') = t \oplus (\mathcal{H}(s) \oplus \mathcal{H}(s')) = \mathcal{H}(s') \oplus r, \qquad r' = \mathcal{H}(s') \oplus t' = r$$

and

$$s' \oplus \mathcal{G}(r') = \Delta \oplus s \oplus \mathcal{G}(r) = \Delta \oplus (m \| 0^{k_1}) = (m \oplus \delta) \| 0^{k_1}.$$

Note that the above definitely contradicts adaptive chosen-ciphertext security: asking the decryption of y' after having received the ciphertext y, an adversary obtains m' and easily recovers the actual cleartext m from m' and δ. Also note that Shoup's counter-example exactly stems from where the intuition developed at the end of the previous section failed: a valid ciphertext y' was created without querying the oracle at the corresponding random seed r', using in place the implicit constraint on $\mathcal{G}$ coming from the received valid ciphertext y.

Using methods from relativized complexity theory, Shoup [?, ?] built a non-standard model of computation, where there exists an XOR-malleable trapdoor one-way permutation. As a consequence, it is very unlikely that one can prove the IND-CCA security of the OAEP construction, under the sole one-wayness of the underlying permutation. Indeed, all methods of proof currently known still apply in relativized models of computation.

9.4.3 The Actual Security of OAEP

Shoup [?, ?] furthermore provided a specific proof for RSA with public exponent 3. However, there is little hope of extending this proof for higher exponents. Hopefully, Fujisaki, Okamoto, Pointcheval, and Stern provided a general security analysis, but under a stronger assumption about the underlying permutation [?, ?]. Indeed, they prove that the scheme is IND-CCA in the random-oracle model [?], relative to the *partial-domain* one-wayness of permutation f.

9.4.3.1 Partial-Domain One-Wayness

Let us first introduce this new computational assumption. Let f be a permutation $f : \{0,1\}^k \longrightarrow \{0,1\}^k$, which can also be written as

$$f : \{0,1\}^{n+k_1} \times \{0,1\}^{k_0} \longrightarrow \{0,1\}^{n+k_1} \times \{0,1\}^{k_0},$$

with $k = n + k_0 + k_1$. In the original description of OAEP from [?], it is only required that f is a trapdoor one-way permutation. However, in the following, we consider two additional related problems, namely partial-domain one-wayness and set partial-domain one-wayness:

- Permutation f is (τ, ε)-one-way if any adversary $\mathcal{A}$ whose running time is bounded by τ has success probability $\mathsf{Succ}_f^{\mathsf{ow}}(\mathcal{A})$ upper-bounded by ε, where

$$\mathsf{Succ}_f^{\mathsf{ow}}(\mathcal{A}) = \Pr_{s,t}[\mathcal{A}(f(s,t)) = (s,t)].$$

- Permutation f is (τ, ε)-partial-domain one-way if any adversary $\mathcal{A}$ whose running time is bounded by τ has success probability $\mathsf{Succ}_f^{\mathsf{pd\text{-}ow}}(\mathcal{A})$ upper-bounded by ε, where

$$\mathsf{Succ}_f^{\mathsf{pd\text{-}ow}}(\mathcal{A}) = \Pr_{s,t}[\mathcal{A}(f(s,t)) = s].$$

- Permutation f is $(\ell, \tau, \varepsilon)$-set partial-domain one-way if any adversary $\mathcal{A}$, outputting a set of ℓ elements within time bound τ, has success probability $\mathsf{Succ}_f^{\mathsf{s\text{-}pd\text{-}ow}}(\mathcal{A})$ upper-bounded by ε, where

$$\mathsf{Succ}_f^{\mathsf{s\text{-}pd\text{-}ow}}(\mathcal{A}) = \Pr_{s,t}[s \in \mathcal{A}(f(s,t))].$$

We denote by $\mathsf{Succ}_f^{\mathsf{ow}}(\tau)$ (resp. $\mathsf{Succ}_f^{\mathsf{pd\text{-}ow}}(\tau)$ and $\mathsf{Succ}_f^{\mathsf{s\text{-}pd\text{-}ow}}(\ell, \tau)$) the maximal success probability $\mathsf{Succ}_f^{\mathsf{ow}}(\mathcal{A})$ (resp. $\mathsf{Succ}_f^{\mathsf{pd\text{-}ow}}(\mathcal{A})$ and $\mathsf{Succ}_f^{\mathsf{s\text{-}pd\text{-}ow}}(\mathcal{A})$). The maximum ranges over all adversaries whose running time is bounded by τ. In the third case, there is an obvious additional restriction on this range from the fact that $\mathcal{A}$ outputs sets with ℓ elements. It is clear that for any τ and $\ell \geq 1$,

$$\mathsf{Succ}_f^{\mathsf{s\text{-}pd\text{-}ow}}(\ell, \tau) \geq \mathsf{Succ}_f^{\mathsf{pd\text{-}ow}}(\tau) \geq \mathsf{Succ}_f^{\mathsf{ow}}(\tau).$$

Note that, by randomly selecting an element in the set returned by an adversary to the set partial-domain one-wayness, one breaks partial-domain one-wayness with probability $\mathsf{Succ}_f^{\mathsf{s\text{-}pd\text{-}ow}}(\mathcal{A})/\ell$. This provides the following inequality

$$\mathsf{Succ}_f^{\mathsf{pd\text{-}ow}}(\tau) \geq \mathsf{Succ}_f^{\mathsf{s\text{-}pd\text{-}ow}}(\ell, \tau)/\ell.$$

However, for specific choices of f, more efficient reductions may exist. Also, in some cases, all three problems are polynomially equivalent. This is the case for the RSA permutation [**?**], hence the global security result for RSA-OAEP.

9.4.4 Intuition behind the Proof of Security

In the following we use starred letters ($r^\star$, $s^\star$, $t^\star$ and $y^\star$) to refer to the challenge ciphertext, whereas unstarred letters (r, s, t and y) refer to the ciphertext asked to the decryption oracle.

Referring to our description of the intuition behind the original OAEP proof of security, given above, we can carry a more subtle analysis by distinguishing the case where s has not been queried from oracle $\mathcal{H}$ from the case where r has not been queried from $\mathcal{G}$. If s is not queried, then $\mathcal{H}(s)$ is random and uniformly distributed and r is necessarily defined as $t \oplus \mathcal{H}(s)$. This holds even if s matches with the string $s^\star$ coming from the valid ciphertext $y^\star$. There is a minute probability that $t \oplus \mathcal{H}(s)$ is queried from $\mathcal{G}$ or equals $r^\star$. Thus, $\mathcal{G}(r)$ is random: there is little chance that the redundancy 0^{k_1} is met and the extractor can safely reject.

We claim that r cannot match with $r^\star$, unless $s^\star$ is queried from $\mathcal{H}$. This is because $r^\star = t^\star \oplus \mathcal{H}(s^\star)$ equals $r = t \oplus \mathcal{H}(s)$ with minute probability. Thus, if r is not queried, then $\mathcal{G}(r)$ is random and we similarly infer that the extractor can safely reject. The argument fails only if $s^\star$ is queried.

Thus rejecting when it cannot combine elements of the lists $\mathsf{G\text{-}List}$ and $\mathsf{H\text{-}List}$ so as to build a pre-image of y, the plaintext-extractor is only wrong with minute probability, unless $s^\star$ has been queried by the adversary. This seems to show that OAEP leads to an $\mathsf{IND\text{-}CCA}$ encryption scheme if it is difficult to invert f "partially", which means: given $y^\star = f(s^\star \| t^\star)$, find $s^\star$.

Chosen-ciphertext security is actually addressed, by turning the intuition explained above into a formal argument, involving a restricted variant of plaintext-awareness (where the list C of ciphertexts is limited to only one ciphertext, the challenge ciphertext $y^\star$):

Theorem 9.3 *Let $\mathcal{A}$ be a* CCA-adversary *against the semantic security of the encryption scheme* OAEP. *Assume that $\mathcal{A}$ has advantage ε and running time τ and makes q_d, q_g and q_h queries to the decryption oracle, and the hash functions $\mathcal{G}$ and $\mathcal{H}$, respectively. Then*

$$\begin{aligned} \mathsf{Succ}_f^{\mathsf{s\text{-}pd\text{-}ow}}(q_h, \tau') &\geq \frac{\varepsilon}{2} - \left(\frac{2(q_d+2)(q_d+2q_g)}{2^{k_0}} + \frac{3q_d}{2^{k_1}} \right), \\ \text{with} \quad \tau' &\leq \tau + q_g \cdot q_h \cdot (T_f + \mathcal{O}(1)), \end{aligned}$$

where T_f denotes the time complexity for evaluating f.

Unfortunately, because of the additional reduction of the basic RSA to the partial-domain RSA problem, the global reduction is very expensive, and is thus meaningful for huge moduli only, more than 4096-bit long. Indeed, the RSA inverter we can build, thanks to the full reduction, has a complexity at least greater than $q_h \cdot (q_h + 2q_g) \times \mathcal{O}(k^3)$. As already remarked, the adversary can ask up to 2^{60} queries to the hash functions, and thus this overhead in the inversion is at least 2^{151}. However, current factoring algorithms can factor up to 4096 bit-long integers within this number of basic operations (see [**?**] for complexity estimates of the most efficient factoring algorithms).

Anyway, the formal proof shows that the global design of OAEP is sound, and that it is still probably safe to use it in practice (*e.g.* in PKCS #1 v2.0, while being very careful during the implementation [**?**]).

Chapter 10

Digital Signature Schemes

10.1 Introduction

Until 1996, no practical **DL**-based cryptographic scheme has ever been formally studied, but heuristically only. And surprisingly, at the Eurocrypt '96 conference, two opposite studies were conducted on the El Gamal signature scheme [**?**], the first **DL**-based signature scheme designed in 1985 and depicted on Figure 10.1.

Initialization → (p, g)
g a generator of $\mathbb{Z}_p^\star$,
where p is a large prime
→ (p, g)

$\mathcal{K}$: **Key Generation** → (y, x)
private key $x \in \mathbb{Z}_{p-1}^\star$
public key $y = g^x \bmod p$
→ (y, x)

$\mathcal{S}$: **Signature of** m → (r, s)
K is randomly chosen in $\mathbb{Z}_{p-1}^\star$
$r = g^K \bmod p$ $s = (m - xr)/K \bmod p - 1$
→ (r, s) is a signature of m

$\mathcal{V}$: **Verification of** (m, r, s)
check whether $g^m \overset{?}{=} y^r r^s \bmod p$
→ Yes/No

Figure 10.1: The El Gamal Signature Scheme.

Whereas existential forgeries were known for that scheme, it was believed to prevent universal forgeries. The first analysis, from Daniel Bleichenbacher [**?**], showed such a universal forgery when the generator g is not properly chosen. The second one, from David Pointcheval and Jacques Stern [**?**], proved the security against existential forgeries under adaptive chosen-message attacks of a slight variant with a randomly chosen generator g. The latter variant simply replaces the message m by $\mathcal{H}(m, r)$ in the computation, while one uses a hash function $\mathcal{H}$ that is assumed to behave like a random oracle. It is amazing to remark that the Bleichenbacher's attack also applies on Pointcheval-Stern's variant. Therefore, depending on the initialization, the variant could be a very strong signature scheme or become a very weak one!

As a consequence, a proof has to be performed in details, with precise assumptions and achievements. Furthermore, the conclusions have to be strictly followed by developers, otherwise the concrete implementation of a secure scheme can be very weak.

10.2 Some Schemes

The first *secure* signature scheme was proposed by Goldwasser *et al.* [?] in 1984. It used the notion of claw-free permutations. A pair of permutations (f, g) is said *claw-free* if it is computationally impossible to find a *claw* (x, y), which satisfies $f(x) = g(y)$. Their proposal provided polynomial algorithms with a polynomial reduction between the research of a claw and an existential forgery under an adaptive chosen-message attack. However, the scheme was totally unpractical. What about practical schemes?

10.2.1 The RSA Signature Scheme

Two years after the Diffie-Hellman paper [?], Rivest, Shamir and Adleman [?] proposed the first signature scheme based on the "trapdoor one-way permutation paradigm", using the RSA function: the generation algorithm produces a large composite number $N = pq$, a public key e, and a private key d such that $e \cdot d = 1 \bmod \varphi(N)$. The signature of a message m, encoded as an element in $\mathbb{Z}_N^\star$, is its e-th root, $\sigma = m^{1/e} = m^d \bmod N$. The verification algorithm simply checks whether $m = \sigma^e \bmod N$.

However, the RSA scheme is not secure by itself since it is subject to existential forgery: it is easy to create a valid message-signature pair, without any help of the signer, first randomly choosing a certificate σ and getting the signed message m from the public verification relation, $m = \sigma^e \bmod N$.

10.2.2 The Schnorr Signature Scheme

In 1986 a new paradigm for signature schemes was introduced. It is derived from fair zero-knowledge identification protocols involving a prover and a verifier [?], and uses hash functions in order to create a kind of virtual verifier. The first application was derived from the Fiat–Shamir [?] zero-knowledge identification protocol, based on the hardness of extracting square roots, with a brief outline of its security. Another famous identification scheme [?], together with the signature scheme [?], has been proposed later by Schnorr, based on that paradigm: the generation algorithm produces two large primes p and q, such that $q \geq 2^k$, where k is the security parameter, and $q \,|\, p - 1$, as well as an element g in $\mathbb{Z}_p^\star$ of order q. It also creates a pair of keys, the private key $x \in \mathbb{Z}_q^\star$ and the public key $y = g^{-x} \bmod p$ The signature of a message m is a triple (r, e, s), where $r = g^K \bmod p$, with a random $K \in \mathbb{Z}_q$, the "challenge" $e = \mathcal{H}(m, r)$ and $s = K + ex \bmod q$. The latter satisfies $r = g^s y^e \bmod p$ with $e = \mathcal{H}(m, r)$, which is checked by the verification algorithm.

The security results for that paradigm have been considered as folklore for a long time but without any formal validation.

10.3 DL-Based Signatures

In [?, ?], David Pointcheval and Jacques Stern formally proved the above paradigm when $\mathcal{H}$ is assumed to behave like a random oracle. The proof is based on the by now classical *oracle replay technique*: by a polynomial replay of the attack with different random oracles (the $\mathcal{Q}_i$'s are the queries and the ρ_i's are the answers), one make the attacker forge signatures that are suitably related. This generic technique is depicted on Figure 10.2, where the signature of a message m is a triple (σ_1, h, σ_2), with $h = \mathcal{H}(m, \sigma_1)$ which depends on the message and the first part of the signature, both bound not to change for the computation of σ_2, which really relies on the knowledge of the private key. If the probability of fraud is high enough, then with good probability, the adversary is able to answer to many distinct outputs from the $\mathcal{H}$ function, on the input (m, σ_1).

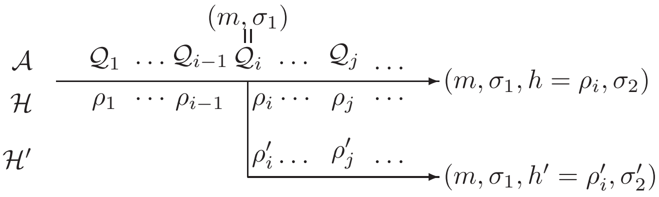

(m, σ_1)
$\mathcal{A}$
$\mathcal{Q}_1 \; \ldots \mathcal{Q}_{i-1} \, \mathcal{Q}_i \; \ldots \; \mathcal{Q}_j \; \ldots$
$(m, \sigma_1, h = \rho_i, \sigma_2)$
$\mathcal{H}$
$\rho_1 \; \cdots \rho_{i-1}$
$\rho_i \cdots \; \rho_j \; \cdots$
$\mathcal{H}'$
$\rho'_i \ldots \; \rho'_j \; \ldots$
$(m, \sigma_1, h' = \rho'_i, \sigma'_2)$

Figure 10.2: The Oracle Replay Technique

Initialization (security parameter k) $\rightarrow (\mathcal{G}, g, \mathcal{H})$

$\mathbf{g}$ a generator of any cyclic group $(\mathcal{G}, +)$
of order q, with $2^{k-1} \leq q < 2^k$
$\mathcal{H}$ a hash function: $\{0,1\}^\star \rightarrow \mathbb{Z}_q$
$\rightarrow (\mathcal{G}, g, \mathcal{H})$

$\mathcal{K}$**: Key Generation** $\rightarrow (\mathbf{y}, x)$

private key $x \in \mathbb{Z}_q^\star$
public key $\mathbf{y} = -x \cdot \mathbf{g}$
$\rightarrow (\mathbf{y}, x)$

$\mathcal{S}$**: Signature of** $m \rightarrow (\mathbf{r}, h, s)$

K is randomly chosen in $\mathbb{Z}_q^\star$
$\mathbf{r} = K \cdot \mathbf{g} \qquad h = \mathcal{H}(m, r) \qquad s = K + xh \bmod q$
$\rightarrow (\mathbf{r}, h, s)$ is a signature of m

$\mathcal{V}$**: Verification of** (m, r, s)

check whether $h \overset{?}{=} \mathcal{H}(m, \mathbf{r})$
and $\mathbf{r} \overset{?}{=} s \cdot \mathbf{g} + h \cdot \mathbf{y}$
$\rightarrow$ Yes/No

Figure 10.3: The Schnorr Signature Scheme.

To be more concrete, let us consider the Schnorr signature scheme, which is presented on Figure 10.3, in any "suitable" cyclic group $\mathcal{G}$ of prime order q, where at least the Discrete Logarithm problem is hard. We expect to obtain two signatures $(\mathbf{r} = \sigma_1, h, s = \sigma_2)$ and $(\mathbf{r}' = \sigma'_1, h', s' = \sigma'_2)$ of an identical message m such that $\sigma_1 = \sigma'_1$, but $h \neq h'$. Thereafter, we can easily extract the discrete logarithm of the public key:

$$\left.\begin{array}{rclcl} \mathbf{r} & = & s \cdot \mathbf{g} & + & h \cdot \mathbf{y} \\ \mathbf{r} & = & s' \cdot \mathbf{g} & + & h' \cdot \mathbf{y} \end{array}\right\} \Rightarrow (s - s') \cdot \mathbf{g} = (h' - h) \cdot \mathbf{y},$$

which leads to $\log_{\mathbf{g}} \mathbf{y} = (s - s') \cdot (h' - h)^{-1} \bmod q$.

10.3.1 General Tools

First, let us recall the "Splitting Lemma" which will be the main probabilistic tool for the "Forking Lemma". It translates the fact that when a subset A is "large" in a product space $X \times Y$, it has many "large" sections.

Lemma 10.1 (The Splitting Lemma) *Let $A \subset X \times Y$ such that $\Pr[(x, y) \in A] \geq \varepsilon$. For any $\alpha < \varepsilon$, define*

$$B = \left\{ (x, y) \in X \times Y \mid \Pr_{y' \in Y}[(x, y') \in A] \geq \varepsilon - \alpha \right\},$$

then the following statements hold:

(i) $\Pr[B] \geq \alpha$

(ii) $\forall (x,y) \in B, \Pr_{y' \in Y}[(x,y') \in A] \geq \varepsilon - \alpha$.

(iii) $\Pr[B \,|\, A] \geq \alpha/\varepsilon$.

Proof : In order to prove statement (i), we argue by contradiction, using the notation $\bar{B}$ for the complement of B in $X \times Y$. Assume that $\Pr[B] < \alpha$. Then

$$\varepsilon \;\leq\; \Pr[B] \cdot \Pr[A \,|\, B] + \Pr[\bar{B}] \cdot \Pr[A \,|\, \bar{B}] \;<\; \alpha \cdot 1 + 1 \cdot (\varepsilon - \alpha) \;=\; \varepsilon.$$

This implies a contradiction, hence the result.

Statement (ii) is a straightforward consequence of the definition.

We finally turn to the last assertion, using Bayes' law:

$$\begin{aligned} \Pr[B \,|\, A] &= 1 - \Pr[\bar{B} \,|\, A] \\ &= 1 - \Pr[A \,|\, \bar{B}] \cdot \Pr[\bar{B}] / \Pr[A] \geq 1 - (\varepsilon - \alpha)/\varepsilon = \alpha/\varepsilon. \end{aligned}$$

10.3.2 No-Message Attacks

The following *Forking Lemma* just states that the above oracle replay technique will often success with any good adversary.

Theorem 10.1 (The Forking Lemma) *Let $(\mathcal{K}, \mathcal{S}, \mathcal{V})$ be a digital signature scheme with security parameter k, with a signature as above, of the form $(m, \sigma_1, h, \sigma_2)$, where $h = \mathcal{H}(m, \sigma_1)$ and σ_2 depends on σ_1 and h only. Let $\mathcal{A}$ be a probabilistic polynomial time Turing machine whose input only consists of public data and which can ask q_h queries to the random oracle, with $q_h > 0$. We assume that, within the time bound T, $\mathcal{A}$ produces, with probability $\varepsilon \geq 7q_h/2^k$, a valid signature $(m, \sigma_1, h, \sigma_2)$. Then, within time $T' \leq 16 q_h T/\varepsilon$, and with probability $\varepsilon' \geq 1/9$, a replay of this machine outputs two valid signatures $(m, \sigma_1, h, \sigma_2)$ and $(m, \sigma_1, h', \sigma_2')$ such that $h \neq h'$.*

Proof : We are given an adversary $\mathcal{A}$, which is a probabilistic polynomial time Turing machine with random tape ω. During the attack, this machine asks a polynomial number of questions to the random oracle $\mathcal{H}$. We may assume that these questions are distinct: for instance, $\mathcal{A}$ can store questions and answers in a table. Let $\mathcal{Q}_1, \ldots, \mathcal{Q}_{q_h}$ be the q_h distinct questions and let $\rho = (\rho_1, \ldots, \rho_{q_h})$ be the list of the q_h answers of $\mathcal{H}$. It is clear that a random choice of $\mathcal{H}$ exactly corresponds to a random choice of ρ. Then, for a random choice of $(\omega, \mathcal{H})$, with probability ε, $\mathcal{A}$ outputs a valid signature $(m, \sigma_1, h, \sigma_2)$. Since $\mathcal{H}$ is a random oracle, it is easy to see that the probability for h to be equal to $\mathcal{H}(m, \sigma_1)$ is less than $1/2^k$, unless it has been asked during the attack. So, it is likely that the question (m, σ_1) is actually asked during a successful attack. Accordingly, we define $\mathit{Ind}_{\mathcal{H}}(\omega)$ to be the index of this question: $(m, \sigma_1) = \mathcal{Q}_{\mathit{Ind}_{\mathcal{H}}(\omega)}$ (we let $\mathit{Ind}_{\mathcal{H}}(\omega) = \infty$ if the question is never asked). We then define the sets

$$\begin{aligned} \mathbf{S} &= \left\{(\omega, \mathcal{H}) \,|\, \mathcal{A}^{\mathcal{H}}(\omega) \text{ succeeds } \& \ \mathit{Ind}_{\mathcal{H}}(\omega) \neq \infty\right\}, \\ \text{and } \mathbf{S}_i &= \left\{(\omega, \mathcal{H}) \,|\, \mathcal{A}^{\mathcal{H}}(\omega) \text{ succeeds } \& \ \mathit{Ind}_{\mathcal{H}}(\omega) = i\right\} \quad \text{ for } \ i \in \{1, \ldots, q_h\}. \end{aligned}$$

We thus call $\mathbf{S}$ the set of the successful pairs $(\omega, \mathcal{H})$.

One should note that the set $\{\mathbf{S}_i \,|\, i \in \{1, \ldots, q_h\}\}$ is a partition of $\mathbf{S}$. With those definitions, we find a lower bound for the probability of success, $\nu = \Pr[\mathbf{S}] \geq \varepsilon - 1/2^k$. Since we did the assumption that $\varepsilon \geq 7q_h/2^k \geq 7/2^k$, then $\nu \geq 6\varepsilon/7$. Let I be the set consisting of the most likely indices i,

$$I = \{i \,|\, \Pr[\mathbf{S}_i \,|\, \mathbf{S}] \geq 1/2q_h\}.$$

The following lemma claims that, in case of success, the index lies in I with probability at least 1/2.

Lemma 10.2

$$\Pr[Ind_{\mathcal{H}}(\omega) \in I \,|\, \mathbf{S}] \geq \frac{1}{2}.$$

Proof : By definition of the sets $\mathbf{S}_i$, $\Pr[Ind_{\mathcal{H}}(\omega) \in I \,|\, \mathbf{S}] = \sum_{i \in I} \Pr[\mathbf{S}_i \,|\, \mathbf{S}]$. This probability is equal to $1 - \sum_{i \notin I} \Pr[\mathbf{S}_i \,|\, \mathbf{S}]$. Since the complement of I contains fewer than q_h elements, this probability is at least $1 - q_h \times 1/2q_h \geq 1/2$.

We now run the attacker $2/\varepsilon$ times with random ω and random $\mathcal{H}$. Since $\nu = \Pr[\mathbf{S}] \geq 6\varepsilon/7$, with probability greater than $1 - (1 - 6\varepsilon/7)^{2/\varepsilon}$, we get at least one pair $(\omega, \mathcal{H})$ in $\mathbf{S}$. It is easily seen that this probability is lower bounded by $1 - e^{-12/7} \geq 4/5$.

We now apply the Splitting-lemma (Lemma 10.1, with $\varepsilon = \nu/2q_h$ and $\alpha = \varepsilon/2$) for each integer $i \in I$: we denote by $\mathcal{H}_{|i}$ the restriction of $\mathcal{H}$ to queries of index strictly less than i. Since $\Pr[\mathbf{S}_i] \geq \nu/2q_h$, there exists a subset Ω_i of executions such that,

$$\begin{aligned} \text{for any } (\omega, \mathcal{H}) \in \Omega_i, \Pr_{\mathcal{H}'}[(\omega, \mathcal{H}') \in \mathbf{S}_i \,|\, \mathcal{H}'_{|i} = \mathcal{H}_{|i}] &\geq \frac{\nu}{4q_h} \\ \Pr[\Omega_i \,|\, \mathbf{S}_i] &\geq \frac{1}{2}. \end{aligned}$$

Since all the subsets $\mathbf{S}_i$ are disjoint,

$$\begin{aligned} &\Pr_{\omega,\mathcal{H}}[(\exists i \in I)\ (\omega, \mathcal{H}) \in \Omega_i \cap \mathbf{S}_i \,|\, \mathbf{S}] \\ &\quad = \Pr\left[\bigcup_{i \in I} (\Omega_i \cap \mathbf{S}_i) \,|\, \mathbf{S}\right] = \sum_{i \in I} \Pr[\Omega_i \cap \mathbf{S}_i \,|\, \mathbf{S}] \\ &\quad = \sum_{i \in I} \Pr[\Omega_i \,|\, \mathbf{S}_i] \cdot \Pr[\mathbf{S}_i \,|\, \mathbf{S}] \geq \left(\sum_{i \in I} \Pr[\mathbf{S}_i \,|\, \mathbf{S}]\right)/2 \geq \frac{1}{4}. \end{aligned}$$

We let β denote the index $Ind_{\mathcal{H}}(\omega)$ corresponding to the successful pair. With probability at least $1/4$, $\beta \in I$ and $(\omega, \mathcal{H}) \in \mathbf{S}_\beta \cap \Omega_\beta$. Consequently, with probability greater than $4/5 \times 1/5 = 1/5$, the $2/\varepsilon$ attacks have provided a successful pair $(\omega, \mathcal{H})$, with $\beta = Ind_{\mathcal{H}}(\omega) \in I$ and $(\omega, \mathcal{H}) \in \mathbf{S}_\beta$. Furthermore, if we replay the attack, with fixed ω but randomly chosen oracle $\mathcal{H}'$ such that $\mathcal{H}'_{|\beta} = \mathcal{H}_{|\beta}$, we know that $\Pr_{\mathcal{H}'}[(\omega, \mathcal{H}') \in \mathbf{S}_\beta \,|\, \mathcal{H}'_{|\beta} = \mathcal{H}_{|\beta}] \geq \nu/4q_h$. Then

$$\begin{aligned} &\Pr_{\mathcal{H}'}[(\omega, \mathcal{H}') \in \mathbf{S}_\beta \text{ and } \rho_\beta \neq \rho'_\beta \,|\, \mathcal{H}'_{|\beta} = \mathcal{H}_{|\beta}] \\ &\qquad \geq \Pr_{\mathcal{H}'}[(\omega, \mathcal{H}') \in \mathbf{S}_\beta \,|\, \mathcal{H}'_{|\beta} = \mathcal{H}_{|\beta}] - \Pr_{\mathcal{H}'}[\rho'_\beta = \rho_\beta] \geq \nu/4q_h - 1/2^k, \end{aligned}$$

where $\rho_\beta = \mathcal{H}(\mathcal{Q}_\beta)$ and $\rho'_\beta = \mathcal{H}'(\mathcal{Q}_\beta)$. Using again the assumption that $\varepsilon \geq 7q_h/2^k$, the above probability is lower-bounded by $\varepsilon/14q_h$. We thus replay the attack $14q_h/\varepsilon$ times with a new random oracle $\mathcal{H}'$ such that $\mathcal{H}'_{|\beta} = \mathcal{H}_{|\beta}$, and get another success with probability greater than

$$1 - (1 - \varepsilon/14q_h)^{14q_h/\varepsilon} \geq 1 - e^{-1} \geq 3/5.$$

Finally, after less than $2/\varepsilon + 14q_h/\varepsilon$ repetitions of the attack, with probability greater than $1/5 \times 3/5 \geq 1/9$, we have obtained two signatures $(m, \sigma_1, h, \sigma_2)$ and $(m', \sigma'_1, h', \sigma'_2)$, both valid w.r.t. their specific random oracle $\mathcal{H}$ or $\mathcal{H}'$, and with the particular relations

$$\mathcal{Q}_\beta = (m, \sigma_1) = (m', \sigma'_1) \text{ and } h = \mathcal{H}(\mathcal{Q}_\beta) \neq \mathcal{H}'(\mathcal{Q}_\beta) = h'.$$

One may have noticed that the mechanics of our reduction depend on some parameters related to the attacker $\mathcal{A}$, namely, its probability of success ε and the number q_h of queries to the random oracle. This induces a lack of uniformity. A uniform version, in expected polynomial time is also possible.

Theorem 10.2 (The Forking Lemma – The Uniform Case) *Let $(\mathcal{K}, \mathcal{S}, \mathcal{V})$ be a digital signature scheme with security parameter k, with a signature as above, of the form $(m, \sigma_1, h, \sigma_2)$, where $h = \mathcal{H}(m, \sigma_1)$ and σ_2 depends on σ_1 and h only. Let $\mathcal{A}$ be a probabilistic polynomial time Turing machine whose input only consists of public data and which can ask q_h queries to the random oracle, with $q_h > 0$. We assume that, within the time bound T, $\mathcal{A}$ produces, with probability $\varepsilon \geq 7q_h/2^k$, a valid signature $(m, \sigma_1, h, \sigma_2)$. Then there is another machine which has control over $\mathcal{A}$ and produces two valid signatures $(m, \sigma_1, h, \sigma_2)$ and $(m, \sigma_1, h', \sigma_2')$ such that $h \neq h'$, in expected time $T' \leq 84480 T q_h/\varepsilon$.*

Proof : Now, we try to design a machine $\mathbf{M}$ which succeeds in expected polynomial time:

1. $\mathbf{M}$ initializes $j = 0$;
2. $\mathbf{M}$ runs $\mathcal{A}$ until it outputs a successful pair $(\omega, \mathcal{H}) \in \mathbf{S}$ and denotes by N_j the number of calls to $\mathcal{A}$ to obtain this success, and by β the index $Ind_{\mathcal{H}}(\omega)$;
3. $\mathbf{M}$ replays, at most $140 N_j \alpha^j$ times, $\mathcal{A}$ with fixed ω and random $\mathcal{H}'$ such that $\mathcal{H}'_{|\beta} = \mathcal{H}_{|\beta}$, where $\alpha = 8/7$;
4. $\mathbf{M}$ increments j and returns to **2**, until it gets a successful forking.

For any execution of $\mathbf{M}$, we denote by J the last value of j and by N the total number of calls to $\mathcal{A}$. We want to compute the expectation of N. Since $\nu = \Pr[\mathbf{S}]$, and $N_j \geq 1$, then $\Pr[N_j \geq 1/5\nu] \geq 3/4$. We define $\ell = \lceil \log_\alpha q_h \rceil$, so that, $140 N_j \alpha^j \geq 28 q_h/\varepsilon$ for any $j \geq \ell$, whenever $N_j \geq 1/5\nu$. Therefore, for any $j \geq \ell$, when we have a first success in $\mathbf{S}$, with probability greater than $1/4$, the index $\beta = Ind_{\mathcal{H}}(\omega)$ is in the set I and $(\omega, \mathcal{H}) \in \mathbf{S}_\beta \cap \Omega_\beta$. Furthermore, with probability greater than $3/4$, $N_j \geq 1/5\nu$. Therefore, with the same conditions as before, that is $\varepsilon \geq 7q_h/2^k$, the probability of getting a successful fork after at most $28 q_h/\varepsilon$ iterations at step **3** is greater than $6/7$.
For any $t \geq \ell$, the probability for J to be greater or equal to t is less than $(1 - 1/4 \times 3/4 \times 6/7)^{t-\ell}$, which is less than $\gamma^{t-\ell}$, with $\gamma = 6/7$. Furthermore,

$$E[N \mid J = t] \leq \sum_{j=0}^{j=t} \left(E[N_j] + 140 E[N_j] \alpha^j \right) \leq \frac{141}{\nu} \times \sum_{j=0}^{j=t} \alpha^j \leq \frac{141}{\nu} \times \frac{\alpha^{t+1}}{\alpha - 1}.$$

So, the expectation of N is $E[N] = \sum_t E[N \mid J = t] \cdot \Pr[J = t]$ and then it can be shown to be less than $84480 q_h/\varepsilon$. Hence the theorem.

10.3.3 Chosen-Message Attacks

However, this just covers the no-message attacks, without any oracle access. Since we can simulate any zero-knowledge protocol, even without having to restart the simulation because of the honest verifier (*i.e.* the challenge is randomly chosen by the random oracle $\mathcal{H}$) one can easily simulate the signer without the private key:

- one first chooses random $h, s \in \mathbb{Z}_q$;
- one computes $\mathbf{r} = s \cdot \mathbf{g} + h \cdot \mathbf{y}$ and defines $\mathcal{H}(m, \mathbf{r})$ to be equal to h, which is a uniformly distributed value;
- one can output $(\mathbf{r}, h, s)$ as a valid signature of the message m.

This furthermore simulates the oracle $\mathcal{H}$, by defining $\mathcal{H}(m, \mathbf{r})$ to be equal to h. This simulation is almost perfect since $\mathcal{H}$ is supposed to output a random value to any new query, and h is indeed a random value. Nevertheless, if the query $\mathcal{H}(m, \mathbf{r})$ has already been asked, $\mathcal{H}(m, \mathbf{r})$ is already defined, and thus the definition $\mathcal{H}(m, \mathbf{r}) \leftarrow h$ is impossible. But such a situation is very rare, which allows us to claim the following result, which stands for the Schnorr signature scheme but also for any signature derived from a three-round honest verifier zero-knowledge interactive proof of knowledge:

Theorem 10.3 *Let $\mathcal{A}$ be a probabilistic polynomial time Turing machine whose input only consists of public data. We denote respectively by q_h and q_s the number of queries that $\mathcal{A}$ can ask to the random oracle and the number of queries that $\mathcal{A}$ can ask to the signer. Assume that, within a time bound T, $\mathcal{A}$ produces, with probability $\varepsilon \geq 10(q_s+1)(q_s+q_h)/2^k$, a valid signature $(m, \sigma_1, h, \sigma_2)$. If the triples (σ_1, h, σ_2) can be simulated without knowing the secret key, with an indistinguishable distribution probability, then, a replay of the attacker $\mathcal{A}$, where interactions with the signer are simulated, outputs two valid signatures $(m, \sigma_1, h, \sigma_2)$ and $(m, \sigma_1, h', \sigma_2')$ such that $h \neq h'$, within time $T' \leq 23 q_h T/\varepsilon$ and with probability $\varepsilon' \geq 1/9$.*

A uniform version of this lemma can also be found in [?]. From a more practical point of view, these results state that if an adversary manages to perform an existential forgery under an adaptive chosen-message attack within an expected time T, after q_h queries to the random oracle and q_s queries to the signing oracle, then the discrete logarithm problem can be solved within an expected time less than $C q_h T$, for some constant C. This result has thereafter been extended to the transformation of any identification scheme secure against passive adversaries into a signature scheme [?].

Brickell, Pointcheval, Vaudenay, and Yung also extended the *forking lemma* technique [?, ?] to many variants of El Gamal [?] and DSA [?], such as the Korean Standard KCDSA [?]. However, the original El Gamal and DSA schemes were not covered by this study, and are certainly not provably secure, even if no attack has ever been found against DSA.

10.4 RSA-Based Signatures

Unfortunately, with the above signatures based on the discrete logarithm, as any construction using the Fiat-Shamir paradigm, we do not really achieve our goal, because the reduction is costly, since q_h can be huge, as much as 2^{60} in practice. This security proof is meaningful for very large groups only.

In 1996, Bellare and Rogaway [?] proposed other candidates, based on the RSA assumption. The first scheme is the by-now classical hash-and-decrypt paradigm (*a.k.a.* the Full-Domain Hash paradigm): as for the basic RSA signature, the generation algorithm produces a large composite number $N = pq$, a public key e, and a private key d such that $e \cdot d = 1 \bmod \varphi(N)$. In order to sign a message m, one first hashes it using a full-domain hash function $\mathcal{H} : \{0,1\}^\star \to \mathbb{Z}_N^\star$, and computes the e-th root, $\sigma = \mathcal{H}(m)^d \bmod N$. The verification algorithm simply checks whether the following equality holds, $\mathcal{H}(m) = \sigma^e \bmod N$.

More generally, the Full-Domain Hash signature can be defined as described on figure 10.4, for any trapdoor one-way permutation f.

For this scheme, Bellare and Rogaway proved, in the random-oracle model:

Theorem 10.4 *Let $\mathcal{A}$ be an adversary which can produce, with success probability ε, an existential forgery under a chosen-message attack within a time t, after q_h and q_s queries to the hash function and the signing oracle respectively. Then the permutation f can be inverted with probability ε' within time t' where*

$$\varepsilon' \geq \frac{\varepsilon}{q_s + q_h + 1} \qquad \textit{and} \qquad t' \leq t + (q_s + q_h) T_f,$$

$\mathcal{K}$: **Key Generation** $\rightarrow (f, f^{-1})$	
public key	$f : X \longrightarrow X$, a trapdoor one-way permutation onto X
private key	f^{-1}
$\rightarrow (f, f^{-1})$	

$\mathcal{S}$: **Signature of** $m \rightarrow \sigma$
$r = \mathcal{H}(m)$ and $\sigma = f^{-1}(r)$
$\rightarrow \sigma$ is the signature of m

$\mathcal{V}$: **Verification of** (m, σ)
check whether $f(\sigma) \stackrel{?}{=} \mathcal{H}(m)$
$\rightarrow$ Yes/No

Figure 10.4: The FDH Signature.

with T_f the time for an evaluation of f.

10.4.1 Basic Proof of the FDH Signature

In this proof, we incrementally define a sequence of games starting at the real game $\mathbf{G}_0$ and ending up at $\mathbf{G}_5$. We make a very detailed sequence of games in this proof, since this is the first one. Some steps will be skipped in the other proofs. The goal of this proof is to reduce the inversion of the permutation f on an element y (find x such that $y = f(x)$) to an attack. We are thus given such a random challenge y.

Game $\mathbf{G}_0$: This is the real attack game, in the random-oracle model, which includes the verification step. This means that the attack game consists in giving the public key to the adversary, and a full access to the signing oracle. When it outputs its forgery, one furthermore checks whether it is actually valid or not. Note that if the adversary asks q_s queries to the signing oracle and q_h queries to the hash oracle, at most $q_s + q_h + 1$ queries are asked to the hash oracle during this game, since each signing query may make such a new query, and the last verification step too. We are interested in the following event: S_0 which occurs if the verification step succeeds (and the signature is new).

$$\mathsf{Succ}^{\mathsf{euf}}_{\mathsf{fdh}}(\mathcal{A}) = \Pr[\mathsf{S}_0]. \tag{10.1}$$

Game $\mathbf{G}_1$: In this game, we simulate the oracles, the hash oracle $\mathcal{H}$ and the signing oracle $\mathcal{S}$, and the last verification step, as shown on Figure 10.5. From this simulation, we easily see that the game is perfectly indistinguishable from the real attack.

$$\Pr[\mathsf{S}_1] = \Pr[\mathsf{S}_0]. \tag{10.2}$$

Game $\mathbf{G}_2$: Since the verification process is included in the attack game, the output message is necessarily asked to the hash oracle. Let us guess the index c of this (first) query. If the guess failed, we abort the game. Therefore, only a correct guess (event $\mathsf{GoodGuess}$) may lead to a success.

$$\begin{aligned} \Pr[\mathsf{S}_2] &= \Pr[\mathsf{S}_1 \wedge \mathsf{GoodGuess}] = \Pr[\mathsf{S}_1 \mid \mathsf{GoodGuess}] \times \Pr[\mathsf{GoodGuess}] \\ &\geq \Pr[\mathsf{S}_1] \times \frac{1}{q_h + q_s + 1}. \end{aligned} \tag{10.3}$$

Game $\mathbf{G}_3$: We can now simulate the hash oracle, incorporating the challenge y, for which we want to extract the pre-image x by f:

$\mathcal{H}$ oracle

For a hash-query $\mathcal{H}(q)$, such that a record $(q, \star, r)$ appears in H-List, the answer is r. Otherwise the answer r is defined according to the following rule:

▶**Rule** $\mathcal{H}^{(1)}$

> Choose a random element $r \in X$. The record $(q, \perp, r)$ is added to H-List.

Note: the second component of the elements of this list will be explained later.

$\mathcal{S}$ oracle

For a sign-query $\mathcal{S}(m)$, one first asks for $r = \mathcal{H}(m)$ to the $\mathcal{H}$-oracle, and then the signature σ is defined according to the following rule:

▶**Rule** $\mathcal{S}^{(1)}$

> Computes $\sigma = f^{-1}(r)$.

$\mathcal{V}$ oracle

The game ends with the verification of the output (m, σ) from the adversary. One first asks for $r = \mathcal{H}(m)$, and checks whether $r = f(\sigma)$.

Figure 10.5: Simulation of the Attack Game against FDH

▶**Rule** $\mathcal{H}^{(3)}$

> If this is the c-th query, set $r \leftarrow y$; otherwise, choose a random element $r \in X$. The record $(q, \perp, r)$ is added to H-List.

Because of the random choice for the challenge y, this rule lets the game indistinguishable from the previous one.

$$\Pr[\mathsf{S}_3] = \Pr[\mathsf{S}_2]. \tag{10.4}$$

Game $\mathbf{G}_4$: We now modify the simulation of the hash oracle for other queries, which may be used in signing queries:

▶**Rule** $\mathcal{H}^{(4)}$

> If this is the c-th query, set $r \leftarrow y$ and $s \leftarrow \perp$; otherwise, choose a random element $s \in X$, and compute $r = f(s)$. The record (q, s, r) is added to H-List.

Because of the permutation property of f, and the random choice for s, this rule lets the game indistinguishable from the previous one.

$$\Pr[\mathsf{S}_4] = \Pr[\mathsf{S}_3]. \tag{10.5}$$

Game $\mathbf{G}_5$: By now, excepted for the c-th hash query, which will be involved in the forgery (and thus not asked to the signing oracle), the pre-image is known. One can thus simulate the signing oracle without quering f^{-1}:

▶**Rule** $\mathcal{S}^{(5)}$

> Lookup for (m, s, r) in H-List, and set $\sigma = s$.

Since the message corresponding to the c-th query cannot be asked to the signing oracle, otherwise it would not be a valid forgery, this rule lets the game indistinguishable from the previous one.

$$\Pr[\mathsf{S}_5] = \Pr[\mathsf{S}_4]. \tag{10.6}$$

Note that now, the simulation can easily be performed, without any specific computational power or oracle access. Just a few more evaluations of f are done to simulate the hash oracle, and the forgery leads to the pre-image of y:

$$\Pr[\mathsf{S}_5] = \mathsf{Succ}_f^{\mathsf{ow}}(t + (q_h + q_s)T_f). \tag{10.7}$$

As a consequence, using equations (10.1), (10.2), (10.3), (10.4), (10.5), (10.6) and (10.7)

$$\begin{aligned}\mathsf{Succ}_f^{\mathsf{ow}}(t + (q_h + q_s)T_f) &= \Pr[\mathsf{S}_5] = \Pr[\mathsf{S}_3] = \Pr[\mathsf{S}_4] = \Pr[\mathsf{S}_2]\\ &\geq \frac{1}{q_h + q_s + 1} \times \Pr[\mathsf{S}_1] \geq \frac{1}{q_h + q_s + 1} \times \Pr[\mathsf{S}_0].\end{aligned}$$

And thus,

$$\mathsf{Succ}_{\mathsf{fdh}}^{\mathsf{euf}}(\mathcal{A}) \leq (q_h + q_s + 1) \times \mathsf{Succ}_f^{\mathsf{ow}}(t + (q_h + q_s)T_f).$$

□

10.4.2 Improved Security Result

This reduction has been thereafter improved [?], thanks to the random self-reducibility of the RSA function. The following result applies as soon as the one-way permutation has some homomorphic property on the group X:

$$f(x \otimes y) = f(x) \otimes f(y).$$

Theorem 10.5 *Let $\mathcal{A}$ be an adversary which can produce, with success probability ε, an existential forgery under a chosen-message attack within a time t, after q_h and q_s queries to the hash function and the signing oracle respectively. Then the permutation f can be inverted with probability ε' within time t' where*

$$\varepsilon' \geq \frac{\varepsilon}{q_s} \times \exp(-2) \quad \text{and} \quad t' \leq t + (q_s + q_h)T_f,$$

with T_f the time for an evaluation of f.

This proof can be performed as the previous one, and thus starts at the real game $\mathbf{G}_0$, then we can use the same simulation as in the game $\mathbf{G}_1$. The sole formal difference in the simulation will be the $\mathsf{H\text{-}List}$ which elements have one more field, and are thus initially of the form $(q, \perp, \perp, r)$. Things differ much after that, using a real value p between 0 and 1, which will be made precise later. The idea here, is to make any forgery useful for inverting the permutation f, not only a specific (guessed) one. On the other hand, one must still be able to simulate the signing oracle. The probability p will separate the two situations:

Game G$_2$: A random coin decides whether we introduce the challenge y in the hash answer, or an element with a known pre-image:

> ►**Rule** $\mathcal{H}^{(2)}$
>
> One chooses a random $s \in X$. With probability p, one sets $r \leftarrow y \otimes f(s)$ and $t \leftarrow 1$; otherwise, $r \leftarrow f(s)$ and $t \leftarrow 0$. The record (q, t, s, r) is added to $\mathsf{H\text{-}List}$.

Because of the homomorphic property on the group X of the permutation f, this rule lets the game indistinguishable from the previous one. Note again that elements in $\mathsf{H\text{-}List}$ contain one more field t than in the previous proof. One may see that $r = y^t \otimes f(s)$.

Game G$_3$: For a proportion $1-p$ of the signature queries, one can simulate the signing oracle without having to invert the permutation f:

▶**Rule** $\mathcal{S}^{(3)}$

> Lookup for (m, t, s, r) in H-List, if $t = 1$ then halt the game, otherwise set $\sigma = s$.

This rule lets the game indistinguishable, unless one signing query fails ($t = 1$), which happens with probability p, for each signature:

$$\Pr[\mathsf{S}_3] = (1 - p)^{q_s} \times \Pr[\mathsf{S}_2]. \tag{10.8}$$

Note that now, the simulation can easily be performed, without any specific computational power or oracle access. Just a few more exponentiations are done to simulate the hash oracle, and the forgery (m, σ) leads to the pre-image of y, if $(t = 1)$. The latter case holds with probability p. Indeed, (m, t, s, r) can be found in the H-List, and then $r = y^t \otimes f(s) = y \otimes f(s) = f(\sigma)$, which easily leads to the pre-image of y by f:

$$\mathsf{Succ}_f^{\mathsf{ow}}(t + (q_h + q_s)T_f) = p \times \Pr[\mathsf{S}_3]. \tag{10.9}$$

Using equations (10.1), (10.2), (10.8) and (10.9)

$$\begin{aligned} \mathsf{Succ}_f^{\mathsf{ow}}(t + (q_h + q_s)T_f) &= p \times \Pr[\mathsf{S}_3] = p \times (1 - p)^{q_s} \times \Pr[\mathsf{S}_2] \\ &= p \times (1 - p)^{q_s} \times \Pr[\mathsf{S}_1] = p \times (1 - p)^{q_s} \times \Pr[\mathsf{S}_0]. \end{aligned}$$

And thus,

$$\mathsf{Succ}_{\mathsf{fdh}}^{\mathsf{euf}}(\mathcal{A}) \leq \frac{1}{p(1-p)^{q_s}} \times \mathsf{Succ}_f^{\mathsf{ow}}(t + (q_h + q_s)T_f).$$

Therefore, the success probability of our inversion algorithm is $p(1-p)^{q_s}\varepsilon$, if ε is the success probability of the adversary. If $q_s > 0$, the latter expression is optimal for $p = 1/(q_s + 1)$. And for this parameter, and a huge value q_s, the success probability is approximately ε/eq_s. It is anyway larger than $\varepsilon/e^2 q_s$ (where $e = \exp(1) \approx 2.17\ldots$).

As far as time complexity is concerned, each random oracle simulation (which can be launched by a signing simulation) requires a modular exponentiation to the power e, hence the result. □

This is a great improvement since the success probability does not depend anymore on q_h. Furthermore, q_s can be limited by the user, whereas q_h cannot. In practice, one only assumes $q_h \leq 2^{60}$, but q_s can be limited below 2^{30}.

10.4.3 PSS: The Probabilistic Signature Scheme

However, one would like to get more, suppressing any coefficient. In their paper [?], Bellare and Rogaway proposed such a better candidate, the Probabilistic Signature Scheme (PSS, see Figure 10.6): the key generation is still the same, but the signature process involves three hash functions

$$\mathcal{F} : \{0,1\}^{k_2} \to \{0,1\}^{k_0}, \quad \mathcal{G} : \{0,1\}^{k_2} \to \{0,1\}^{k_1}, \\ \mathcal{H} : \{0,1\}^\star \to \{0,1\}^{k_2},$$

where $k = k_0 + k_1 + k_2 + 1$ satisfies $\{0,1\}^{k-1} \subset X \subset \{0,1\}^k$. We remind that f is a trapdoor one-way permutation onto X, with an homomorphic relationship. For each message m to be signed, one chooses a random string $r \in \{0,1\}^{k_1}$. One first computes $w = \mathcal{H}(m, r)$, $s = \mathcal{G}(w) \oplus r$ and $t = \mathcal{F}(w)$. Then one concatenates $y = 0\|w\|s\|t$, where $a\|b$ denotes the concatenation of the bit strings a and b. Finally, one computes the pre-image by f, $\sigma = f^{-1}(y)$. The verification algorithm first computes $y = f(\sigma)$, and parses it as $y = b\|w\|s\|t$. Then, one can get $r = s \oplus \mathcal{G}(w)$, and checks whether $b = 0$, $w = \mathcal{H}(m, r)$ and $t = \mathcal{F}(w)$.

About this PSS construction, Bellare and Rogaway proved the security in the random-oracle model.

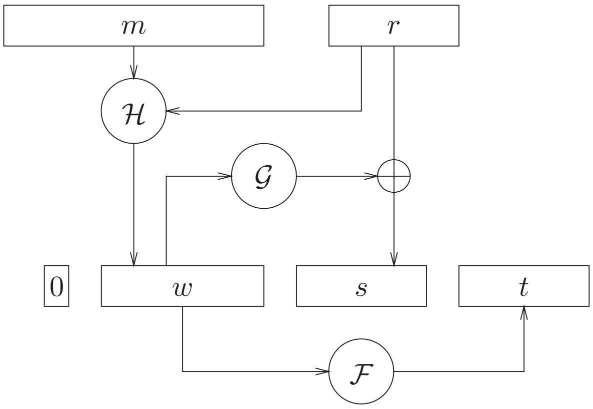


Figure 10.6: Probabilistic Signature Scheme

Theorem 10.6 *Let $\mathcal{A}$ be a* CMA*-adversary against f–*PSS *which produces an existential forgery within a time t, after q_f, q_g, q_h and q_s queries to the hash functions $\mathcal{F}$, $\mathcal{G}$ and $\mathcal{H}$ and the signing oracle respectively. Then its success probability is upper-bounded by*

$$\mathsf{Succ}_f^{\mathsf{ow}}(t + (q_s + q_h)k_2 \cdot T_f) + \frac{1}{2^{k_2}} + (q_s + q_h) \cdot \left(\frac{q_s}{2^{k_1}} + \frac{q_f + q_g + q_h + q_s + 1}{2^{k_2}} \right),$$

with T_f the time for an evaluation of f.

The important point in this security result is the very tight link between success probabilities, but also the almost linear time of the reduction. Thanks to this exact and efficient security result, RSA–PSS has become the new PKCS #1 v2.1 standard for signature [**?**]. Another variant has been proposed with message-recovery: PSS-R which allows one to include a large part of the message inside the signature. This makes a signed-message shorter than the size of the signature plus the size of the message, since the latter is inside the former one.

Chapter 11

Automating Game-Based Proofs

11.1 Introduction

Let us recall that there exist two main models for analyzing security protocols:

- In the symbolic model, often called *Dolev-Yao* model [?], cryptographic primitives are considered as perfect blackboxes, modeled by function symbols in an algebra of terms, possibly with equations. Messages are terms on these primitives and the adversary can compute only using these primitives.

- In contrast, in the *computational* model, messages are bitstrings, cryptographic primitives are functions from bitstrings to bitstrings, and the adversary is any probabilistic Turing machine.

The computational model is close to the real execution of protocols, but the proofs are usually manual and informal. The Dolev-Yao model is an abstract model that makes it easier to build automatic verification tools, but the security proofs are in general not sound with respect to the computational model. Automatic verification in the Dolev-Yao model was the subject of Part II.

In order to mechanize proofs in the computational model, several approaches have been considered.

- In the indirect approach, following the seminal paper by Abadi and Rogaway [?], one shows the soundness of the Dolev-Yao model with respect to the computational model, that is, one proves that the security of a protocol in the Dolev-Yao model implies its security in the computational model, modulo additional assumptions. Combining such a result with a Dolev-Yao automatic verifier, one obtains automatic proofs of protocols in the computational model. This approach received much interest [?, ?, ?, ?, ?, ?] and a tool [?] was developed based on [?] to obtain computational proofs using the Dolev-Yao verifier AVISPA, for protocols that rely on public-key encryption and signatures. However, this approach has limitations: since the computational and Dolev-Yao models do not correspond exactly, soundness requires additional hypotheses. (For example, key cycles have to be excluded, or a specific security definition of encryption is needed [?].) This approach is studied in Part IV.

  In a related approach, Backes, Pfitzmann, and Waidner [?, ?, ?] have designed an abstract cryptographic library including symmetric and public-key encryption, message authentication codes, signatures, and nonces and shown its soundness with respect to computational primitives, under arbitrary active attacks. This framework has been used for a computationally-sound machine-checked proof of the Needham-Schroeder-Lowe protocol [?].

Canetti [?] introduced the notion of universal composability. With Herzog [?], they show how a Dolev-Yao-style symbolic analysis can be used to prove security properties of protocols within the framework of universal composability, for a restricted class of protocols using public-key encryption as only cryptographic primitive. Then, they use the automatic Dolev-Yao verification tool ProVerif [?] for verifying protocols in this framework.

- Techniques used previously in the Dolev-Yao model have also been adapted in order to obtain proofs in the computational model.

  For instance, Datta, Derek, Mitchell, Shmatikov, and Turuani [?, ?] have adapted the logic PCL (Protocol Composition Logic), first designed for proving protocols in the Dolev-Yao model, to the computational model. Other computationally sound logics include CIL (Computational Indistinguishability Logic) [?] and a specialized Hoare logic designed for proving asymmetric encryption schemes in the random oracle model [?, ?].

  Similarly, type systems [?, ?, ?, ?] can provide computational security guarantees. For instance, [?] handles shared-key and public-key encryption, with an unbounded number of sessions. This system relies on the Backes-Pfitzmann-Waidner library. A type inference algorithm is given in [?].

- In the direct approach, one aims at mechanizing proofs in the computational model, without using a Dolev-Yao protocol verifier. As detailed in the previous chapters, computational proofs made by cryptographers are typically presented as sequences of games [?, ?]: the initial game represents the protocol to prove; the goal is to show that the probability of breaking a certain security property is negligible in this game; intermediate games are obtained each from the previous one by transformations such that the difference of probability between consecutive games is negligible; the final game is such that the desired probability is obviously negligible from the form of the game. The desired probability is then negligible in the initial game. Halevi [?] suggested to use tools for mechanizing these proofs, and several techniques have been used for reaching this goal.

  This chapter presents such a tool, CryptoVerif [?, ?, ?, ?]. CryptoVerif generates proofs by sequences of games automatically or with little user interaction. The games are formalized in a probabilistic process calculus. CryptoVerif provides a generic method for specifying security properties of many cryptographic primitives. It proves secrecy and authentication properties. It also provides a bound on the probability of success of an attack. It considerably extends early works by Laud [?, ?] which were limited either to passive adversaries or to a single session of the protocol. More recently, Tšahhirov and Laud [?, ?] developed a tool similar to CryptoVerif but that represents games by dependency graphs; it handles only public-key and shared-key encryption and proves secrecy properties.

  The tool CertiCrypt [?, ?, ?, ?, ?] enables the machine-checked construction and verification of cryptographic proofs by sequences of games. It relies on the general-purpose proof assistant Coq, which is widely believed to be correct. EasyCrypt [?] generates CertiCrypt proofs from proof sketches that formally represent the sequence of games and hints, which makes the tool easier to use. Nowak *et al.* [?, ?, ?] follow a similar idea by providing Coq proofs for several basic cryptographic primitives.

In the tool CryptoVerif, games are represented in a process calculus inspired by the pi-calculus and by the calculi of [?] and of [?]. In this calculus, messages are bitstrings, and cryptographic primitives are functions from bitstrings to bitstrings. The calculus has a probabilistic semantics. The main tool for specifying security assumptions is observational equivalence: Q is observationally equivalent to Q' up to probability p, $Q \approx_p Q'$, when the adversary has probability at most p of distinguishing Q from Q'. With respect to previous calculi mentioned above, our calculus introduces an important novelty which is key for the automatic proof of security protocols: the values of all variables during the execution of a process are stored in arrays. For instance, $x[i]$ is the value of x in the i-th copy of the process that defines x. Arrays replace lists often used by cryptographers in their manual proofs of protocols. For example, consider the standard security assumption on a message authentication code (MAC). Informally, this assumption says that the adversary has a negligible probability of forging a MAC, that is, that all correct MACs have been computed by calling the MAC oracle. So, in cryptographic proofs, one defines a list containing the arguments of calls to the MAC oracle, and when verifying a MAC of a message m, one can additionally check that m is in this list, with a negligible change in probability. In our calculus, the arguments of the MAC oracle are stored in arrays, and we perform a lookup in these arrays in order to find the message m. Arrays make it easier to automate proofs since they are always present in the calculus: one does not need to add explicit instructions to insert values in them, in contrast to the lists used in manual proofs. Therefore, many trivially sound but difficult to automate syntactic transformations disappear. Furthermore, relations between elements of arrays can easily be expressed by equalities, possibly involving computations on array indices.

CryptoVerif relies on a collection of game transformations, in order to transform the initial protocol into a game on which the desired security property is obvious. The most important kind of transformations exploits the security assumptions on cryptographic primitives in order to obtain a simpler game. As described in Section 11.3.2, these transformations can be specified in a generic way: we represent the security assumption of each cryptographic primitive by an observational equivalence $L \approx_p R$, where the processes L and R encode oracles: they input the arguments of the oracle and send its result back. Then, the prover can automatically transform a process Q that calls the oracles of L (more precisely, contains as subterms terms that perform the same computations as oracles of L) into a process Q' that calls the oracles of R instead. We have used this technique to specify several variants of shared-key and public-key encryption, signature, message authentication codes, hash functions, Diffie-Hellman key agreement, simply by giving the appropriate equivalence $L \approx_p R$ to the prover. Other game transformations are syntactic transformations, used in order to be able to apply an assumption on a cryptographic primitive, or to simplify the game obtained after applying such an assumption.

In order to prove protocols, these game transformations are organized using a proof strategy based on advice: when a transformation fails, it suggests other transformations that should be applied before, in order to enable the desired transformation. Thanks to this strategy, protocols can often be proved in a fully automatic way. For delicate cases, CryptoVerif has an interactive mode, in which the user can manually specify the transformations to apply. It is usually sufficient to specify a few transformations coming from the security assumptions of primitives, by indicating the concerned cryptographic primitive and the concerned secret key if any; the prover infers the intermediate syntactic transformations by the advice strategy. This mode is helpful for proving some public-key protocols, in which several security assumptions on primitives can be applied, but only one leads to a proof of the protocol. Importantly, CryptoVerif is always sound: whatever indications the user gives, when the prover shows a security property of the protocol, the property indeed holds assuming the given assumptions on the cryptographic primitives.

CryptoVerif has been implemented in Ocaml (29800 lines of code for version 1.12 of CryptoVerif) and is available at `http://cryptoverif.inria.fr/`.

Outline The next section presents the process calculus for representing games. Section 11.3 describes the game transformations that serve for proving protocols. Section 11.4 gives criteria for proving secrecy properties of protocols. Section 11.5 explains how the prover chooses which transformation to apply at each point. Section 11.6 presents applications of CryptoVerif and Section 11.7 concludes.

$M, N ::=$	terms
i	replication index
$x[M_1, \dots, M_m]$	variable access
$f(M_1, \dots, M_m)$	function application
$Q ::=$	input process
0	nil
$Q \parallel Q'$	parallel composition
$!^{i \le n} Q$	replication n times
$\mathsf{newChannel}\ c; Q$	channel restriction
$\mathsf{in}(c[M_1, \dots, M_l], (x_1[\widetilde{i}] : T_1, \dots, x_k[\widetilde{i}] : T_k)); P$	input
$P ::=$	output process
$\mathsf{out}(c[M_1, \dots, M_l], (N_1, \dots, N_k)); Q$	output
$\mathsf{new}\ x[i_1, \dots, i_m] : T; P$	random number
$\mathsf{let}\ x[i_1, \dots, i_m] : T = M\ \mathsf{in}\ P$	assignment
$\mathsf{if}\ \mathsf{defined}(M_1, \dots, M_l) \wedge M\ \mathsf{then}\ P\ \mathsf{else}\ P'$	conditional
$\mathsf{find}\ (\bigoplus_{j=1}^{m} u_{j1}[\widetilde{i}] \le n_{j1}, \dots, u_{jm_j}[\widetilde{i}] \le n_{jm_j}\ \mathsf{suchthat}$ $\mathsf{defined}(M_{j1}, \dots, M_{jl_j}) \wedge M_j\ \mathsf{then}\ P_j)\ \mathsf{else}\ P$	array lookup
$\mathsf{event}\ e(M_1, \dots, M_l); P$	event

Figure 11.1: Syntax of the process calculus

Notations We recall the following standard notations. We denote by $\{M_1/x_1, \dots, M_m/x_m\}$ the substitution that replaces x_j with M_j for each $j \le m$. The cardinal of a set or multiset S is denoted by $|S|$. If S is a finite set, $x \overset{R}{\leftarrow} S$ chooses a random element uniformly in S and assigns it to x. If $\mathcal{A}$ is a probabilistic algorithm, $x \leftarrow \mathcal{A}(x_1, \dots, x_m)$ denotes the experiment of choosing random coins r and assigning to x the result of running $\mathcal{A}(x_1, \dots, x_m)$ with coins r. Otherwise, $x \leftarrow M$ is a simple assignment statement.

11.2 A Calculus for Games

11.2.1 Syntax and Informal Semantics

CryptoVerif represents games in the syntax of Figure 11.1. This calculus assumes a countable set of channel names, denoted by c. It uses parameters, denoted by n, which are integers that bound the number of executions of processes. It also uses types, denoted by T, which are subsets of $\mathit{bitstring}_\perp = \mathit{bitstring} \cup \{\perp\}$ where $\mathit{bitstring}$ is the set of all bitstrings and $\perp$ is a special symbol. Let *fixed-length* types be types that consist of the set of all bitstrings of a certain length. Particular types are predefined: $\mathit{bool} = \{\mathrm{true}, \mathrm{false}\}$, where false is 0 and true is 1; $\mathit{bitstring}$; $\mathit{bitstring}_\perp$; $[1, n]$ where n is a parameter. (We consider integers as bitstrings without leading zeroes.)

The calculus also uses function symbols f. Each function symbol comes with a type declaration $f : T_1 \times \dots \times T_m \to T$, and represents an efficiently computable, deterministic function that maps each tuple in $T_1 \times \dots \times T_m$ to an element of T. Particular functions are predefined, and some of them use the infix notation: $M = N$ for the equality test, $M \neq N$ for the inequality test (both taking two values of the same type T and returning a value of type *bool*), $M \vee N$ for the boolean or, $M \wedge N$ for the boolean and, $\neg M$ for the boolean negation (taking and returning values of type *bool*).

In this calculus, terms represent computations on bitstrings. The replication index i is an integer which serves in distinguishing different copies of a replicated process $!^{i \leq n}$. (Replication indices are typically used as array indices.) The variable access $x[M_1, \dots, M_m]$ returns the content of the cell of indices $M_1, \dots, M_m$ of the m-dimensional array variable x. We use x, y, z, u as variable names. The function application $f(M_1, \dots, M_m)$ returns the result of applying function f to $M_1, \dots, M_m$.

The calculus distinguishes two kinds of processes: input processes Q are ready to receive a message on a channel; output processes P output a message on a channel after executing some internal computations. The input process 0 does nothing; $Q \parallel Q'$ is the parallel composition of Q and Q'; $!^{i \leq n} Q$ represents n copies of Q in parallel, each with a different value of $i \in [1, n]$; $\mathsf{newChannel}\ c; Q$ creates a new private channel c and executes Q; the semantics of the input $\mathsf{in}(c[M_1, \dots, M_l], (x_1[\widetilde{i}] : T_1, \dots, x_k[\widetilde{i}] : T_k)); P$ will be explained below together with the semantics of the output.

The output process $\mathsf{new}\ x[i_1, \dots, i_m] : T; P$ chooses a new random number uniformly in T, stores it in $x[i_1, \dots, i_m]$, and executes P. (The type T must be a fixed-length type, because probabilistic Turing machines can choose random numbers uniformly only in such types.) Function symbols represent deterministic functions, so all random numbers must be chosen by $\mathsf{new}\ x[i_1, \dots, i_m] : T$. Deterministic functions make automatic syntactic manipulations easier: we can duplicate a term without changing its value. The process $\mathsf{let}\ x[i_1, \dots, i_m] : T = M\ \mathsf{in}\ P$ stores the bitstring value of M (which must be in T) in $x[i_1, \dots, i_m]$ and executes P. The process $\mathsf{event}\ e(M_1, \dots, M_l); P$ executes the event $e(M_1, \dots, M_l)$, then runs P. This event records that a certain program point has been reached with certain values of $M_1, \dots, M_l$, but otherwise does not affect the execution of the process. Next, we explain the process $\mathsf{find}\ (\bigoplus_{j=1}^{m} u_{j1}[\widetilde{i}] \leq n_{j1}, \dots, u_{jm_j}[\widetilde{i}] \leq n_{jm_j}\ \mathsf{suchthat}\ \mathsf{defined}(M_{j1}, \dots, M_{jl_j}) \wedge M_j\ \mathsf{then}\ P_j)\ \mathsf{else}\ P$, where $\widetilde{i}$ denotes a tuple $i_1, \dots, i_{m'}$. The order and array indices on tuples are taken component-wise, so for instance, $u_{j1}[\widetilde{i}] \leq n_{j1}, \dots, u_{jm_j}[\widetilde{i}] \leq n_{jm_j}$ can be further abbreviated $\widetilde{u_j}[\widetilde{i}] \leq \widetilde{n_j}$. A simple example is the following: $\mathsf{find}\ u \leq n\ \mathsf{suchthat}\ \mathsf{defined}(x[u]) \wedge x[u] = a\ \mathsf{then}\ P'\ \mathsf{else}\ P$ tries to find an index u such that $x[u]$ is defined and $x[u] = a$, and when such a u is found, it executes P' with that value of u; otherwise, it executes P. In other words, this find construct looks for the value a in the array x, and when a is found, it stores in u an index such that $x[u] = a$. Therefore, the find construct allows us to access arrays, which is key for our purpose. More generally, $\mathsf{find}\ u_1[\widetilde{i}] \leq n_1, \dots, u_m[\widetilde{i}] \leq n_m\ \mathsf{suchthat}\ \mathsf{defined}(M_1, \dots, M_l) \wedge M\ \mathsf{then}\ P'\ \mathsf{else}\ P$ tries to find values of $u_1, \dots, u_m$ for which $M_1, \dots, M_l$ are defined and M is true. In case of success, it executes P'. In case of failure, it executes P. This is further generalized to m branches: $\mathsf{find}\ (\bigoplus_{j=1}^{m} u_{j1}[\widetilde{i}] \leq n_{j1}, \dots, u_{jm_j}[\widetilde{i}] \leq n_{jm_j}\ \mathsf{suchthat}\ \mathsf{defined}(M_{j1}, \dots, M_{jl_j}) \wedge M_j\ \mathsf{then}\ P_j)\ \mathsf{else}\ P$ tries to find a branch j in $[1, m]$ such that there are values of $u_{j1}, \dots, u_{jm_j}$ for which $M_{j1}, \dots, M_{jl_j}$ are defined and M_j is true. In case of success, it executes P_j. In case of failure for all branches, it executes P. More formally, it evaluates the conditions $\mathsf{defined}(M_{j1}, \dots, M_{jl_j}) \wedge M_j$ for each j and each value of $u_{j1}[\widetilde{i}], \dots, u_{jm_j}[\widetilde{i}]$ in $[1, n_{j1}] \times \ldots \times [1, n_{jm_j}]$. If none of these conditions is true, it executes P. Otherwise, it chooses randomly with uniform[1] probability one j and one value of $u_{j1}[\widetilde{i}], \dots, u_{jm_j}[\widetilde{i}]$ such that the corresponding condition is true and executes P_j. The conditional $\mathsf{if}\ \mathsf{defined}(M_1, \dots, M_l) \wedge M\ \mathsf{then}\ P\ \mathsf{else}\ P'$ executes P if $M_1, \dots, M_l$ are defined and M evaluates to true. Otherwise, it executes P'. This conditional is equivalent to $\mathsf{find}\ \mathsf{suchthat}\ \mathsf{defined}(M_1, \dots, M_l) \wedge M\ \mathsf{then}\ P\ \mathsf{else}\ P'$. The conjunct $\mathsf{defined}(M_1, \dots, M_l)$ can be omitted when $l = 0$ and M can be omitted when it is true.

Finally, let us explain the output $\mathsf{out}(c[M_1, \dots, M_l], (N_1, \dots, N_k)); Q$. A channel $c[M_1, \dots, M_l]$ consists of both a channel name c and a tuple of terms $M_1, \dots, M_l$. Channel names c can

[1] A probabilistic Turing machine can choose a random number uniformly in a set of cardinal m only when m is a power of 2. When m is not a power of 2, there exist approximate algorithms: for example, in order to obtain a random integer in $[0, m-1]$, we can choose a random integer r uniformly among $[0, 2^k - 1]$ for a certain k large enough and return $r \mod m$. The distribution can be made as close as we wish to the uniform distribution by choosing k large enough.

be declared private by $\mathsf{newChannel}\ c$; the adversary can never have access to channel $c[M_1, \dots, M_l]$ when c is private. (This is useful in the proofs, although all channels of protocols are often public.) Terms $M_1, \dots, M_l$ are intuitively analogous to IP addresses and ports, which are numbers that the adversary may guess. A semantic configuration always consists of a single output process (the process currently being executed) and several input processes. When the output process executes $\mathsf{out}(c[M_1, \dots, M_l], (N_1, \dots, N_k)); Q$, one looks for an input on channel $c[M'_l \dots, M'_l]$, where $M'_1, \dots, M'_l$ evaluate to the same bitstrings as $M_1, \dots, M_l$, and with the same arity k, in the available input processes. If no such input process is found, the process blocks. Otherwise, one such input process $\mathsf{in}(c[M'_1, \dots, M'_l], (x_1[\widetilde{i}] : T_1, \dots, x_k[\widetilde{i}] : T_k)); P$ is chosen randomly with uniform probability. The communication is then executed: for each $j \leq k$, the output message N_j is evaluated and stored in $x_j[\widetilde{i}]$ if it is in T_j (otherwise the process blocks). Finally, the output process P that follows the input is executed. The input process Q that follows the output is stored in the available input processes for future execution. The syntax requires an output to be followed by an input process, as in [**?**]. If one needs to output several messages consecutively, one can simply insert fictitious inputs between the outputs. The adversary can then schedule the outputs by sending messages to these inputs.

Using different channels for each input and output allows the adversary to control the network. For instance, we may write $!^{i \leq n}\mathsf{in}(c[i], x[i] : T) \dots \mathsf{out}(c'[i], M) \dots$ The adversary can then decide which copy of the replicated process receives its message, simply by sending it on $c[i]$ for the appropriate value of i.

An else branch of find or if may be omitted when it is $\mathsf{else}\ \mathsf{out}(\mathit{yield}, ()); 0$. (Note that "$\mathsf{else}$ 0" would not be syntactically correct.) Similarly, $\mathsf{out}(\mathit{yield},); 0$ may be omitted after an event or a restriction. A trailing 0 after an output may be omitted.

The *current replication indices* at a certain program point in a process are $i_1, \dots, i_m$ where the replications above the considered program point are $!^{i_1 \leq n_1} \dots !^{i_m \leq n_m}$. We often abbreviate $x[i_1, \dots, i_m]$ by x when $i_1, \dots, i_m$ are the current replication indices, but it should be kept in mind that this is only an abbreviation. Variables x defined under a replication must be arrays with indices the current replication indices at the definition of x: for example $!^{i_1 \leq n_1} \dots !^{i_m \leq n_m}\mathsf{let}\ x[i_1, \dots, i_m] : T = M\ \mathsf{in}\ \dots$ More formally, we require the following invariant:

Invariant 11.1 (Single definition) *The process Q_0 satisfies Invariant 11.1 if and only if*

1. *in every definition of $x[i_1, \dots, i_m]$ in Q_0, the indices $i_1, \dots, i_m$ of x are the current replication indices at that definition, and*

2. *two different definitions of the same variable x in Q_0 are in different branches of a* find *(or* if*).*

Invariant 11.1 guarantees that each variable is assigned at most once for each value of its indices. (Indeed, item 2 shows that only one definition of each variable can be executed for given indices in each trace.)

Invariant 11.2 (Defined variables) *The process Q_0 satisfies Invariant 11.2 if and only if every occurrence of a variable access $x[M_1, \dots, M_m]$ in Q_0 is either*

- *syntactically under the definition of $x[M_1, \dots, M_m]$ (in which case $M_1, \dots, M_m$ are in fact the current replication indices at the definition of x);*

- *or in a* $\mathsf{defined}$ *condition in a* find *process;*

- *or in M'_j or P_j in a process of the form* $\mathsf{find}\ (\bigoplus_{j=1}^{m''} \widetilde{u_j}[\widetilde{i}] \leq \widetilde{n_j}\ \mathsf{suchthat}\ \mathsf{defined}(M'_{j1}, \dots, M'_{jl_j}) \wedge M'_j\ \mathsf{then}\ P_j)\ \mathsf{else}\ P$ *where for some $k \leq l_j$, $x[M_1, \dots, M_m]$ is a subterm of M'_{jk}.*

Invariant 11.2 guarantees that variables can be accessed only when they have been initialized. It checks that the definition of the variable access is either in scope (first item) or checked by a find (last item).

We use a type system, detailed in [**?**, Appendix A], to check that bitstrings of the proper type are given to each function and that array indices are used correctly.

Invariant 11.3 (Typing) *The process Q_0 satisfies Invariant 11.3 if and only if it is well-typed.*

We require the adversary to be well-typed. This requirement does not restrict its computing power, because it can always define type-cast functions $f : T \to T'$ to bypass the type system. Similarly, the type system does not restrict the class of protocols that we consider, since the protocol may contain type-cast functions. The type system just makes explicit which set of bitstrings may appear at each point of the protocol. The three invariants are checked by the prover for the initial game and preserved by all game transformations.

The formal semantics is defined by a probabilistic reduction relation; the asymptotic version of the semantics is detailed in [**?**, Appendix B] (see page 154 for an explanation of asymptotic versus exact security). Our semantics is such that all processes can be simulated by probabilistic Turing machines, and conversely. The notation $E, M \Downarrow a$ means that the term M evaluates to the bitstring a in environment E, which associates a value to each array cell $x[\widetilde{a}]$.

We say that a function $f : T_1 \times \ldots \times T_m \to T$ is *poly-injective* when it is injective and its inverses are efficiently computable, that is, there exist functions $f_j^{-1} : T \to T_j$ $(1 \leq j \leq m)$ such that $f_j^{-1}(f(x_1, \ldots, x_m)) = x_j$ and f_j^{-1} is efficiently computable. When f is poly-injective, we define a pattern matching construct $\mathsf{let}\ f(x_1, \ldots, x_m) = M\ \mathsf{in}\ P\ \mathsf{else}\ Q$ as an abbreviation for $\mathsf{let}\ y : T = M\ \mathsf{in}\ \mathsf{let}\ x_1 : T_1 = f_1^{-1}(y)\ \mathsf{in}\ \ldots \mathsf{let}\ x_m : T_m = f_m^{-1}(y)\ \mathsf{in}\ \mathsf{if}\ f(x_1, \ldots, x_m) = y\ \mathsf{then}\ P\ \mathsf{else}\ Q$. We naturally generalize this construct to $\mathsf{let}\ N = M\ \mathsf{in}\ P\ \mathsf{else}\ Q$ where N is built from poly-injective functions and variables.

We denote by $fv(Q)$ the set of variables that occur in Q.

11.2.2 Example

Let us introduce two cryptographic primitives that we use below.

Definition 11.1 *Let T_{mr}, T_{mk}, and T_{ms} be types that correspond intuitively to random seeds, keys, and message authentication codes, respectively; T_{mr} is a fixed-length type. A message authentication code scheme* MAC *[**?**] consists of three function symbols:*

- $\mathrm{mkgen} : T_{mr} \to T_{mk}$ *is the key generation algorithm taking as argument a random bitstring and returning a key. (Usually,* mkgen *is a randomized algorithm; here, since we separate the choice of random numbers from computation,* mkgen *takes an additional argument representing the random coins.)*
- $\mathrm{mac} : \mathit{bitstring} \times T_{mk} \to T_{ms}$ *is the MAC algorithm taking as arguments a message and a key, and returning the corresponding tag. (We assume here that* mac *is deterministic; we could easily encode a randomized* mac *by adding an additional argument as for* mkgen*.)*
- $\mathrm{verify} : \mathit{bitstring} \times T_{mk} \times T_{ms} \to \mathit{bool}$ *is a verification algorithm such that* $\mathrm{verify}(m, k, t) = \mathrm{true}$ *if and only if t is a valid MAC of message m under key k. (Since* mac *is deterministic,* $\mathrm{verify}(m, k, t)$ *is typically* $\mathrm{mac}(m, k) = t$*.)*

We have $\forall m \in \mathit{bitstring}, \forall r \in T_{mr}, \mathrm{verify}(m, \mathrm{mkgen}(r), \mathrm{mac}(m, \mathrm{mkgen}(r))) = \mathrm{true}$.

The advantage of an adversary against unforgeability under chosen message attacks (UF-CMA) is

$$\mathsf{Succ}^{\mathsf{uf-cma}}_{\mathsf{MAC}}(t, q_m, q_v, l) = \max_{\mathcal{A}} \Pr\left[\begin{array}{l} r \overset{R}{\leftarrow} T_{mr}; k \leftarrow \mathrm{mkgen}(r); \\ (m, s) \leftarrow \mathcal{A}^{\mathrm{mac}(.,k),\mathrm{verify}(.,k,.)} : \mathrm{verify}(m, k, s) \\ \wedge\, m \textit{ was never queried to the oracle } \mathrm{mac}(., k) \end{array}\right]$$

where the adversary $\mathcal{A}$ is any probabilistic Turing machine that runs in time at most t, calls $\mathrm{mac}(., k)$ *at most q_m times with messages of length at most l, and calls* $\mathrm{verify}(., k, .)$ *at most q_v times with messages of length at most l.*

Informally, $\mathsf{Succ}^{\mathsf{uf-cma}}_{\mathsf{MAC}}(t, q_m, q_v, l)$ is the probability that an adversary forges a MAC, that is, returns a pair (m, s) where s is a correct MAC for m, without having queried the MAC oracle $\mathrm{mac}(., k)$ on m. Intuitively, when the MAC is secure, this probability is small: the adversary has little chance of forging a MAC. Hence, the MAC guarantees the integrity of the MACed message because one cannot compute the MAC without the secret key.

Two frameworks exist for expressing security properties. In the asymptotic framework used in [**?**, **?**], the length of keys is determined by a security parameter η, and a MAC is UF-CMA when $\mathsf{Succ}^{\mathsf{uf-cma}}_{\mathsf{MAC}}(t, q_m, q_v, l)$ is a negligible function of η when t is polynomial in η. ($f(\eta)$ is *negligible* when for all polynomials q, there exists $\eta_o \in \mathbb{N}$ such that for all $\eta > \eta_0$, $f(\eta) \leq \frac{1}{q(\eta)}$.) The assumption that functions are efficiently computable means that they are computable in time polynomial in η and in the length of their arguments. The goal is to show that the probability of success of an attack against the protocol is negligible, assuming the parameters n are polynomial in η and the network messages are of length polynomial in η. In contrast, in the exact security framework, on which we focus in this course, one computes the probability of success of an attack against the protocol as a function of the probability of breaking the primitives such as $\mathsf{Succ}^{\mathsf{uf-cma}}_{\mathsf{MAC}}(t, q_m, q_v, l)$, of the runtime of functions, of the parameters n, and of the length of messages, thus providing a more precise security result. Intuitively, the probability $\mathsf{Succ}^{\mathsf{uf-cma}}_{\mathsf{MAC}}(t, q_m, q_v, l)$ is assumed to be small (otherwise, the computed probability of attack will be large), but no formal assumption on this probability is needed to establish the security theorem.

Definition 11.2 *Let T_r and T'_r be fixed-length types used for random coins; let T_k and T_e be types for keys and ciphertexts respectively. A symmetric encryption scheme* SE *[**?**] consists of three function symbols:*

- kgen : $T_r \to T_k$ *is the key generation algorithm taking as argument random coins and returning a key,*
- enc : *bitstring* $\times T_k \times T'_r \to T_e$ *is the encryption algorithm taking as arguments the cleartext, the key, and random coins, and returning the ciphertext,*
- dec : $T_e \times T_k \to$ *bitstring*$_\perp$ *is the decryption algorithm taking as arguments the ciphertext and the key, and returning either the cleartext when decryption succeeds or $\perp$ when decryption fails,*

such that $\forall m \in$ bitstring$, \forall r \in T_r, \forall r' \in T'_r$, $\mathrm{dec}(\mathrm{enc}(m, \mathrm{kgen}(r), r'), \mathrm{kgen}(r)) = m$.

Let $LR(x, y, b) = x$ if $b = 0$ and $LR(x, y, b) = y$ if $b = 1$, defined only when x and y are bitstrings of the same length. The advantage of an adversary against indistinguishability under chosen plaintext attacks (IND-CPA) is

$$\mathsf{Succ}^{\mathsf{ind-cpa}}_{\mathsf{SE}}(t, q_e, l) = \max_{\mathcal{A}} 2 \Pr\left[\begin{array}{l} b \overset{R}{\leftarrow} \{0, 1\}; r \overset{R}{\leftarrow} T_r; k \leftarrow \mathrm{kgen}(r); \\ b' \leftarrow \mathcal{A}^{r' \overset{R}{\leftarrow} T'_r; \mathrm{enc}(LR(.,.,b),k,r')} : b' = b \end{array}\right] - 1$$

where $\mathcal{A}$ is any probabilistic Turing machine that runs in time at most t and calls $r' \overset{R}{\leftarrow} T'_r$; $enc(LR(.,.,b),k,r')$ at most q_e times on messages of length at most l.

Given two bitstrings a_0 and a_1 of the same length, the left-right encryption oracle $r' \overset{R}{\leftarrow} T'_r$; $enc(LR(.,.,b),k,r')$ returns $r' \overset{R}{\leftarrow} T'_r; \mathrm{enc}(LR(a_0,a_1,b),k,r')$, that is, encrypts a_0 when $b = 0$ and a_1 when $b = 1$. $\mathsf{Succ}_{\mathsf{SE}}^{\mathsf{ind-cpa}}(t,q_e,l)$ is the probability that the adversary distinguishes the encryption of the messages a_0 given as first arguments to the left-right encryption oracle from the encryption of the messages a_1 given as second arguments. Intuitively, when the encryption scheme is IND-CPA secure, this probability is small: the ciphertext gives almost no information what the cleartext is (one cannot determine whether it is a_0 or a_1 without having the secret key). The adversary returns its guess b' for bit b, and succeeds when $b' = b$. By simply choosing a random bit b', the adversary has probability 1/2 of succeeding, so $\mathsf{Succ}_{\mathsf{SE}}^{\mathsf{ind-cpa}}(t,q_e,l)$ is 2 times the probability that $b' = b$ minus 1, so that $\mathsf{Succ}_{\mathsf{SE}}^{\mathsf{ind-cpa}}(t,q_e,l) = 0$ when the adversary just returns a random guess b' and $\mathsf{Succ}_{\mathsf{SE}}^{\mathsf{ind-cpa}}(t,q_e,l) = 1$ when the adversary always finds the correct b.

Example 11.1 *Let us consider the following trivial protocol:*

$$A \rightarrow B : e, \mathrm{mac}(e, x_{mk}) \qquad \textit{where } e = \mathrm{enc}(x'_k, x_k, x'_r) \textit{ and } x'_r, x'_k \textit{ are fresh random numbers}$$

A and B are assumed to share a key x_k for a symmetric encryption scheme and a key x_{mk} for a message authentication code. A creates a fresh key x'_k and sends it encrypted under x_k to B. A MAC is appended to the message, in order to guarantee integrity. In other words, the protocol sends x'_k encrypted using the encrypt-then-MAC scheme, in which the encryption of x under (x_k, x_{mk}) is $e, \mathrm{mac}(e, x_{mk})$ where $e = \mathrm{enc}(x, x_k, x'_r)$ and x'_r represents fresh random coins. The goal of the protocol is that x'_k should be a secret key shared between A and B. This protocol can be modeled in our calculus by the following process Q_0:

$$\begin{aligned}
Q_0 = {} & \mathsf{in}(start, ()); \mathsf{new}\ x_r : T_r; \mathsf{let}\ x_k : T_k = \mathrm{kgen}(x_r)\ \mathsf{in} \\
& \mathsf{new}\ x_{mr} : T_{mr}; \mathsf{let}\ x_{mk} : T_{mk} = \mathrm{mkgen}(x_{mr})\ \mathsf{in}\ \mathsf{out}(c, ()); (Q_A \parallel Q_B) \\
Q_A = {} & !^{i \leq n} \mathsf{in}(c_A[i], ()); \mathsf{new}\ x'_k : T_k; \mathsf{new}\ x'_r : T'_r; \\
& \mathsf{let}\ x_m : \mathit{bitstring} = \mathrm{enc}(\mathrm{k2b}(x'_k), x_k, x'_r)\ \mathsf{in}\ \mathsf{out}(c_A[i], (x_m, \mathrm{mac}(x_m, x_{mk}))) \\
Q_B = {} & !^{i' \leq n} \mathsf{in}(c_B[i'], (x'_m, x_{ma})); \mathsf{if}\ \mathrm{verify}(x'_m, x_{mk}, x_{ma})\ \mathsf{then} \\
& \mathsf{let}\ \mathrm{i}_\perp(\mathrm{k2b}(x''_k)) = \mathrm{dec}(x'_m, x_k)\ \mathsf{in}\ \mathsf{out}(c_B[i'], ())
\end{aligned}$$

When Q_0 receives a message on channel start, it begins execution: it generates the keys x_k and x_{mk} by choosing random coins x_r and $x_{r'}$ and applying the appropriate key generation algorithms. Then it yields control to the adversary, by outputting on channel c. After this output, n copies of processes for A and B are ready to be executed, when the adversary outputs on channels $c_A[i]$ or $c_B[i]$ respectively. In a session that runs as expected, the adversary first sends a message on $c_A[i]$. Then Q_A creates a fresh key x'_k (T_k is assumed to be a fixed-length type), encrypts it under x_k with random coins x'_r, computes the MAC under x_{mk} of the ciphertext, and sends the ciphertext and the MAC on $c_A[i]$. The function $\mathrm{k2b} : T_k \rightarrow$ bitstring is the natural injection $\mathrm{k2b}(x) = x$; it is needed only for type conversion. The adversary is then expected to forward this message on $c_B[i]$. When Q_B receives this message, it verifies the MAC, decrypts, and stores the obtained key in x''_k. (The function $\mathrm{i}_\perp$: bitstring $\rightarrow$ bitstring$_\perp$ is the natural injection; it is useful to check that decryption succeeded.) This key x''_k should be secret.

The adversary is responsible for forwarding messages from A to B. It can send messages in unexpected ways in order to mount an attack.

This very small example is sufficient to illustrate the main features of CryptoVerif. Section 11.6 presents results obtained on more realistic protocols.

11.2.3 Observational Equivalence

Let us now formally define game indistinguishability, which we name observational equivalence by analogy with that notion in the Dolev-Yao model. A context is a process containing a hole $[\,]$. An evaluation context C is a context built from $[\,]$, $\mathsf{newChannel}\ c; C$, $Q \parallel C$, and $C \parallel Q$. We use an evaluation context to represent the adversary. We denote by $C[Q]$ the process obtained by replacing the hole $[\,]$ in the context C with the process Q. The executed events can be used to distinguish games, so we introduce an additional algorithm, a *distinguisher* D that takes as input a sequence of events $\mathcal{E}$ and returns true or false. An example of distinguisher is D_e defined by $D_e(\mathcal{E}) = \text{true}$ if and only if $e \in \mathcal{E}$: this distinguisher detects the execution of event e. We denote the distinguisher D_e simply by e, so that $e(\mathcal{E})$ is true if and only if $e \in \mathcal{E}$. More generally, distinguishers can detect various properties of the sequence of events $\mathcal{E}$ executed by the game. We denote by $\Pr[Q : D]$ the probability that Q executes a sequence of events $\mathcal{E}$ such that $D(\mathcal{E})$ returns true.

Definition 11.3 (Observational equivalence) *Let Q and Q' be two processes and V a set of variables. Assume that Q and Q' satisfy Invariants 11.1, 11.2, and 11.3 and the variables of V are defined in Q and Q', with the same types.*

An evaluation context is said to be acceptable *for Q with public variables V if and only if $fv(C) \cap fv(Q) \subseteq V$ and $C[Q]$ satisfies Invariants 11.1, 11.2, and 11.3.*

We say that Q and Q' are observationally equivalent *up to probability p with public variables V, written $Q \approx_p^V Q'$, when for all evaluation contexts C acceptable for Q and Q' with public variables V, for all distinguishers D, $|\Pr[C[Q] : D] - \Pr[C[Q'] : D]| \leq p(C, D)$.*

This definition formalizes that algorithms C and D distinguish Q and Q' with probability at most $p(C, D)$. The probability p typically depends on the runtime of C and D, but may also depend on other parameters, such as the number of messages sent by C to each replicated process. That is why p takes as arguments C and D themselves.

The unusual requirement on variables of C comes from the presence of arrays and of the associated find construct which gives C direct access to variables of Q and Q': the context C is allowed to access variables of Q and Q' only when they are in V. (In more standard settings, the calculus does not have constructs that allow the context to access variables of Q and Q'.) When V is empty, we write $Q \approx_p Q'$ instead of $Q \approx_p^V Q'$.

The following result is not difficult to prove:

Lemma 11.1 1. *Reflexivity: $Q \approx_0^V Q$.*

2. *Symmetry: if $Q \approx_p^V Q'$, then $Q' \approx_p^V Q$.*

3. *Transitivity: if $Q \approx_p^V Q'$ and $Q' \approx_{p'}^V Q''$, then $Q \approx_{p+p'}^V Q''$.*

4. *If $Q \approx_p^V Q'$ and C is an evaluation context acceptable for Q and Q' with public variables V, then $C[Q] \approx_{p'}^{V'} C[Q']$, where $p'(C', D) = p(C'[C], D)$ and $V' \subseteq V \cup fv(C)$.*

Proofs by sequences of games consist of a sequence of observationally equivalent games $Q_0 \approx_{p_1}^V Q_1 \approx_{p_2}^V \ldots \approx_{p_n}^V Q_n$. By transitivity, $Q_0 \approx_{p_1+\ldots+p_n}^V Q_n$, so by definition of observational equivalence, $\Pr[C[Q_0] : D] \leq \Pr[C[Q_n] : D] + (p_1 + \ldots + p_n)(C, D)$.

11.3 Game Transformations

In this section, we describe the game transformations that allow us to transform the process that represents the initial protocol into a process on which the desired security property can be proved directly, by criteria given in Section 11.4. These transformations are parameterized by the set V of variables that the context can access. As we shall see in Section 11.4, V contains variables that we would like to prove secret. (The context will contain test queries that access these variables.) These transformations transform a process Q_0 into a process Q'_0 such that $Q_0 \approx^V_p Q'_0$; CryptoVerif evaluates the probability p.

11.3.1 Syntactic Transformations

RemoveAssign(x): When x is defined by an assignment $\mathsf{let}\ x[i_1, \dots, i_l] : T = M\ \mathsf{in}\ P$ and x does not occur in M (non-cyclic assignment), we replace x with its value. When x has several definitions, we simply replace $x[i_1, \dots, i_l]$ with M in P. (For accesses to x guarded by find, we do not know which definition of x is actually used.) When x has a single definition, we replace everywhere in the game $x[M_1, \dots, M_l]$ with $M\{M_1/i_1, \dots, M_l/i_l\}$. We additionally update the defined conditions of find to preserve Invariant 11.2 and to make sure that, if a condition of find guarantees that $x[M_1, \dots, M_l]$ is defined in the initial game, then so does the corresponding condition of find in the transformed game. (Essentially, when $y[M'_1, \dots, M'_{l'}]$ occurs in M, the transformation typically creates new occurrences of $y[M''_1, \dots, M''_{l'}]$ for some $M''_1, \dots, M''_{l'}$, so the condition that $y[M''_1, \dots, M''_{l'}]$ is defined must sometimes be explicitly added to conditions of find in order to preserve Invariant 11.2.) When $x \in V$, its definition is kept unchanged. Otherwise, when x is not referred to at all after the transformation, we remove the definition of x. When x is referred to only at the root of defined tests, we replace its definition with a constant. (The definition point of x is important, but not its value.)

Example 11.2 *In the process of Example 11.1, the transformation* ***RemoveAssign***(x_{mk}) *substitutes* $\mathrm{mkgen}(x_{mr})$ *for* x_{mk} *in the whole process and removes the assignment* $\mathsf{let}\ x_{mk} : T_{mk} = \mathrm{mkgen}(x_{mr})$*. After this substitution,* $\mathrm{mac}(x_m, x_{mk})$ *becomes* $\mathrm{mac}(x_m, \mathrm{mkgen}(x_{mr}))$ *and* $\mathrm{verify}(x'_m, x_{mk}, x_{ma})$ *becomes* $\mathrm{verify}(x'_m, \mathrm{mkgen}(x_{mr}), x_{ma})$*, thus exhibiting terms required in Section 11.3.2. The situation is similar for* ***RemoveAssign***(x_k)*.*

SArename(x): The transformation **SArename** (single assignment rename) aims at renaming variables so that each variable has a single definition in the game; this is useful for distinguishing cases depending on which definition of x has set $x[\widetilde{i}]$. This transformation can be applied only when $x \notin V$. When x has $m > 1$ definitions, we rename each definition of x to a different variable $x_1, \dots, x_m$. Terms $x[\widetilde{i}]$ under a definition of $x_j[\widetilde{i}]$ are then replaced with $x_j[\widetilde{i}]$. Each branch of find $FB = \widetilde{u}[\widetilde{i}] \leq \widetilde{n}\ \mathsf{suchthat}\ \mathsf{defined}(M_1, \dots, M_l) \wedge M\ \mathsf{then}\ P$ where $x[\widetilde{M}]$ is a subterm of some M_k for $k \leq l$ is replaced with m branches $FB\{x_j[\widetilde{M}]/x[\widetilde{M}]\}$ for $1 \leq j \leq m$.

Example 11.3 *Consider the following process*

$$\begin{aligned}
&\mathsf{in}(start, ()); \mathsf{new}\ r_A : T_r; \mathsf{let}\ k_A : T_k = \mathrm{kgen}(r_A)\ \mathsf{in}\\
&\quad \mathsf{new}\ r_B : T_r; \mathsf{let}\ k_B : T_k = \mathrm{kgen}(r_B)\ \mathsf{in}\ \mathsf{out}(yield, ()); (Q_K \parallel Q_S)\\
&Q_K = !^{i \leq n} \mathsf{in}(c[i], (h : T_h, k : T_k))\\
&\qquad \mathsf{if}\ h = A\ \mathsf{then}\ \mathsf{let}\ k' : T_k = k_A\ \mathsf{in}\ \mathsf{out}(yield, ())\ \mathsf{else}\\
&\qquad \mathsf{if}\ h = B\ \mathsf{then}\ \mathsf{let}\ k' : T_k = k_B\ \mathsf{in}\ \mathsf{out}(yield, ())\ \mathsf{else}\\
&\qquad \mathsf{let}\ k' : T_k = k\ \mathsf{in}\ \mathsf{out}(yield, ())\\
&Q_S = !^{i' \leq n'} \mathsf{in}(c'[i'], (h' : T_h)); \mathsf{find}\ u \leq n\ \mathsf{suchthat}\\
&\qquad \mathsf{defined}(h[u], k'[u]) \wedge h' = h[u]\ \mathsf{then}\ P_1(k'[u])\ \mathsf{else}\ P_2
\end{aligned}$$

The process Q_K *stores in* (h, k') *a table of pairs (host name, key): the key for* A *is* k_A*, for* B*,* k_B*, and for any other* h*, the adversary can choose the key* k*. The process* Q_S *queries this table of keys to find the key* $k'[u]$ *of host* h'*, then executes* $P_1(k'[u])$*. If* h' *is not found, it executes* P_2*.*

By the transformation ***SArename***(k')*, we can perform a case analysis, to distinguish the cases in which* $k' = k_A$*,* $k' = k_B$*, or* $k' = k$*. After transformation, we obtain the following processes:*

$$
\begin{aligned}
Q'_K = {} & !^{i \le n}\mathsf{in}(c[i], (h : T_h, k : T_k)) \\
& \quad \mathsf{if}\ h = A\ \mathsf{then\ let}\ k'_1 : T_k = k_A\ \mathsf{in\ out}(yield, ())\ \mathsf{else} \\
& \quad \mathsf{if}\ h = B\ \mathsf{then\ let}\ k'_2 : T_k = k_B\ \mathsf{in\ out}(yield, ())\ \mathsf{else} \\
& \quad \mathsf{let}\ k'_3 : T_k = k\ \mathsf{in\ out}(yield, ()) \\
Q'_S = {} & !^{i' \le n'}\mathsf{in}(c'[i'], h' : T_h); \\
& \quad \mathsf{find}\ u \le n\ \mathsf{suchthat\ defined}(h[u], k'_1[u]) \wedge h' = h[u]\ \mathsf{then}\ P_1(k'_1[u]) \\
& \quad\ \ \oplus\ u \le n\ \mathsf{suchthat\ defined}(h[u], k'_2[u]) \wedge h' = h[u]\ \mathsf{then}\ P_1(k'_2[u]) \\
& \quad\ \ \oplus\ u \le n\ \mathsf{suchthat\ defined}(h[u], k'_3[u]) \wedge h' = h[u]\ \mathsf{then}\ P_1(k'_3[u])\ \mathsf{else}\ P_2
\end{aligned}
$$

After the simplification (sketched below), Q'_S *becomes:*

$$
\begin{aligned}
Q''_S = {} & !^{i' \le n'}\mathsf{in}(c'[i'], h' : T_h); \\
& \quad \mathsf{find}\ u \le n\ \mathsf{suchthat\ defined}(h[u], k'_1[u]) \wedge h' = A\ \mathsf{then}\ P_1(k_A) \\
& \quad\ \ \oplus\ u \le n\ \mathsf{suchthat\ defined}(h[u], k'_2[u]) \wedge h' = B\ \mathsf{then}\ P_1(k_B) \\
& \quad\ \ \oplus\ u \le n\ \mathsf{suchthat\ defined}(h[u], k'_3[u]) \wedge h' = h[u]\ \mathsf{then}\ P_1(k[u])\ \mathsf{else}\ P_2
\end{aligned}
$$

since, when $k'_1[u]$ *is defined,* $k'_1[u] = k_A$ *and* $h[u] = A$*, and similarly for* $k'_2[u]$ *and* $k'_3[u]$*.*

Simplify: The prover uses a simplification algorithm, based on an equational prover, using an algorithm similar to the Knuth-Bendix completion [**?**]. This equational prover uses:

- User-defined equations, of the form $\forall x_1 : T_1, \dots, \forall x_m : T_m, M$ which mean that for all environments E, if for all $j \le m$, $E(x_j) \in T_j$, then $E, M \Downarrow \text{true}$. For example, considering MAC and encryption schemes as in Definitions 11.1 and 11.2 respectively, we have:

  $$\forall r : T_{mr}, \forall m : \mathit{bitstring}, \mathrm{verify}(m, \mathrm{mkgen}(r), \mathrm{mac}(m, \mathrm{mkgen}(r))) = \mathrm{true} \qquad \text{(mac)}$$
  $$\forall m : \mathit{bitstring}; \forall r : T_r, \forall r' : T'_r, \mathrm{dec}(\mathrm{enc}(m, \mathrm{kgen}(r), r'), \mathrm{kgen}(r)) = \mathrm{i}_\perp(m) \qquad \text{(enc)}$$

  We express the poly-injectivity of the function k2b of Example 11.1 by

  $$\begin{aligned} &\forall x : T_k, \forall y : T_k, (\mathrm{k2b}(x) = \mathrm{k2b}(y)) = (x = y) \\ &\forall x : T_k, \mathrm{k2b}^{-1}(\mathrm{k2b}(x)) = x \end{aligned} \qquad \text{(k2b)}$$

  where $\mathrm{k2b}^{-1}$ is a function symbol that denotes the inverse of k2b. We have similar formulas for $\mathrm{i}_\perp$.

- Equations that come from the process. For example, in the process $\mathsf{if}\ M\ \mathsf{then}\ P\ \mathsf{else}\ P'$, we have $M = \text{true}$ in P and $M = \text{false}$ in P'.

- The low probability of collision between random values. For example, when x is defined by $\mathsf{new}\ x : T$ under replications bounded by $n_1, \dots, n_m$, $x[M_1, \dots, M_m] = x[M'_1, \dots, M'_m]$ implies $M_1 = M'_1, \dots, M_m = M'_m$ up to probability $p = \frac{(n_1 \dots n_m)^2}{2|T|}$. Indeed, the array x consists of $n_1 \dots n_m$ independent uniformly distributed random numbers and, if $x[M_1, \dots, M_m] = x[M'_1, \dots, M'_m]$ and $(M_1, \dots, M_m) \neq (M'_1, \dots, M'_m)$, then the array x contains two equal cells. There are $(n_1 \dots n_m)^2/2$ sets of two cells of x, and the probability that two independent uniformly distributed random elements of T are equal is $1/|T|$, which yields probability p. The replacement of $x[M_1, \dots, M_m] = x[M'_1, \dots, M'_m]$ with

$M_1 = M'_1, \ldots, M_m = M'_m$ is performed when the type T is *large*, which means that $|T|$ is large enough so that the probability p can be considered small.

Similarly, when 1) x is defined by $\mathsf{new}\ x : T$ and T is a large type, 2) for each value of M_1, there is at most one value of x (or of a part of x of a large type) that can yield that value of M_1, and 3) M_2 does not depend on x, then $M_1 \neq M_2$ up to a small probability. The fact that M_2 does not depend on x is proved using a dependency analysis.

The prover combines these properties to simplify terms, and uses simplified forms of terms to simplify processes. For example, if M simplifies to true, then $\mathsf{if}\ M\ \mathsf{then}\ P\ \mathsf{else}\ P'$ simplifies to P. Similarly, a branch of find is removed when the associated condition simplifies to false.

Details on the simplification procedure can be found in [**?**, Appendix C]. The asymptotic version of the following proposition is proved in [**?**, Appendix E.1].

Proposition 11.1 *Let Q_0 be a process that satisfies Invariants 11.1, 11.2, and 11.3 and Q'_0 the process obtained from Q_0 by one of the transformations above. Then Q'_0 satisfies Invariants 11.1, 11.2, and 11.3, and $Q_0 \approx^V_p Q'_0$, where $p = 0$ for the transformations **RemoveAssign** and **SArename**, and p is the probability of eliminated collisions for **Simplify**.*

11.3.2 Applying the Security Assumptions on Primitives

The security of cryptographic primitives is defined using observational equivalences given as axioms. Importantly, this formalism allows us to specify many different primitives in a generic way. Such equivalences are then used by the prover in order to transform a game into another, observationally equivalent game, as explained below.

The primitives are specified using equivalences of the form $(G_1, \ldots, G_m) \approx_p (G'_1, \ldots, G'_m)$ where G is defined by the following grammar, with $l \geq 0$ and $m \geq 1$:

$$\begin{array}{ll}
G ::= & \text{group of oracles} \\
\quad !^{i \leq n}\mathsf{new}\ y_1 : T_1; \ldots; \mathsf{new}\ y_l : T_l; (G_1, \ldots, G_m) & \quad \text{replication, restrictions} \\
\quad O(x_1 : T_1, \ldots, x_l : T_l) := \mathit{OP} & \quad \text{oracle} \\
 & \\
\mathit{OP} ::= & \text{oracle processes} \\
\quad M & \quad \text{term} \\
\quad \mathsf{new}\ x[\widetilde{i}] : T; \mathit{OP} & \quad \text{random number} \\
\quad \mathsf{let}\ x[\widetilde{i}] : T = M\ \mathsf{in}\ \mathit{OP} & \quad \text{assignment} \\
\quad \mathsf{find}\ (\bigoplus_{j=1}^m \widetilde{u_j}[\widetilde{i}] \leq \widetilde{n_j}\ \mathsf{suchthat} & \\
\qquad \mathsf{defined}(M_{j1}, \ldots, M_{jl_j}) \wedge M_j\ \mathsf{then}\ \mathit{OP}_j)\ \mathsf{else}\ \mathit{OP} & \quad \text{array lookup}
\end{array}$$

Intuitively, $O(x_1 : T_1, \ldots, x_l : T_l) := \mathit{OP}$ represents an oracle O that takes as argument values $x_1, \ldots, x_l$ of types $T_1, \ldots, T_l$ respectively and returns a result computed by OP. The observational equivalence $(G_1, \ldots, G_m) \approx_p (G'_1, \ldots, G'_m)$ expresses that the adversary has probability at most p of distinguishing oracles in the left-hand side from corresponding oracles in the right-hand side. Formally, oracles can be encoded as processes that input their arguments and output their result on a channel, as shown in Figure 11.2: $[\![\mathit{OP}]\!]^{\widetilde{j}}_{\widetilde{i}}$ denotes the translation of the oracle process OP into an output process; $[\![G]\!]^{\widetilde{j}}_{\widetilde{i}}$ denotes the translation of the group of oracles G into an input process. The translation of $!^{i \leq n}\mathsf{new}\ y_1 : T_1; \ldots; \mathsf{new}\ y_l : T_l; (G_1, \ldots, G_m)$ inputs and outputs on channel $c_{\widetilde{j}}$ so that the context can trigger the generation of random numbers $y_1, \ldots, y_l$. The translation of $O(x_1 : T_1, \ldots, x_l : T_l) := \mathit{OP}$ inputs the arguments of the oracle on channel $c_{\widetilde{j}}$ and translates OP, which outputs the result of OP on $c_{\widetilde{j}}$. (In the left-hand side of equivalences, the result OP of oracles must simply be a term M.) The observational equivalence $(G_1, \ldots, G_m) \approx_p (G'_1, \ldots, G'_m)$ is then an abbreviation for $[\![(G_1, \ldots, G_m)]\!] \approx_p [\![(G'_1, \ldots, G'_m)]\!]$.

$$
\begin{aligned}
&\llbracket (G_1, \dots, G_m) \rrbracket = \llbracket G_1 \rrbracket^1 \parallel \dots \parallel \llbracket G_m \rrbracket^m \\
&\llbracket !^{i \le n} \mathsf{new}\ y_1 : T_1; \dots ; \mathsf{new}\ y_l : T_l ; (G_1, \dots, G_m) \rrbracket^{\widetilde{j}}_{\widetilde{i}} = \\
&\qquad !^{i \le n} \mathsf{in}(c_{\widetilde{j}}[\widetilde{i}, i], ()); \mathsf{new}\ y_1 : T_1; \dots ; \mathsf{new}\ y_l : T_l; \mathsf{out}(c_{\widetilde{j}}[\widetilde{i}, i], ()); (\llbracket G_1 \rrbracket^{\widetilde{j},1}_{\widetilde{i},i} \parallel \dots \parallel \llbracket G_m \rrbracket^{\widetilde{j},m}_{\widetilde{i},i}) \\
&\llbracket O(x_1 : T_1, \dots, x_l : T_l) := OP \rrbracket^{\widetilde{j}}_{\widetilde{i}} = \mathsf{in}(c_{\widetilde{j}}[\widetilde{i}], (x_1 : T_1, \dots, x_l : T_l)); \llbracket OP \rrbracket^{\widetilde{j}}_{\widetilde{i}} \\
&\llbracket M \rrbracket^{\widetilde{j}}_{\widetilde{i}} = \mathsf{out}(c_{\widetilde{j}}[\widetilde{i}], M) \\
&\llbracket \mathsf{new}\ x[\widetilde{i}] : T; OP \rrbracket^{\widetilde{j}}_{\widetilde{i}} = \mathsf{new}\ x[\widetilde{i}] : T; \llbracket OP \rrbracket^{\widetilde{j}}_{\widetilde{i}} \\
&\llbracket \mathsf{let}\ x[\widetilde{i}] : T = M\ \mathsf{in}\ OP \rrbracket^{\widetilde{j}}_{\widetilde{i}} = \mathsf{let}\ x[\widetilde{i}] : T = M\ \mathsf{in}\ \llbracket OP \rrbracket^{\widetilde{j}}_{\widetilde{i}} \\
&\llbracket \mathsf{find}\ (\textstyle\bigoplus_{j=1}^m \widetilde{u_j}[\widetilde{i}] \le \widetilde{n_j}\ \mathsf{suchthat}\ \mathsf{defined}(M_{j1}, \dots, M_{jl_j}) \wedge M_j\ \mathsf{then}\ OP_j)\ \mathsf{else}\ OP \rrbracket^{\widetilde{j}}_{\widetilde{i}} = \\
&\quad \mathsf{find}\ (\textstyle\bigoplus_{j=1}^m \widetilde{u_j}[\widetilde{i}] \le \widetilde{n_j}\ \mathsf{suchthat}\ \mathsf{defined}(M_{j1}, \dots, M_{jl_j}) \wedge M_j\ \mathsf{then}\ \llbracket OP_j \rrbracket^{\widetilde{j}}_{\widetilde{i}})\ \mathsf{else}\ \llbracket OP \rrbracket^{\widetilde{j}}_{\widetilde{i}}
\end{aligned}
$$

where $c_{\widetilde{j}}$ are pairwise distinct channels, $\widetilde{i} = i_1, \dots, i_{l'}$, and $\widetilde{j} = j_0, \dots, j_{l'}$.

Figure 11.2: Translation from oracle processes to processes

For example, the security of a MAC (Definition 11.1) is represented by the equivalence $L \approx_{p_{\mathrm{mac}}} R$ where:

$$
\begin{aligned}
L = {} & !^{i'' \le n''} \mathsf{new}\ r : T_{mr}; (\\
& \quad !^{i \le n} \mathit{Omac}(x : \mathit{bitstring}) := \mathrm{mac}(x, \mathrm{mkgen}(r)), \\
& \quad !^{i' \le n'} \mathit{Overify}(m : \mathit{bitstring}, \mathit{ma} : T_{ms}) := \mathrm{verify}(m, \mathrm{mkgen}(r), \mathit{ma})) \\
R = {} & !^{i'' \le n''} \mathsf{new}\ r : T_{mr}; (\\
& \quad !^{i \le n} \mathit{Omac}(x : \mathit{bitstring}) := \mathrm{mac}'(x, \mathrm{mkgen}'(r)), \\
& \quad !^{i' \le n'} \mathit{Overify}(m : \mathit{bitstring}, \mathit{ma} : T_{ms}) := \\
& \qquad \mathsf{find}\ u \le n\ \mathsf{suchthat}\ \mathsf{defined}(x[u]) \wedge (m = x[u]) \\
& \qquad\quad \wedge\ \mathrm{verify}'(m, \mathrm{mkgen}'(r), \mathit{ma})\ \mathsf{then}\ \mathrm{true}\ \mathsf{else}\ \mathrm{false}) \\
p_{\mathrm{mac}}(C, D) = {} & n''\, \mathsf{Succ}^{\mathsf{uf-cma}}_{\mathsf{MAC}}(t_C + (n'' - 1)(\mathrm{time}(\mathrm{mkgen}) + n\, \mathrm{time}(\mathrm{mac}, \mathrm{maxl}(x)) \\
& \quad + n'\, \mathrm{time}(\mathrm{verify}, \mathrm{maxl}(m)), n, n', \max(\mathrm{maxl}(x), \mathrm{maxl}(m)))
\end{aligned}
\qquad (\mathrm{mac}_{\mathrm{eq}})
$$

where mac′, verify′, and mkgen′ are function symbols with the same types as mac, verify, and mkgen respectively. (We use different function symbols on the left- and right-hand sides, just to prevent a repeated application of the transformation induced by this equivalence. Since we add these function symbols, we also add the equation

$$\forall r : T_{mr}, \forall m : \mathit{bitstring}, \mathrm{verify}'(m, \mathrm{mkgen}'(r), \mathrm{mac}'(m, \mathrm{mkgen}'(r))) = \mathrm{true} \qquad (\mathrm{mac}')$$

which restates (mac) for mac′, verify′, and mkgen′.) Intuitively, the equivalence $L \approx_{p_{\mathrm{mac}}} R$ leaves MAC computations unchanged (except for the use of primed function symbols in R), and allows one to replace a MAC verification $\mathrm{verify}(m, \mathrm{mkgen}(r), \mathit{ma})$ with a lookup in the array x of messages whose mac has been computed with key $\mathrm{mkgen}(r)$: if m is found in the array x and $\mathrm{verify}(m, \mathrm{mkgen}(r), \mathit{ma})$, we return true; otherwise, the verification fails (up to negligible probability), so we return false. (If the verification succeeds with m not in the array x, then the adversary has forged a MAC.) Obviously, the form of L requires that r is used only to compute or verify MACs, for the equivalence to be correct. In the probability $p_{\mathrm{mac}}(C, D)$, t_C is the runtime of context C; n'' is the maximum number of considered MAC keys; n' and n'' are respectively the maximum number of calls to *Omac* and *Overify* for each MAC key (n, n', n'' are in fact functions of C); $\mathrm{time}(f, l_1, \ldots, l_k)$ is the maximum runtime of f, called with arguments of length at most $l_1, \ldots, l_k$ (the lengths $l_1, \ldots, l_k$ are omitted when the type of the argument already bounds its length); $\mathrm{maxl}(x)$ is the maximum length of x. Formally, the following result shows the correctness of our modeling. It is a fairly easy consequence of Definition 11.1, and its asymptotic version is proved in [**?**, Appendix E.3].

Proposition 11.2 *If* $(\mathrm{mkgen}, \mathrm{mac}, \mathrm{verify})$ *is a UF-CMA message authentication code, and the symbols* mkgen', mac', *and* verify' *represent the same functions as* mkgen, mac, *and* verify *respectively, then* $[\![L]\!] \approx_{p_{\mathrm{mac}}} [\![R]\!]$.

Similarly, if $(\mathrm{kgen}, \mathrm{enc}, \mathrm{dec})$ is an IND-CPA symmetric encryption scheme (Definition 11.2), then we have the following equivalence:

$$\begin{aligned}&!^{i'\leq n'}\mathsf{new}\ r : T_r; !^{i\leq n}\mathit{Oenc}(x : \mathit{bitstring}) := \mathsf{new}\ r' : T'_r; \mathrm{enc}(x, \mathrm{kgen}(r), r')\\ \approx_{p_{\mathrm{enc}}} &!^{i'\leq n'}\mathsf{new}\ r : T_r; !^{i\leq n}\mathit{Oenc}(x : \mathit{bitstring}) := \mathsf{new}\ r' : T'_r; \mathrm{enc}'(\mathrm{Z}(x), \mathrm{kgen}'(r), r')\end{aligned} \qquad (\mathrm{enc}_{\mathrm{eq}})$$

where $p_{\mathrm{enc}}(C, D) = n' \,\mathsf{Succ}^{\mathsf{ind-cpa}}_{\mathsf{SE}}(t_C + t_D + (n'-1)(\mathrm{time}(\mathrm{kgen}) + n\,\mathrm{time}(\mathrm{enc}, \mathrm{maxl}(x)) + n\,\mathrm{time}(Z, \mathrm{maxl}(x))), n, \mathrm{maxl}(x))$, enc' and kgen' are function symbols with the same types as enc and kgen respectively, and $\mathrm{Z} : \mathit{bitstring} \rightarrow \mathit{bitstring}$ is the function that returns a bitstring of the same length as its argument, consisting only of zeroes. Using equations such as $\forall x : T, \mathrm{Z}(\mathrm{T2b}(x)) = \mathrm{Z}_T$, we can prove that $\mathrm{Z}(\mathrm{T2b}(x))$ does not depend on x when x is of a fixed-length type T and $\mathrm{T2b} : T \rightarrow \mathit{bitstring}$ is the natural injection.

Exercice 42
The advantage of the adversary against *strong unforgeability under chosen message attacks (SUF-CMA)* of MACs is:

$$\begin{aligned}&\mathsf{Succ}^{\mathsf{suf-cma}}_{\mathsf{MAC}}(t, q_m, q_v, l) =\\ &\max_{\mathcal{A}} \Pr\left[\begin{array}{l} k \xleftarrow{R} \mathrm{mkgen}; (m, s) \leftarrow \mathcal{A}^{\mathrm{mac}(.,k),\mathrm{verify}(.,k,.)} : \mathrm{verify}(m, k, s) \wedge \\ s \text{ is not the result of calling the oracle } \mathrm{mac}(., k) \text{ on } m \end{array}\right]\end{aligned}$$

where $\mathcal{A}$ runs in time at most t, calls $\mathrm{mac}(., k)$ at most q_m times with messages of length at most l, calls $\mathrm{verify}(., k, .)$ at most q_v times with messages of length at most l. The difference with UF-CMA MACs is that, for SUF-CMA MACs, generating a different MAC for a message whose MAC has already been computed is considered as a forgery.

Represent SUF-CMA MACs in the CryptoVerif formalism.

Exercice 43
A *signature scheme* S consists of

- a key generation algorithm $(pk, sk) \xleftarrow{R} \mathrm{kgen}$
- a signature algorithm $\mathrm{sign}(m, sk)$
- a verification algorithm $\mathrm{verify}(m, pk, s)$

such that $\mathrm{verify}(m, pk, \mathrm{sign}(m, sk)) = 1$.

The advantage of the adversary against *unforgeability under chosen message attacks (UF-CMA)* of signatures is:

$$\begin{aligned}&\mathsf{Succ}^{\mathsf{uf-cma}}_{\mathsf{S}}(t, q_s, l) =\\ &\max_{\mathcal{A}} \Pr\left[\begin{array}{l} (pk, sk) \xleftarrow{R} \mathrm{kgen}; (m, s) \leftarrow \mathcal{A}^{\mathrm{sign}(.,sk)}(pk) : \mathrm{verify}(m, pk, s) \wedge \\ m \text{ was never queried to the oracle } \mathrm{sign}(., sk) \end{array}\right]\end{aligned}$$

where $\mathcal{A}$ runs in time at most t, calls $\mathrm{sign}(., sk)$ at most q_s times with messages of length at most l. Signatures are similar to MACs, except that one uses a key pair instead of a single key: the signature is computed using the secret key sk and verified using the public key pk.

Represent UF-CMA signatures in the CryptoVerif formalism.

Exercice 44

The advantage of the adversary against *ciphertext integrity (INT-CTXT)* of a symmetric encryption scheme SE is:

$$\begin{aligned}&\mathsf{Succ}_{\mathsf{SE}}^{\mathsf{int-ctxt}}(t, q_e, q_d, l_e, l_d) =\\ &\quad\max_{\mathcal{A}} \Pr\left[\begin{array}{l} k \overset{R}{\leftarrow} \mathrm{kgen}; c \leftarrow \mathcal{A}^{\mathrm{enc}(.,k),\mathrm{dec}(.,k)\neq\perp} : \mathrm{dec}(c,k) \neq \perp \wedge \\ c \text{ is not the result of a call to the } \mathrm{enc}(.,k) \text{ oracle}\end{array}\right]\end{aligned}$$

where $\mathcal{A}$ runs in time at most t, calls $\mathrm{enc}(., k)$ at most q_e times with messages of length at most l_e, calls $\mathrm{dec}(., k) \neq \perp$ at most q_d times with messages of length at most l_d. Intuitively, ciphertext integrity means that the adversary is unable to compute a ciphertext that correctly decrypts, except by calling the encryption oracle.

Represent INT-CTXT encryption in the CryptoVerif formalism.

Exercice 45

A *public-key encryption scheme* AE consists of

- a key generation algorithm $(pk, sk) \overset{R}{\leftarrow} \mathrm{kgen}$
- a probabilistic encryption algorithm $\mathrm{enc}(m, pk)$
- a decryption algorithm $\mathrm{dec}(m, sk)$

such that $\mathrm{dec}(\mathrm{enc}(m, pk), sk) = m$.

The advantage of the adversary against *indistinguishability under adaptive chosen-ciphertext attacks (IND-CCA2)* is

$$\begin{aligned}&\mathsf{Succ}_{\mathsf{AE}}^{\mathsf{ind-cca2}}(t, q_d) =\\ &\quad\max_{\mathcal{A}}\ 2\Pr\left[\begin{array}{l} b \overset{R}{\leftarrow} \{0,1\}; (pk, sk) \overset{R}{\leftarrow} \mathrm{kgen};\\ (m_0, m_1, s) \leftarrow \mathcal{A}_1^{\mathrm{dec}(.,sk)}(pk); y \leftarrow \mathrm{enc}(m_b, pk);\\ b' \leftarrow \mathcal{A}_2^{\mathrm{dec}(.,sk)}(m_0, m_1, s, y) : b' = b \wedge\\ \mathcal{A}_2 \text{ has not called } \mathrm{dec}(., sk) \text{ on } y\end{array}\right] - 1\end{aligned}$$

where $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ runs in time at most t and calls $\mathrm{dec}(., sk)$ at most q_d times. The adversary first chooses two messages m_0 and m_1; y is the encryption of one of them m_b under pk, and the adversary should be unable to guess which one was encrypted from the ciphertext y. The adversary has access to the public key pk (so that it can encrypt) and to the decryption oracle $\mathrm{dec}(., sk)$, but obviously it must not call the decryption oracle on the ciphertext y, otherwise it could always find which message was encrypted. The variable s contains some state that is passed from the first part of the adversary $\mathcal{A}_1$ to the second part $\mathcal{A}_2$.

Represent IND-CCA2 encryption in the CryptoVerif formalism.

The representation of these and other primitives can be found in [**?**, Appendix D.3]. The equivalences that formalize the security assumptions on primitives are designed and proved correct by hand from security assumptions in a more standard form, as in the MAC example. Importantly, these manual proofs are done only once for each primitive, and the obtained equivalence can be reused for proving many different protocols automatically.

Assuming $L \approx_p R$, Lemma 11.1 yields $C[\llbracket L \rrbracket] \approx^V_{p'} C[\llbracket R \rrbracket]$ with $p'(C', D) = p(C'[C], D)$, for all evaluation contexts C acceptable for $\llbracket L \rrbracket$ and $\llbracket R \rrbracket$ with no public variables, so we can transform a process Q_0 such that $Q_0 \approx^V_0 C[\llbracket L \rrbracket]$ into a process Q'_0 such that $Q_0 \approx^V_0 C[\llbracket L \rrbracket] \approx^V_{p'} C[\llbracket R \rrbracket] \approx^V_0 Q'_0$. In order to check that $Q_0 \approx^V_0 C[\llbracket L \rrbracket]$, the prover uses sufficient conditions, which essentially guarantee that all uses of certain secret variables of Q_0, in a set S, can be implemented by calling oracles of L. Let $\mathcal{M}$ be a set of occurrences of terms, corresponding to uses of variables of S. Informally, the prover shows the following properties.

- For each $M \in \mathcal{M}$, there exist a term N_M, which is the result of an oracle of L, and a substitution σ_M such that $M = \sigma_M N_M$. (Precisely, σ_M applies to the abbreviated form of N_M in which we write x instead of $x[\widetilde{i}]$.) Intuitively, the evaluation of M in Q_0 will correspond to a call to the oracle with result N_M in $C[\llbracket L \rrbracket]$.

- The variables of S do not occur in V, are bound by restrictions in Q_0, and occur only in terms $M = \sigma_M N_M \in \mathcal{M}$ in Q_0, at occurrences that are images by σ_M of variables bound by restrictions in L. (To be precise, the variables of S are also allowed to occur at the root of defined conditions; in that case, their value does not matter, just the fact that they are defined.)

- Let $\widetilde{i}$ and $\widetilde{i'}$ be the sequences of current replication indices at N_M in L and at M in Q_0, respectively. The prover shows that there exists a function mapIdx_M that maps the array indices at M in Q_0 to the array indices at N_M in L: the evaluation of M when $\widetilde{i'} = \widetilde{a}$ will correspond in $C[\llbracket L \rrbracket]$ to the evaluation of N_M when $\widetilde{i} = \mathrm{mapIdx}_M(\widetilde{a})$. Thus, σ_M and mapIdx_M induce a correspondence between terms and variables of Q_0 and variables of L: for all $M \in \mathcal{M}$, for all $x[\widetilde{i''}]$ that occur in N_M, $(\sigma_M x)\{\widetilde{a}/\widetilde{i'}\}$ corresponds to $x[\widetilde{i''}]\{\mathrm{mapIdx}_M(\widetilde{a})/\widetilde{i}\}$, that is, $(\sigma_M x)\{\widetilde{a}/\widetilde{i'}\}$ in a trace of Q_0 has the same value as $x[\widetilde{i''}]\{\mathrm{mapIdx}_M(\widetilde{a})/\widetilde{i}\}$ in the corresponding trace of $C[\llbracket L \rrbracket]$ ($\widetilde{i''}$ is a prefix of $\widetilde{i}$). We detail below conditions that this correspondence has to satisfy.

For example, consider a process Q_0 that contains $M_1 = \mathrm{enc}(M'_1, \mathrm{kgen}(x_r), x'_r[i_1])$ under a replication $!^{i_1 \leq n_1}$ and $M_2 = \mathrm{enc}(M'_2, \mathrm{kgen}(x_r), x''_r[i_2])$ under a replication $!^{i_2 \leq n_2}$, where x_r, x'_r, x''_r are bound by restrictions. Let $S = \{x_r, x'_r, x''_r\}$, $\mathcal{M} = \{M_1, M_2\}$, and $N_{M_1} = N_{M_2} = \mathrm{enc}(x[i', i], \mathrm{kgen}(r[i']), r'[i', i])$. The functions mapIdx_{M_1} and mapIdx_{M_2} are defined by

$$\begin{aligned}
\mathrm{mapIdx}_{M_1}(a_1) &= (1, a_1) \text{ for } a_1 \in [1, n_1] \\
\mathrm{mapIdx}_{M_2}(a_2) &= (1, a_2 + n_1) \text{ for } a_2 \in [1, n_2]
\end{aligned}$$

Then $M'_1\{a_1/i_1\}$ corresponds to $x[1, a_1]$, x_r to $r[1]$, $x'_r[a_1]$ to $r'[1, a_1]$, $M'_2\{a_2/i_2\}$ to $x[1, a_2 + n_1]$, and $x''_r[a_2]$ to $r'[1, a_2 + n_1]$. The functions mapIdx_{M_1} and mapIdx_{M_2} are such that $x'_r[a_1]$ and $x''_r[a_2]$ never correspond to the same cell of r'; indeed, $x'_r[a_1]$ and $x''_r[a_2]$ are independent random numbers in Q_0, so their images in $C[\llbracket L \rrbracket]$ must also be independent random numbers.

The above correspondence must satisfy the following soundness conditions:

- when x is an oracle argument in L, the term that corresponds to $x[\widetilde{a'}]$ must have the same type as $x[\widetilde{a'}]$, and when two terms correspond to the same $x[\widetilde{a'}]$, they must evaluate to the same value;

- when x is bound by $\mathsf{new}\ x : T$ in L, the term that corresponds to $x[\widetilde{a'}]$ must evaluate to $z[\widetilde{a''}]$ where $z \in S$ and z is bound by $\mathsf{new}\ z : T$ in Q_0, and the relation that associates $z[\widetilde{a''}]$ to $x[\widetilde{a'}]$ is an injective function (so that independent random numbers in L correspond to independent random numbers in Q_0).

It is easy to check that, in the previous example, these conditions are satisfied.

The transformation of Q_0 into Q'_0 consists in two steps. First, we replace the restrictions that define variables of S with restrictions that define fresh variables corresponding to variables bound by new in R. The correspondence between variables of Q_0 and variables of $C[\llbracket L\rrbracket]$ is extended to include these fresh variables. Second, we reorganize Q_0 so that each evaluation of a term $M \in \mathcal{M}$ first stores the values of the arguments $x_1, \dots, x_m$ of the oracle $O(x_1 : T_1, \dots, x_m : T_m) := N_M$ in fresh variables, then computes N_M and stores its result in a fresh variable, and uses this variable instead of M; then we simply replace the computation of N_M with the corresponding oracle process of R, taking into account the correspondence of variables.

The full formal description of this transformation is given in [**?**, Appendix D.1]. The following proposition shows the soundness of the transformation; its asymptotic version is proved in [**?**, Appendix E.4].

Proposition 11.3 *Let Q_0 be a process that satisfies Invariants 11.1, 11.2, and 11.3 and Q_0' the process obtained from Q_0 by the above transformation. Then Q_0' satisfies Invariants 11.1, 11.2, and 11.3 and, if $\llbracket L\rrbracket \approx_p \llbracket R\rrbracket$, then $Q_0 \approx^V_{p'} Q_0'$ where $p'(C', D) = p(C'[C], D)$ and C is an evaluation context such that $Q_0 \approx^V_0 C[\llbracket L\rrbracket] \approx^V_{p'} C[\llbracket R\rrbracket] \approx^V_0 Q_0'$.*

Example 11.4 *In order to treat Example 11.1, the prover is given as input the indication that T_{mr}, T_r, T_r', and T_k are fixed-length types; the type declarations for the functions* $\mathrm{mkgen}, \mathrm{mkgen}' : T_{mr} \to T_{mk}$, $\mathrm{mac}, \mathrm{mac}' : \mathit{bitstring} \times T_{mk} \to T_{ms}$, $\mathrm{verify}, \mathrm{verify}' : \mathit{bitstring} \times T_{mk} \times T_{ms} \to \mathit{bool}$, $\mathrm{kgen}, \mathrm{kgen}' : T_r \to T_k$, $\mathrm{enc}, \mathrm{enc}' : \mathit{bitstring} \times T_k \times T_r' \to T_e$, $\mathrm{dec} : T_e \times T_k \to \mathit{bitstring}_\perp$, $\mathrm{k2b} : T_k \to \mathit{bitstring}$, $\mathrm{i}_\perp : \mathit{bitstring} \to \mathit{bitstring}_\perp$, $\mathrm{Z} : \mathit{bitstring} \to \mathit{bitstring}$, *and the constant* $\mathrm{Z}_k : \mathit{bitstring}$*; the equations* (mac), (mac′), (enc), *and* $\forall x : T_k, \mathrm{Z}(\mathrm{k2b}(x)) = \mathrm{Z}_k$ *(which expresses that all keys have the same length); the indication that* k2b *and* $\mathrm{i}_\perp$ *are poly-injective (which generates the equations* (k2b) *and similar equations for* $\mathrm{i}_\perp$*); equivalences $L \approx_p R$ for MAC* ($\mathrm{mac}_{\mathrm{eq}}$) *and encryption* ($\mathrm{enc}_{\mathrm{eq}}$)*; and the process Q_0 of Example 11.1. Let $V = \{x_k''\}$.*

*The prover first applies **RemoveAssign**(x_{mk}) to the process Q_0 of Example 11.1, as described in Example 11.2, yielding Q_1. The process can then be transformed using the security of the MAC. Let $S = \{x_{mr}\}$, $M_1 = \mathrm{mac}(x_m[i], \mathrm{mkgen}(x_{mr}))$, $M_2 = \mathrm{verify}(x_m'[i'], \mathrm{mkgen}(x_{mr}), x_{ma}[i'])$, and $\mathcal{M} = \{M_1, M_2\}$. Hence $N_{M_1} = \mathrm{mac}(x[i'', i], \mathrm{mkgen}(r[i'']))$, $N_{M_2} = \mathrm{verify}(m[i'', i'], \mathrm{mkgen}(r[i'']), ma[i'', i'])$, $\mathrm{mapIdx}_{M_1}(a_1) = (1, a_1)$, and $\mathrm{mapIdx}_{M_2}(a_2) = (1, a_2)$, so $x_m[a_1]$ corresponds to $x[1, a_1]$, x_{mr} to $r[1]$, $x_m'[a_2]$ to $m[1, a_2]$, and $x_{ma}[a_2]$ to $ma[1, a_2]$.*

After transformation, we get the following process Q_2:

$$
\begin{aligned}
&Q_2 = \mathsf{in}(\mathit{start}, ()); \mathsf{new}\ x_r : T_r; \mathsf{let}\ x_k : T_k = \mathrm{kgen}(x_r)\ \mathsf{in}\ \mathsf{new}\ x_{mr} : T_{mr}; \mathsf{out}(c, ()); (Q_{2A} \parallel Q_{2B})\\
&Q_{2A} = !^{i \le n}\mathsf{in}(c_A[i], ()); \mathsf{new}\ x_k' : T_k; \mathsf{new}\ x_r' : T_r';\\
&\quad \mathsf{let}\ x_m : \mathit{bitstring} = \mathrm{enc}(\mathrm{k2b}(x_k'), x_k, x_r')\ \mathsf{in}\ \mathsf{out}(c_A[i], (x_m, \mathrm{mac}'(x_m, \mathrm{mkgen}'(x_{mr}))))\\
&Q_{2B} = !^{i' \le n}\mathsf{in}(c_B[i'], (x_m', x_{ma}));\\
&\quad \mathsf{find}\ u \le n\ \mathsf{suchthat}\ \mathsf{defined}(x_m[u]) \wedge x_m' = x_m[u] \wedge\\
&\qquad\qquad \mathrm{verify}'(x_m', \mathrm{mkgen}'(x_{mr}), x_{ma})\\
&\quad \mathsf{then}\ (\mathsf{if}\ \mathrm{true}\ \mathsf{then}\ \mathsf{let}\ \mathrm{i}_\perp(\mathrm{k2b}(x_k'')) = \mathrm{dec}(x_m', x_k)\ \mathsf{in}\ \mathsf{out}(c_B[i'], ()))\\
&\quad \mathsf{else}\ (\mathsf{if}\ \mathrm{false}\ \mathsf{then}\ \mathsf{let}\ \mathrm{i}_\perp(\mathrm{k2b}(x_k'')) = \mathrm{dec}(x_m', x_k)\ \mathsf{in}\ \mathsf{out}(c_B[i'], ()))
\end{aligned}
$$

The initial definition of x_{mr} is removed and replaced with a new definition, which we still call x_{mr}. The term $\mathrm{mac}(x_m, \mathrm{mkgen}(x_{mr}))$ *is replaced with* $\mathrm{mac}'(x_m, \mathrm{mkgen}'(x_{mr}))$*. The term* $\mathrm{verify}(x_m', \mathrm{mkgen}(x_{mr}), x_{ma})$ *becomes* $\mathsf{find}\ u \le n\ \mathsf{suchthat}\ \mathsf{defined}(x_m[u]) \wedge x_m' = x_m[u] \wedge \mathrm{verify}'(x_m', \mathrm{mkgen}'(x_{mr}), x_{ma})\ \mathsf{then}\ \mathrm{true}\ \mathsf{else}\ \mathrm{false}$*, which yields Q_{2B} after transformation of oracle processes into processes. The process looks up the message x_m' in the array x_m, which contains the messages whose MAC has been computed with key* $\mathrm{mkgen}(x_{mr})$*. If the MAC of x_m' has never been computed, the verification always fails (it returns* false*) by the security assumption on the MAC. Otherwise, it returns* true *when* $\mathrm{verify}'(x_m', \mathrm{mkgen}'(x_{mr}), x_{ma})$*. By instantiating* *the probability formula given in* $(\mathrm{mac_{eq}})$*,* $Q_1 \approx^V_{p'_{\mathrm{mac}}} Q_2$ *where*

$$\begin{aligned} p'_{\mathrm{mac}}(C, D) &= p_{\mathrm{mac}}(C[C'], D) \\ &= \mathsf{Succ}^{\mathsf{uf-cma}}_{\mathsf{MAC}}(t_C + \mathrm{time}(\mathrm{kgen}) + n\,\mathrm{time}(\mathrm{enc}, \mathrm{length}(T_k)) + n\,\mathrm{time}(\mathrm{dec}, \mathrm{maxl}(x'_m)), \\ &\qquad n, n, \max(\mathrm{maxl}(x'_m), \mathrm{maxl}(x_m))) \end{aligned}$$

since we use one MAC key $(n'' = 1)$*, there are at most* n *calls to* mac *and* verify *for that key* $(n' = n)$*, and the runtime of the adversary against* $(\mathrm{mac_{eq}})$ *is* $t_{C[C']} = t_C + \mathrm{time}(\mathrm{kgen}) + n\,\mathrm{time}(\mathrm{enc}, \mathrm{length}(T_k)) + n\,\mathrm{time}(\mathrm{dec}, \mathrm{maxl}(x'_m))$*.*

Applying ***Simplify*** *yields a game* Q_3*:* Q_{2A} *is unchanged and* Q_{2B} *becomes*

$$\begin{aligned} Q_{3B} = {} & !^{i' \le n}\mathsf{in}(c_B[i'], (x'_m, x_{ma})); \\ & \quad \mathsf{find}\ u \le n\ \mathsf{suchthat}\ \mathsf{defined}(x_m[u], x'_k[u]) \wedge x'_m = x_m[u] \wedge \\ & \qquad \mathrm{verify}'(x'_m, \mathrm{mkgen}'(x_{mr}), x_{ma})\ \mathsf{then} \\ & \quad \mathsf{let}\ x''_k : T_k = x'_k[u]\ \mathsf{in}\ \mathsf{out}(c_B[i'], ()) \end{aligned}$$

First, the tests if true then ... *and* if false then ... *are simplified. The term* $\mathrm{dec}(x'_m, x_k)$ *is simplified knowing* $x'_m = x_m[u]$ *by the* find *condition,* $x_m[u] = \mathrm{enc}(\mathrm{k2b}(x'_k[u]), x_k, x'_r[u])$ *by the assignment that defines* x_m*,* $x_k = \mathrm{kgen}(x_r)$ *by the assignment that defines* x_k*, and* $\mathrm{dec}(\mathrm{enc}(m, \mathrm{kgen}(r), r'), \mathrm{kgen}(r)) = \mathrm{i}_\perp(m)$ *by* (enc)*. So we have* $\mathrm{dec}(x'_m, x_k) = \mathsf{dec}(x_m[u], x_k) = \mathsf{dec}(\{()\}_{\mathrm{k2b}}(x'_k[u]), x_k, x'_r[u]), x_k) = \mathrm{i}_\perp(\mathrm{k2b}(x'_k[u]))$*. By injectivity of* $\mathrm{i}_\perp$ *and* k2b*, the assignment to* x''_k *simply becomes* $x''_k = x'_k[u]$*, using the equations* $\forall x : \mathit{bitstring}, \mathrm{i}_\perp^{-1}(\mathrm{i}_\perp(x)) = x$ *and* $\forall x : T_k, \mathrm{k2b}^{-1}(\mathrm{k2b}(x)) = x$*.*

After applying ***RemoveAssign***(x_k)*, which yields* Q_4*, we use the security of encryption, yielding* Q_5*:* $\mathrm{enc}(\mathrm{k2b}(x'_k), \mathrm{kgen}(x_r), x'_r)$ *becomes* $\mathrm{enc}'(\mathrm{Z}(\mathrm{k2b}(x'_k)), \mathrm{kgen}'(x_r), x'_r)$*. We have* $Q_4 \approx^V_{p'_{\mathrm{enc}}} Q_5$ *where*

$$\begin{aligned} p'_{\mathrm{enc}}(C, D) &= p_{\mathrm{enc}}(C[C''], D) \\ &= \mathsf{Succ}^{\mathsf{ind-cpa}}_{\mathsf{SE}}(t_C + t_D + (n + n^2)\mathrm{time}(\mathrm{mkgen}) + n\,\mathrm{time}(\mathrm{mac}, \mathrm{maxl}(m)) + \\ &\qquad\quad n^2\,\mathrm{time}(\mathrm{verify}, \mathrm{maxl}(m')) + n^2\,\mathrm{time}(=\!\mathit{bitstring}, \mathrm{maxl}(m'), \mathrm{maxl}(m)), \\ &\qquad n, \mathrm{length}(T_k)). \end{aligned}$$

(The evaluation of the runtime of the context C'' *is rather naive since we consider that* $\mathrm{mkgen}(x_{mr})$ *is computed once in each execution of* Q_{4A} *and once for each* find *test in* Q_{4B}*, and similarly* verify *is computed once for each* find *test in* Q_{4B}*. By noticing that it is enough to compute* $\mathrm{mkgen}(x_{mr})$ *once, and* verify *once in each execution of* Q_{4B}*, one would obtain* $\mathsf{Succ}^{\mathsf{ind-cpa}}_{\mathsf{SE}}(t_C + t_D + \mathrm{time}(\mathrm{mkgen}) + n\,\mathrm{time}(\mathrm{mac}, \mathrm{maxl}(m)) + n\,\mathrm{time}(\mathrm{verify}, \mathrm{maxl}(m')) + n^2\,\mathrm{time}(= \mathit{bitstring}, \mathrm{maxl}(m'), \mathrm{maxl}(m)), n, \mathrm{length}(T_k))$*.) After* ***Simplify****,* $\mathrm{enc}'(\mathrm{Z}(\mathrm{k2b}(x'_k)), \mathrm{kgen}'(x_r), x'_r)$ *becomes* $\mathrm{enc}'(\mathrm{Z}_k, \mathrm{kgen}'(x_r), x'_r)$*, using* $\forall x : T_k, \mathrm{Z}(\mathrm{k2b}(x)) = \mathrm{Z}_k$ *(which expresses that all keys have the same length).*

So we obtain the following game:

$$\begin{aligned} & Q_6 = \mathsf{in}(\mathit{start}, ()); \mathsf{new}\ x_r : T_r; \mathsf{new}\ x_{mr} : T_{mr}; \mathsf{out}(c, ()); (Q_{6A} \parallel Q_{6B}) \\ & Q_{6A} = !^{i \le n}\mathsf{in}(c_A[i], ()); \mathsf{new}\ x'_k : T_k; \mathsf{new}\ x'_r : T'_r; \\ & \quad \mathsf{let}\ x_m : \mathit{bitstring} = \mathrm{enc}'(\mathrm{Z}_k, \mathrm{kgen}'(x_r), x'_r)\ \mathsf{in}\ \mathsf{out}(c_A[i], (x_m, \mathrm{mac}'(x_m, \mathrm{mkgen}'(x_{mr})))) \\ & Q_{6B} = Q_{3B} \end{aligned}$$

By transitivity of $\approx$ *(Lemma 11.1),* $Q_0 \approx^V_{p'_{\mathrm{mac}} + p'_{\mathrm{enc}}} Q_6$ *since the probability is 0 for steps other than applying the security of MAC and encryption.*

Using lists instead of arrays simplifies games transformation: we do not need to add instructions that insert values in the list, since all variables are always implicitly arrays. Moreover, if there are several occurrences of $\mathrm{mac}(x_i, k)$ with the same key in the initial process, each $\mathrm{verify}(m_j, k, ma_j)$ is replaced with a find with one branch for each occurrence of mac. Therefore, the prover distinguishes automatically the cases in which the verified MAC ma_j comes from each occurrence of mac, that is, it distinguishes cases depending on the value of i such that $m_j = x_i$. Typically, distinguishing these cases is useful in the following steps of the proof of the protocol. (A similar situation arises for other cryptographic primitives specified using find.)

11.4 Criteria for Proving Secrecy Properties

Let us now define syntactic criteria that allow us to prove secrecy properties of protocols. The proofs of asymptotic versions of these results can be found in [?, Appendix E.5].

Definition 11.4 (One-session secrecy) *Suppose that the variable x of type T is defined in Q under a single $!^{i \leq n}$. Q* preserves the one-session secrecy of *x up to probability p when, for all evaluation contexts C acceptable for $Q \parallel Q_x$ without public variables that do not contain S, $2\Pr[C[Q \parallel Q_x] : \mathsf{S}] - 1 \leq p(C)$, where*

$$\begin{aligned} Q_x = {} & \mathsf{in}(c_0, ()); \mathsf{new}\ b : \mathit{bool}; \mathsf{out}(c_0, ()); \\ & (\mathsf{in}(c, u : [1, n]); \mathsf{if}\ \mathsf{defined}(x[u])\ \mathsf{then}\ \mathsf{if}\ b\ \mathsf{then}\ \mathsf{out}(c, x[u])\ \mathsf{else}\ \mathsf{new}\ y : T; \mathsf{out}(c, y) \\ & \ \parallel \mathsf{in}(c', b' : \mathit{bool}); \mathsf{if}\ b = b'\ \mathsf{then}\ \mathsf{event}\ \mathsf{S}) \end{aligned}$$

c_0, c, c', b, b', u, y, and S do not occur in Q.

Intuitively, the adversary C distinguishes the value of each secret $x[u]$ from a random number with probability at most $p(C)$. (But the elements $x[u]$ need not be independent.) The adversary performs a single test query on $x[u]$, modeled by sending u on channel c in Q_x. This test query returns $x[u]$ when the random bit b is true and a random number otherwise. The adversary then tries to guess b, by sending its guess b' on channel c'. When the guess is correct, event S is executed.

Proposition 11.4 (One-session secrecy) *Consider a process Q such that there exists a set of variables S such that 1) the definitions of x are either restrictions $\mathsf{new}\ x[\widetilde{i}] : T$ and $x \in S$, or assignments $\mathsf{let}\ x[\widetilde{i}] : T = z[M_1, \ldots, M_l]$ where z is defined by restrictions $\mathsf{new}\ z[\widetilde{i'_1, \ldots, i'_l}] : T$, and $z \in S$, and 2) all accesses to variables $y \in S$ in Q are of the form "$\mathsf{let}\ y'[\widetilde{i}] : T' = y[M_1, \ldots, M_l]$" with $y' \in S$. Then Q preserves the one-session secrecy of x up to probability 0.*

Intuitively, only the variables in S depend on the restriction that defines x; the sent messages and the control flow of the process are independent of x, so the adversary obtains no information on x. In the implementation, the set S is computed by fixpoint iteration, starting from x or z and adding variables y' defined by "$\mathsf{let}\ y'[\widetilde{i}] : T' = y[M_1, \ldots, M_l]$" when $y \in S$.

Definition 11.5 (Secrecy) *Assume that the variable x of type T is defined in Q under a single $!^{i \leq n}$. Q* preserves the secrecy of *x up to probability p when, for all evaluation contexts C acceptable for $Q \parallel R_x$ without public variables that do not contain S, $2\Pr[C[Q \parallel R_x] : \mathsf{S}] - 1 \leq p(C)$, where*

$$\begin{aligned} R_x = {} & \mathsf{in}(c_0, ()); \mathsf{new}\ b : \mathit{bool}; \mathsf{out}(c_0, ()); \\ & (!^{i \leq n'} \mathsf{in}(c, u : [1, n]); \mathsf{if}\ \mathsf{defined}(x[u])\ \mathsf{then}\ \mathsf{if}\ b\ \mathsf{then}\ \mathsf{out}(c, x[u])\ \mathsf{else} \\ & \quad \mathsf{find}\ u' \leq n'\ \mathsf{suchthat}\ \mathsf{defined}(y[u'], u[u']) \wedge u[u'] = u\ \mathsf{then}\ \mathsf{out}(c, y[u']) \end{aligned}$$

$$\begin{array}{l} \quad \mathsf{else\ new}\ y : T; \mathsf{out}(c, y) \\ \| \ \mathsf{in}(c', b' : \mathit{bool}); \mathsf{if}\ b = b'\ \mathsf{then\ event}\ \mathsf{S}) \end{array}$$

c_0, c, c', b, b', u, u', y, and S *do not occur in Q, and $n' \geq n$.*

Intuitively, the adversary C distinguishes the secret array x from an array of independent random numbers with probability at most $p(C)$. In this definition, the adversary can perform several test queries, modeled by R_x, which all return the value of x if b is true and a random number if b is false. This corresponds to the "real-or-random" definition of security [**?**]. (As shown in [**?**], this notion is stronger than the more standard approach in which the adversary can perform a single test query and some reveal queries, which always reveal $x[u]$.)

Proposition 11.5 (Secrecy) *Assume that Q satisfies the hypothesis of Proposition 11.4.*

When $\mathcal{T}$ is a trace of $C[Q]$ for some evaluation context C, we define $\mathrm{defRestr}_{\mathcal{T}}(x[\widetilde{a}])$*, the defining restriction of $x[\widetilde{a}]$ in trace $\mathcal{T}$, as follows: if $x[\widetilde{a}]$ is defined by* $\mathsf{new}\ x[\widetilde{a}] : T$ *in $\mathcal{T}$,* $\mathrm{defRestr}_{\mathcal{T}}(x[\widetilde{a}]) = x[\widetilde{a}]$*; if $x[\widetilde{a}]$ is defined by* $\mathsf{let}\ x[\widetilde{a}] : T = z[M_1, \dots, M_l]$, $\mathrm{defRestr}_{\mathcal{T}}(x[\widetilde{a}]) = z[a'_1, \dots, a'_l]$ *where $E, M_k \Downarrow a'_k$ for all $k \leq l$ and E is the environment in $\mathcal{T}$ at the definition of $x[\widetilde{a}]$.*

For all evaluation contexts C acceptable for Q with public variables $\{x\}$, let $p(C) = \Pr[\exists(\mathcal{T}, \widetilde{a}, \widetilde{a'}), C[Q]$ *reduces according to* $\mathcal{T} \wedge \widetilde{a} \neq \widetilde{a'} \wedge \mathrm{defRestr}_{\mathcal{T}}(x[\widetilde{a}]) = \mathrm{defRestr}_{\mathcal{T}}(x[\widetilde{a'}])]$. *Then Q preserves the secrecy of x up to probability $2p$.*

The collisions $\mathrm{defRestr}_{\mathcal{T}}(x[\widetilde{a}]) = \mathrm{defRestr}_{\mathcal{T}}(x[\widetilde{a'}])$ are eliminated using the same equational prover as for **Simplify** in Section 11.3.1, which yields a bound on $p(C)$. Intuitively, when $\widetilde{a} \neq \widetilde{a'}$, we have $\mathrm{defRestr}_{\mathcal{T}}(x[\widetilde{a}]) \neq \mathrm{defRestr}_{\mathcal{T}}(x[\widetilde{a'}])$ (except in cases of probability $p(C)$), so $x[\widetilde{a}]$ and $x[\widetilde{a'}]$ are defined by different restrictions, so they are independent random numbers.

As we show in [**?**], secrecy composed with correspondence assertions [**?**] can be used to prove security of a key exchange. (Correspondence assertions are properties of the form "if some event $e(\widetilde{M})$ has been executed then some events $e_i(\widetilde{M_i})$ for $i \leq m$ have been executed". The verification of correspondence assertions in CryptoVerif in presented in [**?**].)

Lemma 11.2 *If $Q \approx_p^{\{x\}} Q'$ and Q preserves the one-session secrecy of x up to probability p' then Q' preserves the one-session secrecy of x up to probability* $p''(C) = p'(C) + 2p(C[[\,] \parallel Q_x], \mathsf{S})$. *A similar result holds for secrecy.*

We can then apply the following technique. When we want to prove that Q_0 preserves the (one-session) secrecy of x, we transform Q_0 by the transformations described in Section 11.3 with $V = \{x\}$. By Propositions 11.1 and 11.3, we obtain a process Q'_0 such that $Q_0 \approx_p^V Q'_0$. We use Propositions 11.4 or 11.5 to show that Q'_0 preserves the (one-session) secrecy of x and finally conclude that Q_0 also preserves the (one-session) secrecy of x up to a certain probability by Lemma 11.2.

Example 11.5 *After the transformations of Example 11.4, the only variable access to x'_k in the considered process is* $\mathsf{let}\ x''_k : T_k = x'_k[u]$ *and x''_k is not used in the considered process. So by Proposition 11.4, the considered process preserves the one-session secrecy of x''_k (with $S = \{x'_k, x''_k\}$). By Lemma 11.2, the process of Example 11.1 also preserves the one-session secrecy of x''_k up to probability* $2(p'_{\mathrm{mac}} + p'_{\mathrm{enc}})(C[[\,] \parallel Q_x], \mathsf{S})$. *(The runtimes of Q_x and of the distinguisher* S *can be neglected inside this formula.) However, this process does not preserve the secrecy of x''_k, because the adversary can force several sessions of B to use the same key x''_k, by replaying the message sent by A. Accordingly, the hypothesis of Proposition 11.5 is not satisfied.*

The criteria given in this section might seem restrictive, but in fact, they should be sufficient for all protocols, provided the previous transformation steps are powerful enough to transform the protocol into a simpler protocol, on which these criteria can then be applied.

11.5 Proof Strategy

Up to now, we have described the available game transformations. Next, we explain how we organize these transformations in order to prove protocols.

At the beginning of the proof and after each successful cryptographic transformation (that is, a transformation of Section 11.3.2), the prover executes **Simplify** and tests whether the desired security properties are proved, as described in Section 11.4. If so, it stops.

In order to perform the cryptographic transformations and the other syntactic transformations, our proof strategy relies of the idea of advice. Precisely, the prover tries to execute each available cryptographic transformation in turn. When such a cryptographic transformation fails, it returns some syntactic transformations that could make the desired transformation work. (These are the advised transformations.) Then the prover tries to perform these syntactic transformations. If they fail, they may also suggest other advised transformations, which are then executed. When the syntactic transformations finally succeed, we retry the desired cryptographic transformation, which may succeed or fail, perhaps with new advised transformations, and so on.

Examples of advised transformations include:

- Assume that we try to execute a cryptographic transformation, and need to recognize a certain term M of L, but we find in Q_0 only part of M, the other parts being variable accesses $x[\ldots]$ while we expect function applications. In this case, we advise **RemoveAssign**(x). For example, if Q_0 contains $\mathrm{enc}(M', x_k, x'_r)$ and we look for $\mathrm{enc}(x_m, \mathrm{kgen}(x_r), x'_r)$, we advise **RemoveAssign**(x_k). If Q_0 contains $\mathsf{let}\ x_k = \mathrm{mkgen}(x_r)$ and we look for $\mathrm{mac}(x_m, \mathrm{mkgen}(x_r))$, we also advise **RemoveAssign**(x_k). (The transformation of Example 11.2 is advised for this reason.)

- When we try to execute **RemoveAssign**(x), x has several definitions, and there are accesses to variable x guarded by find in Q_0, we advise **SArename**(x).

- When we want to prove that x is secret or one-session secret, we have an assignment $\mathsf{let}\ x[\widetilde{i}] : T = y[\widetilde{M}]\ \mathsf{in}\ P$, and there is at least one assignment defining y, we advise the transformation **RemoveAssign**(y).

  When we want to prove that x is secret or one-session secret, we have an assignment $\mathsf{let}\ x[\widetilde{i}] : T = y[\widetilde{M}]\ \mathsf{in}\ P$, y is defined by restrictions, y has several definitions, and some variable accesses to y are not of the form $\mathsf{let}\ y'[\widetilde{i'}] : T = y[\widetilde{M'}]\ \mathsf{in}\ P'$, we advise **SArename**$(y)$.

11.6 Experimental Results

CryptoVerif has been tested on a number of protocols given in the literature. We proved secrecy of keys for the Otway-Rees and Yahalom protocols as well as original and corrected versions of the Needham-Schroeder shared-key and public-key and Denning-Sacco public-key protocols, as reported in [?]. We proved authentication properties for these protocols as well as for original and corrected versions of the Woo-Lam shared-key and public-key protocols [?]. The proof succeeded in most cases (it failed for only 3 properties that in fact hold). For some proofs, for public-key protocols, we needed to provide manual indications of the game transformations to perform, mainly because several game transformations are sometimes applicable, and the proof succeeds only for a particular choice of the applied game transformation.

For each proof, the prover outputs the sequence of games it has built, a succinct explanation of the transformation performed between consecutive games, and an indication of whether the proof succeeded or failed. When the proof fails, the prover still outputs a sequence of games, but the last game of this sequence does not show the desired property and cannot be transformed further by the prover. Manual inspection of this game often makes it possible to understand why the proof failed: because there is an attack (if there is an attack on the last game), because of a limitation of the prover (if it should in fact be able to prove the property or to transform the game further), for other reasons (such as the protocol cannot be proved from the given assumptions; this situation may not lead immediately to a practical attack in the computational model).

CryptoVerif can also be used for proving cryptographic schemes, such as the FDH signature scheme [**?**]. It has been used for studying more complex protocols: the Kerberos protocol, with and without its public-key extension PKINIT [**?**], as well as parts of the record protocol and of the handshake protocol of TLS [**?**].

11.7 Conclusion

CryptoVerif produces proofs by sequences of games, in the computational model. The security assumptions on primitives are given as observational equivalences, which are proved once for each primitive and can be reused for proving many different protocols. The protocol or cryptographic scheme to prove is specified in a process calculus. CryptoVerif provides the sequence of games that leads to the proof and a bound on the probability of success of an attack. The user is allowed, but does not have, to provide manual indications on the game transformations to perform.

The essential idea of simulating proofs by sequences of games in an automatic tool can be applied to any protocol or cryptographic scheme. However, CryptoVerif applies in a fairly direct way the security assumptions on the primitives and cannot perform deep mathematical reasoning. Therefore, it is best suited for proving security protocols that use rather high-level primitives such as encryption and signatures. It is more limited for proving the security of such primitives from lower-level primitives, since more subtle mathematical arguments are often needed.

Future work includes adding support for more primitives, for example associativity for exclusive or and primitives with internal state. Improvements in the proof strategy and the possibility to give more precise manual hints would also be useful. Future case studies will certainly suggest additional extensions. In the long term, it would be interesting to certify CryptoVerif, possibly to combine it with the Coq-based framework CertiCrypt [**?**]. Grand challenges include the proof of protocol implementations in the computational model, by analyzing them (as started in [**?**] for instance) or by generating them from specifications, and taking into account side-channel attacks.

11.8 More Exercises

The next definition will be useful in Exercise 46.

Definition 11.6 (IND-CCA2 symmetric encryption) *A symmetric encryption scheme* SE *is* indistinguishable under adaptive chosen-ciphertext attacks (IND-CCA2) *if and only if the probability* $\mathsf{Succ}^{\mathsf{ind-cca2}}_{\mathsf{SE}}(t, q_e, q_d, l_e, l_d)$ *is negligible when* t *is polynomial in the security parameter:*

$$\mathsf{Succ}^{\mathsf{ind-cca2}}_{\mathsf{SE}}(t, q_e, q_d, l_e, l_d) =$$

$$\max_{\mathcal{A}} 2 \Pr \begin{bmatrix} b \overset{R}{\leftarrow} \{0,1\}; k \overset{R}{\leftarrow} \mathrm{kgen}; b' \leftarrow \mathcal{A}^{\mathrm{enc}(LR(.,.,b),k),\mathrm{dec}(.,k)} : b' = b \wedge \\ \mathcal{A} \textit{ has not called } \mathrm{dec}(.,k) \textit{ on the result of } \mathrm{enc}(LR(.,.,b),k) \end{bmatrix} - 1$$

where $\mathcal{A}$ runs in time at most t, calls $\mathrm{enc}(LR(.,.,b),k)$ *at most q_e times on messages of length at most l_e, calls* $\mathrm{dec}(.,k)$ *at most q_d times on messages of length at most l_d.*

Exercice 46

1. Show using CryptoVerif that, if the MAC scheme is SUF-CMA and the encryption scheme is IND-CPA, then the encrypt-then-MAC scheme is IND-CCA2.

   We recall that, in the encrypt-then-MAC scheme, the encryption of m under key (k, mk) is $(e, \mathrm{mac}(e, mk))$ where $e = \mathrm{enc}(m, k, r)$ and r represents random coins.

2. Show using the same assumptions that the encrypt-then-MAC scheme is INT-CTXT.

3. What happens if the MAC scheme is only UF-CMA?

For the next exercise, please read [**?**] first.

Exercice 47

Suppose that H is a hash function in the Random Oracle Model and that f is a one-way trapdoor permutation.

Consider the encryption function $E_{pk}(x) = f_{pk}(r)||H(r) \oplus x$, where || denotes concatenation and $\oplus$ denotes exclusive or (Bellare & Rogaway, CCS'93).

1. What is the decryption function?

2. Show using CryptoVerif that this public-key encryption scheme is IND-CPA. (IND-CPA is defined like IND-CCA2 except that the adversary does not have access to a decryption oracle.)

Part IV

Links between the two Settings

In this part, we will relate several previous parts of these lecture notes, trying to answer, at least partly, the paradoxical situation described in the section 1.3. The goal is to explicit under which conditions, the symbolic reasoning that has been developed in the chapters 3,5,6,4 accounts for any attack that could be mounted in the model that described in the chapters 7,11.

In other words: when is the term model and the associated definitions of security properties accurate enough, so as to cover all attacks that could be mounted by an arbitrary Probabilistic Polynomial Time attacker ? At a first glance, the computational attacker seems to be more powerful since it can perform any (polynomial time) computation, whereas the symbolic one can only forge messages using a fixed set of primitives (corresponding to the function symbols in the signature), all of which computable in polynomial time. The *soundness* results show in which cases the additional capabilities of a computational attacker are actually useless.

In their seminal research paper [**?**], M. Abadi and P. Rogaway show the first soundness result, roughly proving that the static equivalence (defined for instance in the chapter 5) implies the computational indistinguishability, in case of a symmetric IND-CPA encryption scheme. This is the first result that we want to demonstrate in these lecture notes.

Chapter 12

Soundness of Static Equivalence

We will show that under some assumptions on the cryptographic primitives static equivalence is sound with respect to a computational model, i.e. whenever two frames φ_1 and φ_2 are statically equivalent, the distributions corresponding to the implementations of these frames are computationally indistinguishable.

12.1 Security properties of symmetric encryption schemes

We recast first the security definition of IND-CPA (defined in the section 8.1 for public-key encryption), in the case of symmetric key encryption.

We wrile $\mathcal{A}^{\mathcal{O}}$ a Probabilistic Polynomial Time Turing machine, equipped with an oracle $\mathcal{O}$. Let us recall that such machines include in particular a random tape, which is read-only and whose content is drawn uniformly at random when the machine starts. The polynomial time computation should only depend on the input of the machine, not on the actual values on the random tape. We sometimes write $\mathcal{A}(x \mid R)$ for the result of the (deterministic) computation of $\mathcal{A}$ on x with a random tape R.

The machine has also a special tape for oracle calls (and replies). It may write on this tape and, from a special state corresponding to the oracle call, there is a transition of the machine from a configuration γ to a configuration in which only the control state and the content of the oracle tape have changed; if the oracle tape contains m before the call to the oracle, it contains $\mathcal{O}(m)$ after the transition. In case the oracle itself is randomized, it is assumed to be equiped with a random (infinite) string R, which is drawn at its first call. Each time the oracle needs a random input, it takes the appropriate prefix of R and removes this prefix from R. When needed, we write $\mathcal{O}(m \mid R)$ to explicitly state what is the random input of the oracle.

The following definition captures the minimal expectations for a symmetric encryption scheme: key generation/ encryption/decryption can be performed in polynomial time and the decryption with a correct key of the encryption of a plaintext gives back the plaintext.

Definition 12.1 *A* symmetric encryption scheme *consists of three deterministic polynomial time functions $\mathcal{G}, \mathcal{E}, \mathcal{D}$.*

- *$\mathcal{G}$ is the* key generation algorithm*. We assume here that the length of $\mathcal{G}(x)$ only depends on the length of x.*

- *$\mathcal{E}$ is an encryption algorithm, that, given x, k, r (a plaintext, a key and a random seed) returns $\mathcal{E}(x, k, r)$. We assume that the length of $\mathcal{E}(x, k, r)$ only depends on the length of x for a fixed length key k. $\mathcal{E}(x, \mathcal{G}(y), r)$ is also assume to depend only on a prefix of length $|y|$ of r: $\mathcal{E}(x, \mathcal{G}(y), r_1) = \mathcal{E}(x, \mathcal{G}(y), r_2)$ if r_1 and r_2 have the same prefix of length $|y|$.*

- $\mathcal{D}$ *is the decryption algorithm. It is assumed to satisfy the equation*

$$\mathcal{D}(\mathcal{E}(x, \mathcal{G}(y), r), \mathcal{G}(y)) = x$$

for all x, y, r.

Note that in case of key mismatches, or a key that is not in the range of $\mathcal{G}$, the result of $\mathcal{D}$ is not specified.

Sometimes it is more convenient to hide the key generation algorithm and to use a key distribution. Given $\eta \in \mathbb{N}$, we write $\mathcal{K}(\eta)$ the distribution defined by the image of the uniform distribution on $\{0,1\}^\eta$ by $\mathcal{G}$: for every a, $\mathbb{P}[k \leftarrow \mathcal{K}(\eta) : k = a] = \mathbb{P}[r \leftarrow U(\{0,1\}^\eta) : \mathcal{G}(r) = a]$.

Now, as in the section 8.1, we are going to use encryption oracles. Given a symmetric encryption scheme $\mathcal{G}, \mathcal{E}, \mathcal{D}$ and a key k in $\mathcal{G}(\{0,1\}^\eta)$, we define the two randomized oracles $\mathcal{O}_k^1(\_ \mid R)$ and $\mathcal{O}_k^2(\_ \mid R)$ as follows:

- on an input $x\#y$ where $x, y \in \{0,1\}^*$ and $|x| = |y|$, $\mathcal{O}_k^1(x\#y \mid R)$ returns $\mathcal{E}(x, k, r)$ and $\mathcal{O}_k^2(x\#y \mid R)$ returns $\mathcal{E}(y, k, r)$, for a bitstring r that is taken from R.
- If the input does not have the above format, $\mathcal{O}_k^1$ (resp. $\mathcal{O}_k^2$) returns 0.

Note that two successive calls to $\mathcal{O}_k^i$ with the same input may return different values, as the value r is drawn at each call.

Finally, the security cannot be ensured for fixed length keys (there is always then an attacker); it is rather an asymptotic property. We therefore use the following definition of negligible functions:

Definition 12.2 *A function $f : \mathbb{N} \to \mathbb{Q}$ is* negligible *if, for any positive polynomial P in one variable, there is an $N \in \mathbb{N}$ such that, for every $\eta > N$, $f(\eta) < \frac{1}{P(\eta)}$.*

We are now ready to define IND-CPA:

Definition 12.3 *Let $\mathcal{S} = (\mathcal{G}, \mathcal{E}, \mathcal{D})$ be a symmetric encryption scheme.*
Given any oracle PPT machine $\mathcal{A}$ and any security parameter $\eta \in \mathbb{N}$ we define

$$\begin{aligned}\mathrm{Adv}(\mathcal{A}, \eta) = \\ &|\mathbb{P}[k \leftarrow \mathcal{K}(\eta), R_1, R_2 \leftarrow U : \mathcal{A}^{\mathcal{O}_k^1(\_|R_2)}(0^\eta \mid R_1) = 1] \\ &\quad -\mathbb{P}[k \leftarrow \mathcal{K}(\eta), R_1, R_2 \leftarrow U : \mathcal{A}^{\mathcal{O}_k^2(\_|R_2)}(0^\eta \mid R_1) = 1]|\end{aligned}$$

$\mathcal{S}$ is IND-CPA *if, for any PPT machine $\mathcal{A}$,* $\mathrm{Adv}(\mathcal{A}, \eta)$ *is a negligible function of η.*

This property states intuitively that an attacker cannot distinguish between the encryption of two plaintexts of his choice.

Exercice 48
Show that the following is a symmetric encryption scheme and is not IND-CPA:

- $\mathcal{G}$ is the identity
- $\mathcal{E}(x, k, r) = x$
- $\mathcal{D}(y, k) = y$

Exercice 49
Show that any encryption scheme, in which $\mathcal{E}(x, k, r)$ is independent of r is not IND-CPA.

Exercice 50
Show that there is no encryption scheme that satisfies

$$\forall P, \exists N, \forall \mathcal{A}, \forall \eta > N. \quad \mathrm{Adv}(\mathcal{A}, \eta) < \tfrac{1}{P(\eta)}$$

Where $\mathcal{A}$ ranges over PPTs and P over positive polynomials in one variable.

Exercice 51
Show that there is no symmetric encryption scheme that satisfies

$$\forall \mathcal{A}, \exists N, \forall \eta > N. \ \mathrm{Adv}(\mathcal{A}, \eta) < \frac{1}{2^\eta}$$

where $\mathcal{A}$ ranges over PPTs.

Exercice 52
Given a symmetric encryption scheme, we let

$$\mathrm{Adv}'(\mathcal{A}, \eta) = |2 \times \mathbb{P}[b \leftarrow U(\{1,2\}), k \leftarrow \mathcal{K}(\eta), R_1, R_2 \leftarrow U : \ \mathcal{A}^{\mathcal{O}_k^b(\_|R_1)}(0^\eta \mid R_2) = 1] - 1|$$

Show that IND-CPA is equivalent to:

For every PPT $\mathcal{A}$, $\mathrm{Adv}'(\mathcal{A}, \eta)$ is a negligible function of η.

Exercice 53
Given a symmetric encryption scheme, we define

$$\mathrm{Adv}''(\mathcal{A}, \eta) = \mathbf{Average}[k \leftarrow \mathcal{K}(\eta) : \begin{array}{c} |\mathbb{P}[R_1, R_2 \leftarrow U : \mathcal{A}^{\mathcal{O}_k^1(\_|R_1)}(0^\eta \mid R_2) = 1] \\ - \quad \mathbb{P}[R_1, R_2 \leftarrow U : \mathcal{A}^{\mathcal{O}_k^2(\_|R_1)}(0^\eta \mid R_2) = 1]| \end{array}]$$

Is IND-CPA equivalent to the following property:

For every PPT $\mathcal{A}$, $\mathrm{Adv}''(\mathcal{A}, \eta)$ is a negligible function of η.

12.2 The symbolic model

We consider here a fixed set of function symbols $\mathcal{F}$: symmetric encryption $\{\_\}_\_^\_$, pairing $\langle \_, \_ \rangle$, symmetric decryption $\mathsf{dec}(\_, \_)$, projections $\pi_1(\_), \pi_2(\_)$, as well as a collection of constants $\mathcal{W}$. In addition, $\mathcal{N}$ is a set of names. This set of names can be partitioned into different sets, for instance keys, random seeds and nonces. For simplicity, we are going to consider in what follows only one name sort.

We also consider the equational theory E:

$$\begin{array}{ll} \mathsf{dec}(\{x\}_k^r, k) = x & \text{For every } k, r \in \mathcal{N} \\ \pi_1(\langle x, y \rangle) = x & \pi_2(\langle x, y \rangle) = y \end{array}$$

Orienting the equations from left to right, we get a (recursive) convergent rewriting system: every term u in $T(\mathcal{F}, \mathcal{X})$ has a unique normal form $u \downarrow$.

In what follows, we consider only (for simplicity) the set of *valid terms* $\mathcal{M}_0$, that is the least set of terms such that:

- $\mathcal{N} \cup \mathcal{W} \subseteq \mathcal{M}_0$
- if $u, v \in \mathcal{M}_0$, then $\mathsf{pair} uv \in \mathcal{M}_0$
- if $u \in \mathcal{M}_0$, $r, k \in \mathcal{N}$, then $\{u\}_k^r \in \mathcal{M}_0$

For convenience, we re-define the static equivalence (and we will see later that it may match the definition of chapter 5: instead of only checking equalities, we give the attacker the ability to check some other predicates.

Valid messages: the unary predicate symbol M is assumed to check the well-formedness of messages.

Its interpretation in our message structure is set M^I of ground terms u such that $u \downarrow \in \mathcal{M}_0$.

Equality: the binary predicate EQ checks the equality of messages: its interpretation EQ^I is the set of pairs of messages (u, v) such that $u \in M^I$, $v \in M^I$ and $u\downarrow = v\downarrow$.

Note that, for instance $(\mathsf{dec}(k,k), \mathsf{dec}(k,k)) \notin EQ^I$: pairs of ill-formed terms are not considered as equal.

Equal keys: as we will see, we will need a predicate EK checking that two ciphertexts use the same encryption keys (the indistinguishability of two such ciphertexts is not guaranteed by IND-CPA).

EK^I is true on pairs of ciphertexts that are using the same encryption key: $(u, v) \in SK^I$ iff there is a $k \in \mathcal{N}$, there are $r_1, r_2 \in \mathcal{N}$, there are terms $u_1, v_1 \in \mathcal{M}_0$ such that $u \downarrow = \{u_1\}_k^{r_1}$ and $v \downarrow = \{v_1\}_k^{r_2}$.

Equal lengths: in the computational model, if two plaintexts have different lengths, then the corresponding ciphertexts have different lengths. Hence we need to reflect this ability to distinguish messages in the symbolic model. Formally, we use the binary predicate symbol EL, whose interpretation will be formally defined later in this section. Informally, EL^I is the pair of terms (u, v) such that there are terms $u_1, v_1 \in \mathcal{M}_0$, names k_1, k_2, r_1, r_2 such that $u \downarrow = \{u_1\}_{k_1}^{r_1}$, $v \downarrow = \{v_1\}_{k_2}^{r_2}$ and, for every $\eta \in \mathbb{N}$, $l(u_1, \eta) = l(v_1, \eta)$.

Let us recall that a *frame* is an expression $\nu\overline{n}.\{x_1 \mapsto s_1, \ldots, x_m \mapsto s_m\}$ where $s_1, \ldots, s_m$ are ground terms, $x_1, \ldots, x_m$ are distinct variables and $\overline{n}$ is a sequence of distinct names.

The *free names* $fn(\phi)$ of a frame $\phi = \nu\overline{n}.\{x_1 \mapsto s_1, \ldots, x_m \mapsto s_m\}$. are the names appearing in $s_1, \ldots, s_m$, that are not in $\overline{n}$. If $\phi = \nu\overline{n}.\{x_1 \mapsto s_1, \ldots, x_m \mapsto s_m\}$, we write σ_ϕ the substitution $\{x_1 \mapsto s_1, \ldots, x_m \mapsto s_m\}$.

A frame is defined up to the renaming of the names in $\overline{n}$: $\phi = \nu n_1, \ldots, n_k.\sigma_\phi$ is considered to be the same frame as $\phi' = \nu n'_1, \ldots, n'_k.\sigma_{\phi'}$ if $fn(\phi) = fn(\phi')$ and $\sigma_{\phi'}$ is obtained by replacing each n_i with n'_i in σ_ϕ.

Definition 12.4 *Given a set of predicate symbols $\mathcal{P}$, two frames $\phi_1 = \nu\overline{n_1}.\{x_1 \mapsto s_1, \ldots, x_k \mapsto s_k\}$ and $\phi_2 = \nu\overline{n_2}.\{x_1 \mapsto t_1, \ldots, x_m \mapsto t_m\}$, such that $fn(\phi_1) \cap \overline{n_2} = fn(\phi_2) \cap \overline{n_1} = \emptyset$, are* statically equivalent*, which we write $\phi_1 \sim \phi_2$, if $k = m$ and*

$$\begin{array}{c}\forall P \in \mathcal{P}, \quad \forall u_1, \ldots, u_i \in T(\mathcal{F} \cup (\mathcal{N} \setminus (\overline{n_1} \cup \overline{n_2})), \{x_1, \ldots, x_k\}), \\ (u_1\sigma_{\phi_1}, \ldots, u_i\sigma_{\phi_1}) \in P^I \quad \Leftrightarrow \quad (u_1\sigma_{\phi_2}, \ldots, u_i\sigma_{\phi_2}) \in P^I\end{array}$$

Exercice 54
Let $\mathcal{F}$ be the set of function symbols that has been defined in the beginning of this section and $\mathcal{P}$ be $\{M, EQ, EK\}$. Show that the above definition coincides with the definition of chapter 5, for a well chosen (recursive) equational theory $\mathcal{E}$.

12.3 Indistinguishability of ensembles

Two sequences of distributions (called *ensembles*) parametrized by $\eta \in \mathbb{N}$ are indistinguishable, if any PPT adversary, when faced to the two experiments, cannot guess with a significant advantage with which of the two experiments it is faced:

Definition 12.5 *Let $D = \{D_\eta\}_{\eta\in\mathbb{N}}$ and $D' = \{D'_\eta\}_{\eta\in\mathbb{N}}$ be two ensembles. D and D' are* computationally indistinguishable*, which is written $D \approx D'$ if, for any PPT $\mathcal{A}$, the advantage:*

$$\epsilon(\mathcal{A},\eta) = |\mathbb{P}[x \leftarrow D_\eta, r \leftarrow U : \mathcal{A}(x, 0^\eta \mid r) = 1] - \mathbb{P}[x \leftarrow D'_\eta, r \leftarrow U : \mathcal{A}(x, 0^\eta \mid r) = 1]|$$

is a negligible function of η.

Exercice 55
Show that the Dirac ensemble defined by $\mathbb{P}[x \leftarrow \delta_\eta : x = 0^\eta] = 1$ and the uniform ensemble $\mathbb{P}[x \leftarrow U(\{0,1\}^\eta) : x = a] = \frac{1}{2^\eta}$ for every $a \in \{0,1\}^\eta$ are distinguishable.

Exercice 56
Fix $k \in \mathbb{N}$. Show that the two following ensembles are indistinguishable: the uniform distribution on $\{0,1\}^\eta$ and the distribution U^k_η defined by:

$$\mathbb{P}[x \leftarrow U^k_\eta : x = a] = \begin{cases} 0 & \text{if } a = b0^{n-k} \text{ for some } b \\ \frac{1}{2^n - 2^k} & \text{Otherwise} \end{cases}$$

12.4 The computational interpretation of terms

We let $\mathcal{G}, \mathcal{E}, \mathcal{D}$ be a symmetric encryption scheme and assume that p is a polynomial time pairing function on bitstrings: p is an injection from $\{0,1\}^* \times \{0,1\}^*$ intp $\{0,1\}^*$, whose two inverses p_1^{-1} and p_2^{-1} are also polynomially computable and such that, forall $x, y \in \{0,1\}^*$, $p_1^{-1}(p(x,y)) = x$ and $p_2^{-1}(p(x,y)) = y$. We also assume that, if $|x_1| = |x_2|$ and $|y_1| = |y_2|$, then $|p(x_1,x_2)| = |p(y_1,y_2)|$.

We define here, for each security parameter $\eta \in \mathbb{N}$ the interpretation of terms as bitstrings. First, each $w \in \mathcal{W}$ is interpreted as $[\![w]\!] \in \{0,1\}^*$. Typically, the constants w denote some specific bitstrings and we could have $[\![0101]\!] = 0101$.

Next, given η, we let τ be a mapping from $\mathcal{N}$ to $\{0,1\}^\eta$. Then $[\![\cdot]\!]^\tau_\eta$ is the homomorphism from $T(\mathcal{F} \cup \mathcal{N})$ to $\{0,1\}^*$ that extends τ:

- If $w \in \mathcal{W}$, $[\![w]\!]_\eta = ([\![w]\!])^\eta$
- If $n \in \mathcal{N}$, $[\![n]\!]^\tau_\eta = \tau(n)$
- If $k, r \in \mathcal{N}$ and $u \in \mathcal{M}_0$, then $[\![\{u\}^r_k]\!]^\tau_\eta = \mathcal{E}([\![u]\!]^\tau_\eta, [\![k]\!]^\tau_\eta, [\![r]\!]^\tau_\eta)$
- If $u, v \in \mathcal{M}_0$, $[\![\langle u, v\rangle]\!]^\tau_\eta = p([\![u]\!]^\tau_\eta, [\![v]\!]^\tau_\eta)$.
- $[\![\mathsf{dec}(u,v)]\!]^\tau_\eta = \mathcal{D}([\![u]\!]^\tau_\eta, [\![v]\!]^\tau_\eta)$
- $[\![\pi_1(u)]\!]^\tau_\eta = p_1^{-1}([\![u]\!]^\tau_\eta)$
- $[\![\pi_2(u)]\!]^\tau_\eta = p_2^{-1}([\![u]\!]^\tau_\eta)$

If the names occurring in a ground term u are partitioned into $\mathcal{N}_1$ and $\mathcal{N}_2$ and τ_1 is a mapping from $\mathcal{N}_1$ to $\{0,1\}^\eta$ and τ_2 is a mapping from $\mathcal{N}_2$ to $\{0,1\}^\eta$, then $[\![u]\!]^{\tau_1}_\eta$ defines a distribution: $\mathbb{P}[x \leftarrow [\![u]\!]^{\tau_1}_\eta : x = a] = \mathbb{P}[\tau_2 : [\![u]\!]^{\tau_1 \cup \tau_2}_\eta = a]$. As a particular case, $[\![u]\!]_\eta$ is an ensemble, in which all names in u are sampled in $\{0,1\}^\eta$ (according to a distribution that is not precised here, and which may be assumed to be uniform, for simplicity).

Similarly, if $u_1, \ldots, u_k$ is a sequences of terms and τ is a partial interpretation of the names occurring in $u_1, \ldots, u_k$, $[\![u_1, \ldots, u_k]\!]^\tau_\eta$ is an ensemble: for each η, it defines a distribution on k-uples of bitstrings.

Exercice 57
Assume that the names occurring in $s_1, \ldots, s_n$ are disjoints from the names occurring in $t_1, \ldots, t_m$ then show that $[\![s_1, \ldots, s_n, t_1, \ldots, t_m]\!]_\eta = [\![s_1, \ldots, s_n]\!]_\eta \times [\![t_1, \ldots, t_m]\!]_\eta$.

Show, (using a uniform distribution of name interpretations) that it is not always true when the assumption on the disjointness of the set of names is dropped.

We may now precise the interpretation of EL. We first observe that, according to the assumptions on $\mathcal{E}, p$ and the name samples, $|[\![u]\!]_\eta^\tau|$ is independent of τ (that interprets all names occurring in u) and only depends on η. We let then $l(u, \eta) = |[\![u]\!]_\eta^\tau|$, which completes the definition of EL.

12.5 Preliminary indistinguishability results relying on the property of the encryption scheme

Lemma 12.1 *Fix an interpretation τ of the names occurring in a term u. Assume that the encryption scheme is IND-CPA. Let $u \in \mathcal{M}_0$ be such that k, r do not occur in u and are not in the domain of τ. Then:*

$$[\![\{u\}_k^r]\!]_\eta^\tau \approx [\![\{0^{l(u,\eta)}\}_k^r]\!]_\eta^\tau$$

Proof : Let $\mathcal{A}$ be a PPT machine and

$$\epsilon(\mathcal{A}, \eta) = |\mathbb{P}[x \leftarrow [\![\{u\}_k^r]\!]_\eta^\tau, R \leftarrow U : \mathcal{A}(x, 0^\eta \mid R) = 1] - \mathbb{P}[x \leftarrow [\![\{0^{l(u,\eta)}\}_k^r]\!]_\eta^\tau, R \leftarrow U : \mathcal{A}(x, 0^\eta \mid R) = 1]|$$

Consider now the oracle PPT machine $\mathcal{B}$ (which may depend on τ) such that:

1. Computes $[\![u]\!]_\eta^\tau$ and stores this in y
2. Submits the pair $(y, 0^{l(u,\eta)})$ to the oracle
3. Simulates $\mathcal{A}$ on the reply x of the oracle.

$\mathcal{B}$ runs in polynomial time,as each computation step runs in polynomial time. Furthermore, $|y| = l(u, \eta)$. Hence the machine $\mathcal{B}$ is an attacker on $IND - CPA$, whose advantage is exactly $\mathrm{Adv}(\mathcal{B}, \eta) = \epsilon(\mathcal{A}, \eta)$. Therefore $\epsilon(\mathcal{A}, \eta)$ is negligible.

This is easily generalized to sequences of ciphertexts:

Lemma 12.2 *Let $u_1, \ldots, u_m \in \mathcal{M}_0$. Fix an interpretation τ of the names occurring in $u_1, \ldots, u_m$. Assume that the encryption scheme is IND-CPA. If the names $k, r_1, \ldots, r_m$ are distinct and do not occur in $u_1, \ldots, u_m$, then*

$$[\![\{u_1\}_k^{r_1}, \ldots, \{u_m\}_k^{r_m}]\!]_\eta^\tau \approx [\![\{0^{l(u_1,\eta)}\}_k^{r_1}, \ldots, \{0^{l(u_m,\eta)}\}_k^{r_m}]\!]_\eta^\tau$$

Exercice 58
Complete the proof of the above lemma, using the same ideas as in the proof of the lemma 12.1.

The condition on the occurrences of k, r in u is necessary for lemma 12.1, as shown by the following

Exercice 59
Assume that there exists at least one IND-CPA symmetric encryption scheme. Construct another IND-CPA encryption scheme such that $[\![\{k\}_k^r]\!]_\eta \not\approx [\![\{0^\eta\}_k^r]\!]_\eta$.

We may, however, relax a little bit the assumptions:

Exercice 60
Assume that the encryption scheme is IND-CPA and that $u, v \in \mathcal{M}_0$ are such that $l(u,\eta) = l(v,\eta)$ and, for any name n not occuring in u, v, $[\![\langle u, n\rangle]\!]_\eta \approx [\![\langle u, k\rangle]\!]_\eta$ and $[\![\langle v, n\rangle]\!]_\eta \approx [\![\langle v, k\rangle]\!]_\eta$. Prove then $[\![\{u\}_k^r]\!]_\eta \approx [\![\{v\}_k^r]\!]_\eta$ for any name r not occurring in u, v, k.

Give an example of such u, v that do contain occurrences of k.

12.6 Proof of soundness of static equivalence: a special case

As we saw above, we need some assumptions on the sequence of terms, ruling out situations such as a key encrypting itself.

Given a sequence of terms $s_1, \dots, s_n$, we define the relation $k >_{s_1,\dots,s_n} k'$ between names, as the least transitive relation such that:

If k_1 occurs in u and there is an index i and a subterm $\{u\}_{k_2}^r$ of s_i, then $k_2 >_{s_1,\dots,s_n} k_1$

A *random seed* is a name that is used as a third argument of an encryption symbol.

Definition 12.6 *A* valid frame *(resp.* valid term sequence*) is a frame (resp. term sequence) $\nu\overline{n}.\{x_1 \mapsto s_1, \dots, x_n \mapsto s_n\}$ (resp. $s_1, \dots, s_n$), such that*

1. $s_1, \dots, s_n \in \mathcal{M}_0$
2. $\overline{n}$ *is the set of names occurring in* $s_1, \dots, s_n$
3. $\geq_{s_1,\dots,s_n}$ *is an ordering.*
4. *each random seed is used only once in* $s_1, \dots, s_n$

Example 12.1 *The following are not valid term sequences*

1. $\{k\}_k^r$
2. $\{\{k_1\}_{k_2}^{r_1}\}_{k_3}^{r_2}, \{\{k_2\}_{k_1}^{r_3}\}_{k_3}^{r_4}$
3. $\{\{k_1\}_{k_2}^{r_1}\}_{k_3}^{r_2}, \{\{k_1\}_{k_3}^{r_3}\}_{k_2}^{r_4}$
4. $\{\{k_1\}_{k_2}^{r_1}\}_{k_2}^{r_2}, \{\{k_2\}_{k_3}^{r_3}\}_{k_3}^{r_4}$

The main restriction imposed by the first condition is the use of names as keys: only atomic keys are considered.

The second condition is not a restriction: we may always bind all names and disclose explicitly the names that are supposed to be available to the attacker.

The third condition is a real (strong) restriction. Some restriction that rules out key cycles is necessary (with the current state of the art). The above condition rules out terms $\{\{u\}_k^{r_1}\}_k^{r_2}$ in the frames. We will see in the section 12.7, that the condition can be slightly relaxed, allowing for such terms in the frames.

The last condition forbids several occurrences of the same ciphertext in the frames. This condition will also be relaxed in the section 12.7.

Theorem 12.1 *Let $\nu\overline{n}.\{x_1 \mapsto s_1, \dots x_n \mapsto s_n\}$ and $\nu\overline{n'}.\{x_1 \mapsto t_1, \dots, x_n \mapsto t_n\}$ be two valid frames.*

$$\nu\overline{n}.\{x_1 \mapsto s_1, \dots x_n \mapsto s_n\} \sim \nu\overline{n'}.\{x_1 \mapsto t_1, \dots, x_n \mapsto t_n\} \quad \Rightarrow \quad [\![s_1, \dots, s_n]\!]_\eta \approx [\![t_1, \dots, t_n]\!]_\eta$$

In other words, if the two frames are statically equivalent, then they are computationally indistinguishable.

In what follows, we drop the name binders and the variables, since all names are bound and the variable symbols can be inferred from the context. Furthermore, if P is a predicate symbol and u, v are valid recipes (terms that do not use any names in our case), we write $s_1, \ldots, s_n \models P(u, v)$ insteand of $(u\{x_1 \mapsto s_1, \ldots, x_n \mapsto s_n\}, v\{x_1 \mapsto s_1, \ldots, x_n \mapsto s_n\}) \in P^I$.

Proof : We use an induction on $|s_1| + \cdots + |s_n| + |t_1| + \cdots |t_n|$ where $|s|$ is the number of function symbols and names appearing in s (we do not count the constants). In each case, we leave to the reader the verification that the induction hypothesis is applied to valid sequences indeed.

Base case: $s_1, \ldots, s_n$ and $t_1, \ldots, t_n$ are constants. Since there is no names in the frames, s_i is a valid recipe: $s_1, \ldots, s_n \models EQ(x_i, s_i)$ and therefore $t_1, \ldots, t_n \models EQ(x_i, s_i)$. It follows that, for every i, $s_i = t_i$, hence the indistinguishability of the two sequences.

Induction case: We successively investigate cases. In each case, we assume that the previous cases do not apply.

1. If one of the two sequences contains a pair.
   w.l.o.g., assume $s_1 = \langle s_{11}, s_{12} \rangle$. Then $s_1, \ldots, s_n \models M(\pi_1(x_1))$, hence $t_1, \ldots, t_n \models M(\pi_1(x_1))$. This implies that there are terms $t_{11}, t_{12} \in \mathcal{M}_0$ such that $t_1 = \mathsf{pair} t_{11} t_{12}$.
   Then $s_{11}, s_{12}, s_2, \ldots, s_n \sim t_{11}, t_{12}, t_2, \ldots, t_n$:

$$\begin{array}{lll} s_{11}, s_{12}, s_2, \ldots, s_n \models P(u, v) & \text{iff} & s_1, s_2, \ldots, s_n \models P(u, v)\{x_{11} \mapsto \pi_1(x_1), x_{12} \mapsto \pi_2(x_1)\} \\ & \text{iff} & t_1, t_2, \ldots, t_n \models P(u, v)\{x_{11} \mapsto \pi_1(x_1), x_{12} \mapsto \pi_2(x_1)\} \\ & \text{iff} & t_{11}, t_{12}, t_2, \ldots, t_n \models P(u, v) \end{array}$$

   By induction hypothesis, $[\![s_{11}, s_{12}, s_2, \ldots, s_n]\!]_\eta \approx [\![t_{11}, t_{12}, t_2, \ldots, t_n]\!]$.
   We use then the following exercise:

   Exercice 61
   Let f be a function symbol, whose computational interpretation is a PPT function $[\![f]\!]$. Assume $[\![s_1, \ldots s_p, s_{p+1} \ldots, s_n]\!]_\eta \approx [\![t_1, \ldots, t_p, t_{p+1}, \ldots, t_n]\!]_\eta$ and $r \notin fn(s_1, \ldots, s_n, t_1, \ldots, t_n)$. Prove that $[\![f(s_1, \ldots, s_p \mid r), s_{p+1}, \ldots, s_n]\!]_\eta \approx [\![f(t_1, \ldots, t_p \mid r), t_{p+1}, \ldots, t_n]\!]_\eta$ (r is assumed to be drawn according to a polynomial distribution).

   With the pairing function for f and conclude that $[\![s_1, \ldots, s_n]\!]_\eta \approx [\![t_1, \ldots, t_n]\!]_\eta$.

2. If $s_i = s_j$(resp. $t_i = t_j$) for some $i \neq j$ and s_i is not a constant. Then $s_1, \ldots, s_n \models EQ(x_i, x_j)$, hence $t_1, \ldots, t_n \models EQ(x_i, x_j)$, which implies $t_i = t_j$. Then $s_1, \ldots, s_{i-1}, s_{i+1}, \ldots, s_n \sim t_1, \ldots, t_{i-1}, t_{i+1}, \ldots, t_n$ and, by induction hypothesis, $[\![s_1, \ldots, s_{i-1}, s_{i+1}, \ldots, s_n]\!]_\eta \approx [\![t_1, \ldots, t_{i-1}, t_{i+1}, \ldots, t_n]\!]_\eta$, hence the result.

3. If some $s_i = \{u_i\}_k^{r_i}$ (or some t_i; this case is symmetric) where u_i is not a constant and k is maximal w.r.t. $\geq_{s_1, \ldots, s_n}$ and $k \notin \{s_1, \ldots, s_n\}$. After possibly renumbering the terms, let $s_1 = \{u_1\}_k^{r_1}, \ldots s_p = \{u_p\}_k^{r_p}$ and $s_{p+1}, \ldots, s_n$ are not encryptions with k.

   k does not occur in $u_1, \ldots, u_p, s_{p+1}, \ldots, s_n$ since every s_i is either a name (different from k), a constant, or a ciphertext (thanks to step 1) and, in the latter case, k does not occur in s_i by maximality of k.

   Let $\sigma = \{x_1 \mapsto s_1, \ldots, x_n \mapsto s_n\}$ and $\sigma' = \{x_1 \mapsto s'_1, \ldots, x_n \mapsto s'_n\}$ and let ρ be the replacement of $s_1, \ldots, s_p$ with $\{0^{l(u_1, \eta)}\}_k^{r_1}, \ldots, \{0^{l(u_p, \eta)}\}_k^{r_p}$ respectively. We observe first that, for any recipe u, $\rho(u\sigma\downarrow) = u\sigma'\downarrow$ by consistency of use of the random seeds.

If $s_1, \ldots, s_n \models P(u, v)$, $(u\sigma\!\downarrow, v\sigma\!\downarrow) \in P^I$. For $P \in \{M, EQ, EK, EL\}$, $(u_1, u_2) \in P^I$ iff $(\rho(u_1), \rho(u_2)) \in P^I$ (for instance, when $P = EL$, this is thanks to the presevation of plaintexts lengths by ρ). Hence

$$\begin{array}{lll} s_1. \ldots, s_n \models P(u, v) & \text{iff} & (\rho(u\sigma\!\downarrow), \rho(v\sigma\!\downarrow)) \in P^I \\ & \text{iff} & (u\sigma'\!\downarrow, v\sigma'\!\downarrow) \in P^I \\ & \text{iff} & s'_1, \ldots, s'_n \models P(u, v) \end{array}$$

$s_1, \ldots, s_n \sim t_1, \ldots, t_n$ therefore implies $s'_1, \ldots, s'_n \sim t_1, \ldots, t_n$. Since, by assumption, at least one s_i is such that $s_i\{u_i\}_k^{r_i}$ where u_i is not a constant and $s'_i = \{0^{l(u_i,\eta)}\}_k^{r_i}$ has a sctritly smaller size, we may apply the induction hypothesis:

$$[\![s'_1, \ldots, s'_n]\!]_\eta \approx [\![t_1, \ldots, t_n]\!]_\eta$$

Then, thanks to our assumptions (validity of the sequences) and lemma 12.2, $[\![s_1, \ldots, s_n]\!]_\eta \approx [\![s'_1, \ldots, s'_n]\!]_\eta$.
We conclude $[\![s_1, \ldots, s_n]\!]_\eta \approx [\![t_1, \ldots, t_n]\!]$.

4. If there are two indices i, j such that $s_i = \{u_i\}_{s_j}^{r_i}$. Assume w.l.o.g $i = 1$. $s_1, \ldots, s_n \models M(\mathsf{dec}(x_1, x_j))$ implies $t_1, \ldots, t_n \models M(\mathsf{dec}(x_1, x_j))$, hence $t_1 = \{v_1\}_{t_j}^{r'_1}$ for some v_1.
We claim that $u_1, s_2, \ldots, s_n \sim v_1.t_2, \ldots, t_n$:

$$\begin{array}{lll} u_1, s_2 \ldots s_n \models P(w_1, w_2) & \text{iff} & s_1, s_2 \ldots, s_n \models P(w_1\{x_1 \mapsto \mathsf{dec}(x_1, x_j)\}, w_2\{x_1 \mapsto \mathsf{dec}(x_1, x_j)\}) \\ & \text{iff} & t_1, \ldots, t_n \models P(w_1\{x_1 \mapsto \mathsf{dec}(x_1, x_j)\}, w_2\{x_1 \mapsto \mathsf{dec}(x_1, x_j)\}) \\ & \text{iff} & v_1, t_2, \ldots, t_n \models P(w_1, w_2) \end{array}$$

By induction hypothesis, $[\![u_1, s_2, \ldots, s_n]\!]_\eta \approx [\![v_1, t_2, \ldots, t_n]\!]_\eta$. By exercise 61 *and since we assumed that each random seed is used only once*, we conclude $[\![s_1, \ldots, s_n]\!]_\eta \approx [\![t_1, \ldots, t_n]\!]_\eta$. (We sill see in the next section how to modify this part, to avoid the assumption on the unique use of random seeds).

5. Now we have only to consider sequences $s_1, \ldots, s_n, t_1, \ldots, t_n$ that consist of encryptions of constants, names that are not used as encryption keys, and constants. If one of the sequence contains at least one ciphertext, assume w.l.o.g. that this is s_1: $s_1 = \{c_1\}_k^{r_1}$. Let $s_1, \ldots s_p$ be all ciphertexts in the sequence $s_1, \ldots, s_n$, whose encryption key is k: $\mathrm{v}s_i = \{c_i\}_k^{r_i}$ for $1 \leq i \leq p$.
For every $1 \leq i, j \leq p$, $s_1, \ldots, s_n \models EK(x_i, x_j)$, hence $t_1, \ldots, t_p \models EK(x_i, x_j)$: for $i = 1, ..., p, t_i = \{c'_i\}_{k'}^{r'_i}$ for some constants c'_i. Furthermore, thanks to EL, for every $i = 1, ..., p$, $l(c_i, \eta) = l(c'_i, \eta)$. It follows that $[\![s_1, \ldots, s_p]\!]_\eta \approx [\![t_1, \ldots, t_p]\!]_\eta$ by lemma 12.2.
The key k does not occur in the sequence $s_{p+1}, ldots, s_n$ and, by symmetry, the key k' does not occur in the sequence $t_{p+1}, \ldots, t_n$. Then, by consistency of use of random seeds,
$$[\![s_1, \ldots, s_n]\!]_\eta = [\![s_1, \ldots, s_p]\!]_\eta \times [\![s_{p+1}, \ldots, s_n]\!]_\eta$$
and
$$[\![t_1, \ldots, t_p]\!]_\eta \times [\![t_{p+1}, \ldots, t_n]\!]_\eta = [\![t_1, \ldots, t_n]\!]_\eta$$
By induction hypothesis, and since $s_{p+1}, \ldots, s_n \sim t_{p+1}, \ldots, t_n$, $[\![s_{p+1}, \ldots, s_n]\!]_\eta \approx [\![t_{p+1}, \ldots, t_n]\!]_\eta$. It follows that
$$[\![s_1, \ldots, s_n]\!]_\eta = [\![s_1, \ldots, s_p]\!]_\eta \times [\![s_{p+1}, \ldots, s_n]\!]_\eta \approx [\![t_1, \ldots, t_p]\!]_\eta \times [\![t_{p+1}, \ldots, t_n]\!]_\eta = [\![t_1, \ldots, t_n]\!]_\eta$$

6. We are left to a case where $s_1, \ldots, s_n$, (and $t_1, \ldots t_n$) are sequences of distinct names and constants. If there is at least one name in the sequence, say s_1, then t_1 must also be a name (otherwise $EQ(x_1, t_1)$ would be satisfied in the second sequence and not in the first one). By induction hypothesis, $[\![s_2, \ldots, s_n]\!]_\eta \approx [\![t_2, \ldots, t_n]\!]_\eta$ and then

$$[\![s_1, \ldots, s_n]\!]_\eta = [\![s_1]\!]_\eta \times [\![s_2, \ldots, s_n]\!]_\eta \approx [\![t_1]\!]_\eta \times [\![t_2, \ldots, t_n]\!]_\eta = [\![t_1, \ldots, t_n]\!]_\eta$$

Exercice 62
Give an example showing that the above proof does not work when a random seed may occur twice in a valid frame.

12.7 The proof in the general case

In this section, we prove exactly the same result as in the previous section, however relaxing two assumptions.

First, we relax the condition on keys occurences, in order to allow for instance terms $\{\{u\}_k^r\}_k^{r'}$ in the sequence. We define the relation $\sqsubseteq$ on terms (read $s \sqsubseteq t$ as "s occurs as plaintext in t") as the least symmetric and transitive relation such that:

- If $u \sqsubseteq u_1$ or $u \sqsubseteq u_2$, then $u \sqsubseteq \langle u_1, u_2 \rangle$.
- If $u \sqsubseteq v$ then $u \sqsubseteq \{v\}_k^r$.

Now, $\gg_{s_1 \ldots, s_n}$ is redefined as the (more general) least transitive relation such that, $k_2 \gg_{s_1, \ldots, s_n} l_1$ whenever there is a subterm $\{u\}_{k_2}^r$ of some s_i such that $k_1 \sqsubseteq u$.

Definition 12.7 *A sequence of terms $s_1, \ldots, s_n$ in $\mathcal{M}_0$* has no key cycle *if $\geqq_{s_1, \ldots, s_n}$ is an ordering. Dually, if there is a name k such that $k \gg_{s_1, \ldots, s_n} k$, then $s_1, \ldots, s_n$* contains a key-cycle.

Key cycles are defined now according to this new ordering.

Example 12.2 *1. $\{k\}_k^r$ contains a key cycle*

2. $\{\{k_1\}_{k_2}^{r_1}\}_{k_3}^{r_2}, \{\{k_2\}_{k_1}^{r_3}\}_{k_3}^{r_4}$ contains a key-cycle

3. $\{\{k_1\}_{k_2}^{r_1}\}_{k_3}^{r_2}, \{\{k_1\}_{k_3}^{r_3}\}_{k_2}^{r_4}$ does not contains a key-cycle

4. $\{\{k_1\}_{k_2}^{r_1}\}_{k_2}^{r_2}, \{\{k_2\}_{k_3}^{r_3}\}_{k_3}^{r_4}$ does not contains a key-cycle

Second, we relax the conditions on random seeds, allowing several copies of the same ciphertext:

Definition 12.8 *A sequence $s_1, \ldots, s_n$ of terms* uses the random seeds in a consistent way *if*

1. *Any random seed occurring in $s_1, \ldots, s_n$, only occurs in $s_1, \ldots, s_n$ as the third argument of an encryption*
2. *If $\{u_1\}_{k_1}^r$ and $\{u_2\}_{k_2}^r$ are two subterms of $s_1, \ldots, s_n$, then $u_1 = u_2$ and $k_1 = k_2$.*

Definition 12.9 *A sequence of terms (resp. a frame) $s_1, \ldots, s_n$ is* weakly valid *if*

- *it has no key cycle*
- *the sandom seeds are used in a consistent way*

Example 12.3 *$\{\{u\}_k^r\}_k^{r'}, \{\{u\}_k^r\}_{k'}^{r''}, \{k'\}_k^{r'''}$ is a weakly valid sequence, with $k \gg k'$.*

Definition 12.10 *A key k is* deducible *from a frame ϕ, if there is a recipe u such that $u\sigma_\phi\downarrow = k$.*

Now we can generalize theorem 12.1 to:

Theorem 12.2 *Let $\nu\overline{n}.\{x_1 \mapsto s_1, \dots x_n \mapsto s_n\}$ and $\nu\overline{n'}.\{x_1 \mapsto t_1, \dots, x_n \mapsto t_n\}$ be two weakly valid frames.*

$$\nu\overline{n}.\{x_1 \mapsto s_1, \dots x_n \mapsto s_n\} \sim \nu\overline{n'}.\{x_1 \mapsto t_1, \dots, x_n \mapsto t_n\} \quad \Rightarrow \quad [\![s_1, \dots, s_n]\!]_\eta \approx [\![t_1, \dots, t_n]\!]_\eta$$

The proof is basically the same as the proof of theorem 12.1. The only difference lies in points 3 and 4: at step 3, we need to replace ciphertexts $\{u\}_k^r$ with $\{0^{l(u,\eta)}\}_k^r$ *at every position.* And, at step 4, we have to show that maximal keys for one sequence correspond to maximal keys for the other sequence.

The following two lemmas prepare these steps.

Again, we confuse the frames and the term sequences, as all names are assumed to be bound.

Lemma 12.3 *Let $v_1, \dots, v_m$ be a valid sequence of terms, let $u \in \mathcal{M}_0$, k, r be names such that $k \not\sqsubseteq u, v_1, \dots, v_m$, and r occurs in $u, v_1, \dots, v_m$ only as a random seed in subterms $\{u\}_k^r$. Then*

$$[\![v_1, \dots, v_m]\!]_\eta \approx [\![v_1[\{u\}_k^r \mapsto \{0^{l(u,\eta)}\}_k^r], \dots, v_m[\{u\}_k^r \mapsto \{0^{l(u,\eta)}\}_k^r]]\!]_\eta$$

and

$$(\nu\overline{n})v_1, \dots, v_m \sim (\nu\overline{n})v_1[\{u\}_k^r \mapsto \{0^{l(u,\eta)}\}_k^r], \dots, v_m[\{u\}_k^r \mapsto \{0^{l(u,\eta)}\}_k^r]$$

Proof : We prove first the first claim of the lemma.

Let τ be a partial assigment of all names, except k and the random seeds s that are used in ciphertexts $\{w\}_k^s$ occurring in $v_1, \dots v_m$. We prove actually

$$[\![v_1, \dots, v_m]\!]_\eta^\tau \approx [\![v_1[\{u\}_k^r \mapsto \{0^{l(u,\eta)}\}_k^r], \dots, v_m[\{u\}_k^r \mapsto \{0^{l(u,\eta)}\}_k^r]]\!]_\eta^\tau$$

Assume that $\mathcal{A}$ can distinguish the two above sequences with an advantage ϵ:

$$\begin{aligned}\epsilon = |&\mathbb{P}[k, R, r_1, \dots, r_n : \mathcal{A}([\![v_1, \dots, v_m]\!]_\eta^\tau | R) = 1] \\ &-\mathbb{P}[k, R, r_1, \dots, r_n : \mathcal{A}([\![v_1[\{u\}_k^r \mapsto \{0^{l(u,\eta)}\}_k^r], \dots, v_m[\{u\}_k^r \mapsto \{0^{l(u,\eta)}\}_k^r]]\!]_\eta^\tau | R) = 1]|\end{aligned}$$

We construct as follows an adversary $\mathcal{B}$ on IND-CPA: let $w_1, \dots, w_n$ be the set of terms w such that $w \sqsubseteq v_1, \dots, v_m$. $w_1, \dots, w_n$ are ordered in such a way that $i < j$ whenever w_i is a subterm of w_j.

1. $\mathcal{B}$ stores in its memory a table associating the terms w_i with their computational interpretations: For $i = 1$ to n, $\mathcal{B}$
   (a) if w_i is a constant or a name (which is then in he domain of τ), $\mathcal{B}$ stores in its table $[\![w_i]\!]_\eta^\tau$
   (b) if $w_i = \langle w_j, w_k \rangle$, then $\mathcal{B}$ retrieves the values of $[\![w_j]\!]_\eta^\tau$, $[\![w_k]\!]_\eta^\tau$, that are stored in its table, computes teh pair of them and stores the result in the table
   (c) if $w_i = \{w_j\}_{k'}^{r'}$ where $k' \neq k$, then $\mathcal{B}$ retreives $[\![w_j]\!]_\eta^\tau$ and computes $[\![w_i]\!]_\eta^\tau$ and stores the result
   (d) if $w_i = \{w_j\}_k^{r'}$, with $r \neq r'$, then $\mathcal{B}$ retrieves $[\![w_j]\!]_\eta^\tau$ from its table, queries the encryption oracle with $([\![w_j]\!]_\eta^\tau, [\![w_j]\!]_\eta^\tau)$ and stores the result.

(e) if $w_i = \{u\}_k^r$, then $\mathcal{B}$ retrieves $[\![u]\!]_\eta^\tau$ from its table and queries the encryption oracle with $([\![u]\!]_\eta^\tau, 0^{l(u,\eta)})$ and strores the result

At the end of this (PTIME) procedure, $\mathcal{B}$ has stored in its table all the computational interpretations of the subterms of $v_1, \ldots, v_m$, if it was interacting with the left-oracle. Otherwise, the stores contains the computational interpretation of the subterms of $v_1[\{u\}_k^r \mapsto \{0^{l(u,\eta)}\}_k^r], \ldots, v_m[\{u\}_k^r \mapsto \{0^{l(u,\eta)}\}_k^r], u$.

2. $\mathcal{B}$ simulates $\mathcal{A}$ with the inputs corresponding to the values stored at $v_1, \ldots, v_m$ locations. It produces the same output as $\mathcal{A}$.

The advantage of $\mathcal{B}$ is ϵ: if $\mathcal{A}$ distinguishes the two sequences of terms with non negligible probability, then $\mathcal{B}$ breaks IND-CPA.

Now, remains to prove the second claim.

We let ρ be the replacement of $\{u\}_k^r$ with $\{0^{l(u,\eta)}\}_k^r$ and, for every i, $v'_i = \rho(v_i)$. The consistency of the use of random seeds implies that ρ is a bijection, whose inverse is ρ'. We claim that $v_1, \ldots, v_n \sim v'_1, \ldots, v'_n$. To see this, let $\sigma = \{x_1 \mapsto v_1, \ldots, x_n \mapsto v_n\}$ and $\sigma' = \{x_1 \mapsto v'_1, \ldots, x_n \mapsto v'_n\}$. Note first that there is no recipe w such that $w\sigma \downarrow = k$, because $k \not\sqsubseteq v_i$ for every i (and since, for every rewrite rule $l \to r$, $r\sigma = k$ implies that $r \sqsubseteq l$).

By induction on m, we prove that, for any recipe t, $t\sigma \xrightarrow{m} s$ iff $t\sigma' \xrightarrow{m} \rho(s)$. If $m = 0$ this is because none of the random seeds of $v_1, \ldots, v_n$ is occurring in t, hence $\rho(t\sigma) = t\sigma'$. Otherwise, there is a position p in $t\sigma$, a rewrite rule $l \to r$ and a subsitution θ such that $t\sigma|_p = l\theta$ and $t\sigma[r\theta]_p \xrightarrow{m-1} s$. If $l = \pi_i(\langle x_1, x_2\rangle)$ and $r = x_i$, $\rho(l\theta) = \pi_i(\langle \rho(x_1\sigma), \rho(x_2\sigma)\rangle)$. It follows that $t\sigma'|_p = l\theta'$ and $\rho(t\sigma[r\theta]_p) = t\sigma'[r\theta']_p$ and we may apply the induction hypothesis. If $l = \mathsf{dec}(\{x\}_{k_1}^{r_1}, k_1)$, then $k_1 \neq k$ and $\rho(l\theta) = \mathsf{dec}(\{\rho(x\theta)\}_{k_1}^{r_1}, k_1)$. Again $\rho(t\sigma[x\theta]_p) = t\sigma'[x\theta']_p$ and $t\sigma' \to \rho(u\sigma[r\theta]_p)$: we may apply the induction hypothesis.

Now, $v_1, \ldots, v_n \models P(t_1, t_2)$ iff $(t_1\sigma\downarrow, t_2\sigma\downarrow) \in P^I$. As already observed, for $P \in \{M, EQ, EK, EL\}$, P^I is invariant by ρ (and ρ'): $(t_1\sigma\downarrow, t_2\sigma\downarrow) \in P^I$ iff $(\rho(t_1\sigma\downarrow), \rho(t_2\sigma\downarrow)) \in P^I$. Thanks to what we proved above, $\rho(t_i\sigma\downarrow) = t_i\sigma'\downarrow$, hence $v_1, \ldots, v_n \models P(t_1, t_2)$ iff $(t_1\sigma'\downarrow, t_2\sigma'\downarrow) \in P^I$ iff $v'_1, \ldots, v'_n \models P(t_1, t_2)$. This concludes the proof of the second claim.

Lemma 12.4 *Assume that $(s_1, \ldots, s_n), (t_1, \ldots, t_n)$ are weakly valid term sequences such that*

- $(s_1, \ldots, s_n) \sim (t_1, \ldots, t_n)$
- *for every $t = \{u\}_k^r \in \mathcal{M}_0$, if $u \notin \mathcal{W}$ and t occurs in some s_i (resp. some t_i), then there is a recipe v such that $v\{x_1 \mapsto s_1, \ldots, x_n \mapsto s_n\}\downarrow = k$ (resp. $v\{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}\downarrow = k$).*

Then, for every $j_1 \neq j_2$,

$$s_{j_1} \notin \mathcal{W} \text{ and } s_{j_1} \sqsubseteq s_{j_2} \text{ implies } t_{j_1} \notin \mathcal{W} \text{ and } t_{j_1} \sqsubseteq t_{j_2}.$$

Proof : We prove the lemma by induction on $|s_{j_2}|$.

In the base case, $|s_{j_2}| = 1$, then $s_{j_1} = s_{j_2}$. Using EQ, it follows that $t_{j_1} = t_{j_2}$.

Now, for the induction step. If $s_{j_2} = \left\langle s'_{j_2}, s''_{j_2} \right\rangle$, using M, t_{j_2} must also be a pair $t_{j_2} = \left\langle t'_{j_2}, t''_{j_2} \right\rangle$ and

$$(s_1, \ldots, s_{j_2-1}, s'_{j_2}, s''_{j_2}, s_{j_2+1}, \ldots, s_n) \sim (t_1, \ldots, t_{j_2-1}, t'_{j_2}, t''_{j_2}, t_{j_2+1}, \ldots, t_n)$$

Moreover, either $s_{j_1} \sqsubseteq s'_{j_2}$ or else $s_{j_1} \sqsubseteq s''_{j_2}$. By induction hypothesis, t_{j_1} is a name and $t_{j_1} \sqsubseteq t'_{j_2}$ or $t_{j_1} \sqsubseteq t''_{j_2}$. In any case, $t_{j_1} \sqsubseteq \left\langle t'_{j_2}, t''_{j_2} \right\rangle$.

If $s_{j_2} = \{s'_{j_2}\}_{k_{j_2}}^{r_{j_2}}$, then $s_{j_1} \sqsubseteq s'_{j_2}$ and $s_{j_1} \neq k_{j_2}$ and $s_{j_1} \neq r_{j_2}$ (because the sequences are weakly valid). In particular, $s'_{j_2} \notin \mathcal{W}$, hence, by hypothesis, there is a recipe u such that $u\{x_1 \mapsto s_1, \ldots, x_n \mapsto s_n\}\downarrow = k_{j_2}$. Using M again, $u\{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}\downarrow = k'_{j_2}$ is a key and, considering the recipe $\mathsf{dec}(x_{j_2}, u)$, t_{j_2} must be $\{t'_{j_2}\}_{k'_{j_2}}^{r'_{j_2}}$ for some t'_{j_2}. Now, $s_1, \ldots, s_{j_2-1}, s'_{j_2}, s_{j_2+1}, \ldots, s_n) \sim (t_1, \ldots, t_{j_2-1}, t'_{j_2}, t_{j_2+1}, \ldots, t_n)$ and, by induction hypothesis, $s_{j_1} \sqsubseteq s'_{j_2}$ implies $t_{j_1} \sqsubseteq t'_{j_2}$. It follows that $t_{j_1} \sqsubseteq t_{j_2}$.

Proof of the theorem 12.2 : Again, we use an induction on $|s_1| + \cdots + |s_n| + |t_1| + \cdots + |t_n|$ (where constants are not counted in $|s_i|, |t_i|$). The base case and cases 1,2 are exactly the same as in the proof of theorem 12.1.

Let us assume now that the two sequences only consists of ciphertexts, constants and names.

- If in one of the two sequences, there is a subterm $\{u\}_k^r$ such that u is not a constant and k is not deducible.

  Assume for instance that such a term occurs as a subterm in the sequence $s_1, \ldots, s_n$. Consider a key k, which is maximal w.r.t. $\underline{\gg}_{s_1,\ldots,s_n}$ among the keys that are not deducible from $s_1, \ldots, s_n$ and that encrypt at least one non-constant term.

  We claim that $k \not\sqsubseteq s_1, \ldots, s_n$. If it was the case, say $k \sqsubseteq s_1$, we prove, by induction on s_1, that either k is deducible from the sequence, or else there is a non-deducible key k' such that $k' \gg k$. In the base case, $s_1 = k$ is deducible. If $s_1 = \langle s_{11}, s_{12}\rangle$, then either $k \sqsubseteq s_{11}$ or else $k \sqsubseteq s_{12}$ and, by induction hypothesis, k is deducible from $s_{11}, s_{12}, s_2, \ldots, s_n$ or else there is a non deducible key k' such that $k' \gg_{s_{11},s_{12},s_2,\ldots,s_n} k$. In the first case, k is deducible from $s_1, \ldots, s_n$ (replacing x_{11} with $\pi_1(x_1)$ and x_{12} with $\pi_2(x_1)$ in the recipe) and, in the second case, $k' \gg_{s_1,\ldots,s_n} k$. Now, if $s_1 = \{s_{11}\}_{k'}^r$, then, by weak validity of the sequence, $k \neq k'$. Then $k' \gg_{s_1,\ldots,s_n} k$. Either k' is not deducible, and we are done, or else, there is a recipe u such that $u\{x_1 \mapsto s_1, \ldots, x_n \mapsto s_n\}\downarrow = k'$. Then $\mathsf{dec}(x_1, u)$ is a recipe yielding s_{11}. Since, in addition, $k \sqsubseteq s_{11}$, by induction hypothesis, either k is deducible from $s_{11}, s_2, \ldots, s_n$, hence from $s_1, \ldots, s_n$ (replacing x_1 with $\mathsf{dec}(x_1, u)$ in the recipe) or there is a key k', that is not deducible from $s_{11}, \ldots, s_n$, such that $k' \gg_{s_{11},s_2,\ldots,s_n} k$. In the latter case, k' is neither deducible from $s_1, \ldots, s_n$ and $k' \gg_{s_1,\ldots,s_n} k$, which concludes the proof of our claim.

  Then, we may use the lemma 12.3: let, for every i, $s'_i = s_i\{\{u\}_k^r \mapsto \{0^{l(u,\eta)}\}_k^r\}$ where $\{u\}_k^r$ is any subterm of the sequence $s_1, \ldots, s_n$ such that u is not a constant. (There is such a term by assumption). By lemma 12.3,

  $$[\![s_1, \ldots, s_n]\!]_\eta \approx [\![s'_1, \ldots, s'_n]\!]_\eta$$

  and $s_1, \ldots, s_n \sim s'_1, \ldots, s'_n$. It follows that $s'_1, \ldots, s'_n \sim t_1, \ldots, t_n$, hence, by induction hypothesis, $[\![s'_1, \ldots, s'_n]\!]_\eta \approx [\![t_1, \ldots, t_n]\!]_\eta$ and, since $[\![s'_1, \ldots, s'_n]\!]_\eta \approx [\![s_1, \ldots, s_n]\!]_\eta$, we get the desired result: $[\![s_1, \ldots, s_n]\!]_\eta \approx [\![t_1, \ldots, t_n]\!]_\eta$.

- We assume now that all terms of both sequences are either names, constants or ciphertexts and that every ciphertext $\{u\}_k^r$ occurring in the sequences is such that either k is deducible or else u is a constant. Furthermore, none of the sequences contain two identical terms.

  If one of the sequences contains a ciphertext $\{u\}_k^r$ such that k is deducible: $s_i = \{u_i\}_k^{r_i}$ and there is a recipe v such that $v\sigma\downarrow = k$ where $\sigma = \{x_1 \mapsto s_1, \ldots, x_n \mapsto s_n\}$.

  After a possible renumbering, let $s_1 = \{u_1\}_k^{r_1}$ be a term in the sequence, whose encryption key k is deducible and which is maximal w.r.t. $\sqsubseteq$. This implies that s_1 has no other occurrence in the sequence (because the plaintext of encryptions with non-deducible keys are constant).

Since $s_1, \ldots, s_n \models M(\mathsf{dec}(x_1, v))$, we must have $t_1, \ldots, t_n \models M(\mathsf{dec}(x_1, v))$: $t_1 = \{v_1\}_{k'}^{r'_1}$ and k' is a deducible key (using the recipe v). By lemma 12.4, t_1 is also maximal w.r.t. $\sqsubseteq$, and, for the same reason as above, has no other occurrence in the sequence.

As before, $u_1, s_2, \ldots, s_n \sim v_1, t_2, \ldots, t_n$. By induction hypothesis, we therefore have $[\![u_1, s_2, \ldots, s_n]\!]_\eta \approx [\![v_1, t_2, \ldots, t_n]\!]_\eta$. Then, by maximality w.r.t. $\sqsubseteq$ of s_1, t_1 and by the consistency of use of random numbers, r has no other occurrence in the sequence $s_1, \ldots, s_n$ and r' has no other occurrence in the sequence $t_1, \ldots, t_n$. From $[\![u_1, s_2, s_2, s_3, \ldots, s_n]\!]_\eta \approx [\![v_1, t_2, t_2, t_3, \ldots, t_n]\!]_\eta$ and exercice 61, it follows $[\![s_1, \ldots, s_n]\!]_\eta \approx [\![t_1, \ldots, t_n]\!]_\eta$.

Now, we can proceed with the last cases of the proof, as in the proof of theorem 12.1 and conclude.

Exercice 63
We say that frame $(\nu\overline{n})\ s_1, \ldots, s_n$ is *transparent* if:

- for every ciphertext $\{u\}_k^r$ that occurs in $s_1, \ldots, s_n$, either u is constant or else k is deducible.
- The random seeds are used in a consistent way

Given a transparent frame $\phi = (\nu\overline{n}).s_1, \ldots, s_n$, we define the *flatened* frame $F(\phi)$ by induction as follows:

- If there is an index i such that $s_i = \langle s_{i1}, s_{i2}\rangle$, then $F(\phi) = F((\nu\overline{n}).s_1, \ldots, s_{i-1}, s_{i1}, s_{i2}, s_{i+1}, \ldots, s_n)$.
- If there is are indices i, j such that $s_i = \{s_{i1}\}_{s_j}^{r_i}$, then $F(\phi) = F((\nu\overline{n}).s_1, \ldots, s_{i-1}, s_{i1}, s_{i+1}, \ldots, s_n)$
- Otherwise, $F(\phi) = \phi$.

1. Show that, if ϕ is a transparent frame, then $F(\phi)$ contains only names, constants and encryptions of constants with non-deducible keys.
2. Show that, if ϕ, ϕ' are two transparent frames (that may contain key-cycles), then $F(\phi) \sim F(\phi')$ implies $[\![F(\phi)]\!]_\eta \approx [\![F(\phi')]\!]_\eta$
3. Given two transparent frames ϕ, ϕ', show that $\phi \sim \phi'$ implies $F(\phi) \sim F(\phi')$.
4. Given two transparent frames ϕ, ϕ' such that $\phi \sim \phi'$, show that $[\![\phi]\!]_\eta \approx [\![\phi']\!]_\eta$ iff $[\![F(\phi)]\!]_\eta \approx [\![F(\phi')]\!]_\eta$.
5. Prove an extension of the theorem 12.2, in which the frames may contain key-cycles on deducible keys.

Exercice 64
If we allow arbitrary ground terms (not containing decryption or pairing symbols) as keys, show that the soundness of static equivalence does not hold.

More precisely, construct (from any IND-CPA encryption scheme) another IND-CPA encryption scheme, for which, for two well-chosen terms t_1, t_2, $\{u\}_{t_1}^r \sim \{u\}_{t_2}^r$, none of the names occurring in t_1, t_2 occurs in u, and $[\![\{u\}_{t_1}^r]\!]_\eta \not\approx [\![\{u\}_{t_2}^r]\!]_\eta$.

Exercice 65
Assume here that the encryption scheme is not only IND-CPA, but also *which-key concealing*, which is defined as follows: for any PPT machine $\mathcal{A}$ that has access to two oracles, and security parameter η, let

$$\epsilon_1(\mathcal{A}, \eta) = \begin{array}{l} |\mathbb{P}\left[k, k' \leftarrow \mathcal{K}(\eta), R \leftarrow U : \mathcal{A}^{\mathcal{O}^1_k, \mathcal{O}^1_{k'}}(0^\eta \mid R) = 1\right] \\ \qquad -\mathbb{P}\left[k \leftarrow \mathcal{K}(\eta), R \leftarrow U : \mathcal{A}^{\mathcal{O}^2_k \mathcal{O}^2_k}(0^\eta \mid R) = 1\right]| \end{array}$$

Where $\mathcal{O}_k^1(x,y) = \mathcal{E}(x,k,r)$ and $\mathcal{O}_k^2(x,y) = \mathcal{E}(y,k,r)$ if $|x| = |y|$ (and 0 otherwise).

The encryption scheme is which-key concealing if, for every PPT machine $\mathcal{A}$, $\epsilon_1(\mathcal{A},\eta)$ is negligible.

We consider a new definition of static equivalence $\sim_1$, in which we do not have the predicate EK but, instead, a unary predicate Cipher, which is true exactly on terms that are in $\mathcal{M}_0$ and whose top symbol is an encryption.

1. Show that $\nu k, k'k'', r, r'. \{k''\}_k^r, \{k''\}_{k'}^{r'} \sim_1 \nu k, k', k'', r, r'. \{k''\}_k^r, \{k''\}_k^{r'}$

2. Let $u_1, \ldots, u_m \in \mathcal{M}_0$ be such that all names occurring in $u_1, \ldots, u_m$ are in the domain of τ and $k, k_1, \ldots, k_m, n_1, \ldots, n_m \notin \mathsf{Dom}(\tau)$. Assume the encryption scheme is which-key concealing. Prove

$$[\![\{u_1\}_{k_1}^{n_1}, \ldots, \{u_m\}_{k_m}^{n_m}]\!]_\eta^\tau \approx [\![\{0^{l(u_1,\eta)}\}_k^{n_1}, \ldots, \{0^{l(u_m,\eta)}\}_k^{n_m}]\!]_\eta^\tau$$

3. Assume that $(s_1, \ldots, s_n)$ and $(t_1, \ldots, t_n)$ are two valid sequences of terms and that the encryption scheme is which-key concealing, then prove

$$(\nu\overline{n})s_1, \ldots, s_n \sim_1 (\nu\overline{m})t_1, \ldots, t_n \Rightarrow [\![s_1, \ldots, s_n]\!]_\eta \approx [\![t_1, \ldots, t_n]\!]_\eta$$

12.8 Completeness

The completeness problem is the converse of theorem 12.2: given two sequences of terms that are computationally indistinguishable, are they statically equivalent ? In other words, completeness ensures that we did not give too much power to the symbolic attacker.

The main issue is to be sure that the distinguishing capabilities of the symbolic attacker, the predicates, can be implemented. That is what we consider first.

12.8.1 Predicate implementation

We assume a set of function symbols, all of which have a computational interpretation as a PPT algorithm. Then, if $\mathcal{M}$ is the set of ground terms constructed using this set of function symbols and a set of names, for every mapping τ from names to bitstrings, $[\![\_]\!]_\eta^\tau$ is the unique extension of τ as a homomorphism from $\mathcal{M}$ to $\{0,1\}^*$. As before, we consider name distributions that are parametrized by a security parameter $\eta \in \mathbb{N}$ and, when τ is a partial interpretation only, $[\![\_]\!]_\eta^\tau$ is the corresponding distribution.

Definition 12.11 *A predicate $P \in \mathcal{P}$ of arity k is* implementable *if there is a PPT algorithm $[\![P]\!]$ such that:*

$$\begin{aligned}&\forall s_1, \ldots, s_k \in \mathcal{M}, \exists Q \in \mathrm{POL}_1, \exists N \in \mathbb{N}, \forall \eta > N.\\ &\qquad \mathbb{P}\left[(x_1, \ldots, x_k) \leftarrow [\![s_1, \ldots, s_k]\!]_\eta : [\![P]\!]^{\mathcal{A}}(x_1, \ldots, x_k) = P^I(s_1, \ldots, s_k)\right] > \frac{1}{2} + \frac{1}{Q(\eta)}\end{aligned}$$

We will see later examples of predicate symbols that are (not) implementable.

Definition 12.12 *Given a convergent rewrite system $\mathcal{S}$ on $\mathcal{M}$, and an interpretation of the predicate symbols, a $\mathcal{S}, \mathcal{P}^I$-*computational structure *is a computational interpretation of the function symbols and the predicate symbols such that*

$$\forall s, t \in \mathcal{M}, \forall \eta \in \mathbb{N}, \forall \tau. \ (s =_{\mathcal{S}} t \ \ \Rightarrow \ \ [\![s]\!]_\eta^\tau = [\![t]\!]_\eta^\tau)$$

and, for every $P \in \mathcal{P}$, P is implementable.

Note here that we require not only indistinguishability, but true equality: τ is universally quantified over the assignments of appropriate lengths.

12.8.2 Completeness

Theorem 12.3 (completeness) *For any computational structure,*

$$[\![s_1,\ldots,s_n]\!]_\eta \approx [\![t_1,\ldots,t_n]\!]_\eta \Rightarrow (\nu\overline{n})\; s_1,\ldots,s_n \sim (\nu\overline{m})\; t_1,\ldots,t_n.$$

Proof : By definition, if $s_1,\ldots,s_n \not\sim t_1,\ldots,t_n$, then there are recipes $u_1,\ldots,u_k$ and a predicate $P \in \mathcal{P}$ such that $P^I(u_1\sigma\!\downarrow,\ldots,u_k\sigma\!\downarrow) \not\Leftrightarrow P^I(u_1\sigma'\!\downarrow,\ldots,u_k\sigma'\!\downarrow)$. where $\sigma = \{x_1 \mapsto s_1,\ldots,x_n \mapsto s_n\}$ and $\sigma' = \{x_1 \mapsto t_1;\ldots,x_n \mapsto t_n\}$. For instance, assume w.l.o.g. that $P^I(u_1\sigma\!\downarrow,\ldots,u_k\sigma\!\downarrow) = 1$ and $P^I(u_1\sigma'\!\downarrow,\ldots,u_k\sigma'\!\downarrow) = 0$.

Moreover, $u_1,\ldots,u_k$ do not contain any name, by a simple induction on the length of the rewrite sequence, there is a deterministic polynomial time Turing machine $\mathcal{B}$, which, on input $[\![v_1,\ldots,v_n]\!]_\eta^\tau$, computes $[\![P(u_1\theta\!\downarrow,\ldots,u_k\theta\!\downarrow)]\!]_\eta^\tau$, where $\theta = \{x_1 \mapsto v_1,\ldots,x_n \mapsto v_n\}$. By definition, for every $\eta \in \mathbb{N}$, for every assignment τ of keys and random numbers occcurring in $v_1,\ldots,v_n$,

$$\mathcal{B}([\![v_1,\ldots v_n]\!]_\eta^\tau) = [\![P]\!]([\![u_1\theta\!\downarrow]\!]_\eta^\tau,\ldots [\![u_k\theta\!\downarrow]\!]_\eta^\tau)$$

So,

$$\begin{array}{l}\mathbb{P}\left[(x_1,\ldots,x_n) \leftarrow [\![v_1,\ldots,v_n]\!]_\eta : \mathcal{B}(x_1,\ldots,x_n) = 1\right] = \\ \qquad \mathbb{P}\left[(y_1,\ldots,y_k) \leftarrow [\![u_1\theta\!\downarrow,\ldots,u_k\theta\!\downarrow]\!]_\eta : [\![P]\!](y_1,\ldots,y_k) = 1\right]\end{array}$$

On the other hand, according to the definition, there are Q_1 and N_1 such that, for all $\eta > N_1$,

$$\mathbb{P}\left[(y_1,\ldots,y_k) \leftarrow [\![u_1\sigma\!\downarrow,\ldots,u_k\sigma\!\downarrow]\!]_\eta : [\![P]\!](y_1,\ldots,y_k) = 1\right] > \frac{1}{2} + \frac{1}{Q_1(\eta)}$$

and there are Q_2 and N_2 such that, for all $\eta > N_2$,

$$\mathbb{P}\left[(y_1,\ldots,y_k) \leftarrow [\![u_1\sigma'\!\downarrow,\ldots,u_k\sigma'\!\downarrow]\!]_\eta : [\![P]\!](y_1,\ldots,y_k) = 0\right] > \frac{1}{2} + \frac{1}{Q_2(\eta)}$$

Altogether, if we let

$$\begin{array}{l}\epsilon(\eta) = \\ \qquad \mathbb{P}\left[(x_1,\ldots,x_n) \leftarrow [\![s_1,\ldots,s_n]\!]_\eta : \mathcal{B}(x_1,\ldots,x_n) = 1\right] - \\ \qquad\qquad \mathbb{P}\left[(x_1,\ldots,x_n) \leftarrow [\![t_1,\ldots,t_n]\!]_\eta : \mathcal{B}(x_1,\ldots,x_n) = 1\right]\end{array},$$

For $\eta > \max(N_1, N_2)$,

$$\epsilon(\eta) > (\frac{1}{2} + \frac{1}{Q_1(\eta)}) - (\frac{1}{2} - \frac{1}{Q_2(\eta)}) = \frac{1}{Q_1(\eta)} + \frac{1}{Q_2(\eta)}$$

It follows that $([\![s_1,\ldots,s_n]\!]_\eta)_{\eta\in\mathbb{N}} \not\approx ([\![t_1,\ldots,t_n]\!]_\eta)_{\eta\in\mathbb{N}}$.

12.8.3 Examples of computational structures

M, the predicate defining “valid messages” can be implemented if there is an algorithm, which can recognize when a message is a pair (which we always assume), an algorithm which can recognize when a message is a key, and also an algorithm, which decides when the decryption algorithm is used with a valid input.

More precisely, we assume a particular bitstring $\perp$ (an error message), which is returned when π_i is applied on a bitstring, which is not a pair, or when there is an attempt to encrypt a message, which is not a key or when trying to decrypt something, which is not a ciphertext. Following a strict interpretation, we also assume that any function applied to the error message $\perp$ returns $\perp$. This corresponds actually to a typing predicate, which we assume to be implemented deterministically, for instance using tags (it would also work with a probabilistic implementation of such predicates). Finally, we assume *confusion freeness* as defined below (a definition taken from [**?**], who also show that such a condition is necessary for completeness, and is not ensured by IND-CPA):

Definition 12.13 *An encryption scheme is* confusion free *if, for every bitstring x,*

$$\mathbb{P}\left[k_1, k_2 \leftarrow \mathcal{K}(\eta), r \leftarrow U : \mathcal{D}(\mathcal{E}(x, k_1 \mid r), k_2) \neq \perp\right] = \nu(\eta)$$

is negligible.

In other words: it is very likely to get an error message when trying to decrypt with a wrong key.

Proposition 12.1 *If the encryption scheme is confusion-free, then the predicates M, EQ are implementable.*

Proof : Let us start with M. Let $[\![M]\!](x) = 1$ iff $x \neq \perp$.

Let $s \in \mathcal{M}$. We have to compute

$$\epsilon(s, \eta) \overset{\text{def}}{=} \mathbb{P}\left[x \leftarrow [\![s]\!]_\eta : M^I(s) \neq [\![M]\!](x)\right]$$

If $s \in \mathcal{M}_0$, then $M^I(s) = 1$ and $[\![M]\!](x) = 1$ and therefore $\epsilon(s, \eta) = 0$. So, we only have to consider the case $M^I(s) = 0$. We proceed by induction on s. If s is a constant, then the only possibility for not being accepted by M^I is $s = \perp$, in which case $[\![M]\!](x) = 0$.

Otherwise, if s is a pair or an encryption with a valid key, then, since M^I has a strict interpretation, there must be a direct subterm of s which does not belong to $\mathcal{M}_0$. Then, we use the induction hypothesis and the strictness of $[\![M]\!]$: if s_1, s_2 are the direct subterms of s,

$$\begin{array}{rcl}\epsilon(s, \eta) & = & \mathbb{P}\left[(x_1, x_2) \leftarrow [\![s_1, s_2]\!]_\eta : [\![M]\!](x_1) = 1 \wedge [\![M]\!](x_2) = 1\right] \\ & \leq & \min(\mathbb{P}\left[(x_1, x_2) \leftarrow [\![s_1, s_2]\!]_\eta : [\![M]\!](x_1) = 1\right], \mathbb{P}\left[(x_1, x_2) \leftarrow [\![s_1, s_2]\!]_\eta : [\![M]\!](x_2) = 1\right]) \\ & = & \min(\mathbb{P}\left[x_1 \leftarrow [\![s_1]\!]_\eta : [\![M]\!](x_1) = 1\right], \mathbb{P}\left[x_2 \leftarrow [\![s_2]\!]_\eta : [\![M]\!](x_2) = 1\right]) \\ & \leq & \min(\epsilon(s_1, \eta), \epsilon(s_2, \eta))\end{array}$$

If $s = \{u\}_v^r$ and v is not a valid key or if s is a projection of a term which is not a pair, or if s is a decryptoin of a term which is not a ciphertext, then $[\![s]\!]_\eta$ is always $\perp$, by definition, hence $[\![M]\!](x) = 0$ (for all $x \in [\![s]\!]_\eta$).

Finally, if $s = \mathcal{D}(\{t\}_{k_1}^r, k_2)$, if $\{t\}_{k_1}^r$ is not in M^I, we are back to the previous computation. Assume now that $\{t\}_{k_1}^r \in M^I$ and therefore that $k_2 \neq k_1$. Then, by definition of confusion freeness, $\epsilon(s, \eta) = \nu(\eta)$

Next, $[\![EQ]\!](x, y)$ is defined as $[\![M]\!](x) \wedge [\![M]\!](y) \wedge x = y$, which is implementable, as $M^{\mathcal{A}}$ is implementable.

Now there are several symmetric encryption schemes that are proved to ensure confusion-freeness. Typically, the encryption schemes that satisfy, additionally to IND-CPA, some integrity properties. A discussion and constructions on such authenticated encryption schemes can be found in [**?**].

The converse of theorem 12.2 follows then from the theorem 12.3 and the implementability of EK, EL, which is satisfied by some (but not all) IND-CPA encryption schemes.

Exercice 66
Given an IND-CPA encryption scheme, construct another IND-CPA encryption scheme for which the predicates EK and EL are implementable.

Exercice 67
We extend here the definition of the computational interpretation of terms to terms that may contain decryption symbols. $[\![\mathsf{dec}(s, t)]\!]_\eta^\tau$ is defined by:

1. draw all names occurring in s, t and not in $\mathsf{Dom}(\tau)$,

2. The previous step yielda, together with τ, an assignment τ'; Apply the function $\mathcal{D}$ to $[\![s]\!]_\eta^{\tau'}$ and $[\![t]\!]_\eta^{\tau'}$. This function returns a special bitstring $\perp$ in case it is not defined on the input.

An encryption scheme is *decryption-confusing* if:

1. for every name k, $[\![\{0^\eta\}_k^r]\!]_\eta \approx [\![k]\!]_\eta$
2. for every $x \in \mathcal{M}_0$ and every two distinct names k_1, k_2 not occurring in x,

$$[\![\{\{x\}_{k_1}^{r_1}\}_{k_2}^{r_2}, k_2]\!]_\eta \approx [\![\{x\}_{k_1}^{r_1}, k_2]\!]_\eta$$

1. Show that, if the encryption scheme is decryption confusing, then

   (a) for every distinct names k_1, k_2,

   $$[\![\mathsf{dec}(k_1, k_2), k_2]\!]_\eta \approx [\![k_1, k_2]\!]_\eta \approx [\![\{k_1\}_{k_2}^r, k_2]\!]$$

   (b) For any term x not containing projection symbols, and for any names k_1, k_2 such that $k_1 \neq k_2$, and k_1, k_2 do not occur in x,

   $$[\![\mathsf{dec}(\{x\}_{k_1}^r, k_2), k_2]\!]_\eta \approx [\![\{x\}_{k_1}^r, k_2]\!]_\eta \approx [\![\{\{x\}_{k_1}^{r_1}\}_{k_2}^{r_2}, k_2]\!]_\eta$$

2. In what follows, we assume the encryption scheme IND-CPA and *which-key concealing*, as defined in exercise 65.

   The definition of static equivalence is modified, using a different set of predicate symbols. We use three predicates *El* and *Eq*, m which are interpreted in a slightly different way as before since, now, decryption can not fail. m is interpreted as the set of terms, which do not contain projection symbols. *El* is interpreted as the set of pairs of terms whose lengths are identical, where the length L is defined by:

   $$\begin{array}{rcll} L(k) & = & S & \text{For } k \in \mathcal{N} \\ L(c) & = & S & \text{for any constant c} \end{array} \qquad \begin{array}{rcl} L(D(x,k)) & = & L(x) \\ L(\{x\}_k^r) & = & L(x) \\ L(\langle x, y\rangle) & = & L(x) + L(y) \end{array}$$

   S is an integer constant, used only for the purpose of this definition. Also, we assume $L(c) = S$ for every constant, for simplicity, which means that $\mathcal{W}$ is now restricted to words whose length is a multiple of S. We can think of S as the length of a block in a block cipher encryption scheme.

   Eq is interpreted as equality on the terms that are in m. The rewrite system is unchanged. This defines a staticic equivalence $\sim_c$.

   Show that the following sequences are (statically as well as computationally) distinguishable:

   (a) $(\{\langle \mathbf{0}, k_0\rangle\}_{k_1}^r, k_2, k_1)$ and $(\{\langle \mathbf{1}, k_0\rangle\}_{k_1'}^{r'}, k_1', k_2')$

   (b) $(\{\{\langle \mathbf{0}, k_0\rangle\}_{k_2}^{r_1}\}_{k_1}^{r_2}, k_2, k_1)$ and $(\{\{\langle \mathbf{0}, k_0\rangle\}_{k_2'}^{r_1'}\}_{k_1'}^{r_2'}, k_1', k_2')$

   (c) $(\{k_0\}_{k_1}^{r_1}, k_1, \{\langle k_2, k_3\rangle\}_{k_1}^{r_2}, k_2)$ and $(\{k_0'\}_{k_1'}^{r_1'}, k_2', \{\langle k_2', k_3'\rangle\}_{k_1'}^{r_2'}, k_1')$

   Where $\mathbf{0}$ and $\mathbf{1}$ are sequences of 0's and 1's respectively, of the same appropriate length.

3. Show that the following are statically equivalent:

   (a) $(\{\{\langle \mathbf{0}, k_0\rangle\}_{k_3}^{r_1}\}_{k_1}^{r_2}, k_2, k_1)$ and $(\{\{\langle \mathbf{1}, k_0\rangle\}_{k_3'}^{r_1'}\}_{k_1'}^{r_2'}, k_1', k_2')$

(b) $(\{k_0\}^{r_1}_{k_1}, k_1, k_2, \{\{k_3\}^{r_2}_{k_2}\}^{r_3}_{k_1})$ and $(\{k'_0\}^{r'_1}_{k'_1}, k'_2, k'_1, \{\{k'_3\}^{r'_2}_{k'_2}\}^{r'_3}_{k'_1})$

4. *Valid* sequences are defined as before: sequences of messages in $\mathcal{M}_0$ (in particular not containing decryption symbols) with a consistent use of random numbers and no key-cycle.

   We say that a term sequence is *transparent* if, for every ciphertext $\{u\}^r_k$ occurring in $s_1, \ldots, s_n$, either k is deducible from $s_1, \ldots, s_n$ or else $u \in \mathcal{W}$.

   Let $(s_1, \ldots, s_n)$ be a transparent sequence such that

   (a) $s_1, \ldots, s_n$ are names or ciphertexts

   (b) the only occurrences of $w \in \mathcal{W}$ in the sequence are in expressions $\{w\}^r_k$ where k is not deducible.

   (c) $s_1, \ldots, s_n$ do not contain any pairing

   (d) for every $i \neq j$, $s_i \not\sqsubseteq_{(s_1,\ldots,s_n)} s_j$

   Prove then
$$[\![s_1, \ldots, s_n]\!]_\eta \approx [\![\{0^{l(s_1,\eta)}\}^{r_1}_{k_1}, \ldots, \{0^{l(s_n,\eta)}\}^{r_n}_{k_n}]\!]_\eta$$
   where $k_1, \ldots, k_n \in \mathcal{N}$ are distinct and $r_1, \ldots, r_n$ are distinct.

5. Prove that, if $(s_1, \ldots, s_n)$ and $(t_1, \ldots, t_n)$ are valid transparent sequences of terms and $(s_1, \ldots, s_n) \sim_c (t_1, \ldots, t_n)$, then $s_i \sqsubseteq_{(s_1,\ldots,s_n)} s_j$ implies $t_i \sqsubseteq_{(t_1,\ldots,t_n)} t_j$.

6. Assuming the encryption scheme is decryption confusing, if $s_1, \ldots, s_n$ and $t_1, \ldots, t_n$ are valid sequences, prove
$$(s_1, \ldots, s_n) \sim_c (t_1, \ldots, t_n) \Rightarrow [\![s_1, \ldots, s_n]\!]_\eta \approx [\![t_1, \ldots, t_n]\!]_\eta$$

Bibliography

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

[2] M. Abadi and B. Blanchet. Secrecy types for asymmetric communication. In F. Honsell and M. Miculan, editors, *Foundations of Software Science and Computation Structures (FoSSaCS 2001)*, volume 2030 of *Lecture Notes on Computer Science*, pages 25–41, Genova, Italy, Apr. 2001. Springer.

[3] M. Abadi and B. Blanchet. Analyzing security protocols with secrecy types and logic programs. *Journal of the ACM*, 52(1):102–146, Jan. 2005.

[4] M. Abadi and B. Blanchet. Computer-assisted verification of a protocol for certified email. *Science of Computer Programming*, 58(1–2):3–27, Oct. 2005. Special issue SAS'03.

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

[6] M. Abadi and V. Cortier. Deciding knowledge in security protocols under equational theories. *Theoretical Computer Science*, 387(1-2):2–32, 2006.

[7] M. Abadi and C. 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.

[8] M. Abadi and C. Fournet. Private authentication. *Theoretical Computer Science*, 322(3):427–476, 2004.

[9] M. Abadi, N. Glew, B. Horne, and B. Pinkas. Certified email with a light on-line trusted third party: Design and implementation. In *11th International World Wide Web Conference*, pages 387–395, Honolulu, Hawaii, May 2002. ACM Press.

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

[11] M. Abadi and R. M. Needham. Prudent engineering practice for cryptographic protocols. *IEEE Trans. Software Eng.*, 22(1):6–15, 1996.

[12] M. Abadi and P. Rogaway. Reconciling two views of cryptography: the computational soundness of formal encryption. In *Proc. 1rst IFIP International Conference on Theoretical Computer Science*, volume 1872 of *Lecture Notes in Computer Science*, Sendai, Japan, 2000.

[13] M. Abadi and P. Rogaway. Reconciling two views of cryptography (the computational soundness of formal encryption). *Journal of Cryptology*, 15(2):103–127, 2002.

[14] M. Abdalla, P.-A. Fouque, and D. Pointcheval. Password-based authenticated key exchange in the three-party setting. *IEE Proceedings Information Security*, 153(1):27–39, Mar. 2006.

[15] P. Adão, G. Bana, J. Herzog, and A. Scedrov. Soundness of formal encryption in the presence of key-cycles. In S. de Capitani di Vimercati, P. Syverson, and D. Gollmann, editors, *Proceedings of the 10th European Symposium On Research In Computer Security (ESORICS 2005)*, volume 3679 of *Lecture Notes on Computer Science*, pages 374–396, Milan, Italy, Sept. 2005. Springer.

[16] R. Affeldt, D. Nowak, and K. Yamada. Certifying assembly with formal cryptographic proofs: the case of BBS. In *9th International Workshop on Automated Verification of Critical Systems (AVoCS'09)*, volume 23 of *Electronic Communications of the EASST*. EASST, Sept. 2009.

[17] W. Aiello, S. M. Bellovin, M. Blaze, R. Canetti, J. Ioannidis, K. Keromytis, and O. Reingold. Just Fast Keying: Key agreement in a hostile Internet. *ACM Transactions on Information and System Security*, 7(2):242–273, May 2004.

[18] X. Allamigeon and B. Blanchet. Reconstruction of attacks against cryptographic protocols. In *18th IEEE Computer Security Foundations Workshop (CSFW-18)*, pages 140–154, Aix-en-Provence, France, June 2005. IEEE.

[19] American National Standards Institute. Public Key Cryptography for the Financial Services Industry: The Elliptic Curve Digital Signature Algorithm. ANSI X9.62-1998. January 1999.

[20] R. Anderson and R. Needham. Programming Satan's computer. In J. van Leeuven, editor, *Computer Science Today: Recent Trends and Developments*, volume 1000 of *Lecture Notes on Computer Science*, pages 426–440. Springer, 1995.

[21] S. Andova, C. Cremers, K. G. Steen, S. Mauw, S. M. lsnes, and S. Radomirović. Sufficient conditions for composing security protocols. *Information and Computation*, 206(2-4):425–459, 2008.

[22] M. Arapinis, S. Delaune, and S. Kremer. From one session to many: Dynamic tags for security protocols. In I. Cervesato, H. Veith, and A. Voronkov, editors, *Proceedings of the 15th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR'08)*, volume 5330 of *Lecture Notes in Artificial Intelligence*, pages 128–142, Doha, Qatar, Nov. 2008. Springer.

[23] M. Arapinis and M. Duflot. Bounding messages for free in security protocols. In *Proc. 27th Conference on Foundations of Software Technology and Theoretical Computer Science (FST&TCS'07)*, volume 4855 of *LNCS*, pages 376–387. Springer, 2007.

[24] A. Armando, D. A. Basin, Y. Boichut, Y. Chevalier, L. Compagna, J. Cuéllar, P. H. Drielsma, P.-C. Héam, O. Kouchnarenko, J. Mantovani, S. Mödersheim, D. von Oheimb, M. Rusinowitch, J. Santiago, M. Turuani, L. Viganò, and L. 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.

[25] M. Arnaud, V. Cortier, and S. Delaune. Combining algorithms for deciding knowledge in security protocols. In F. Wolter, editor, *Proceedings of the 6th International Symposium on Frontiers of Combining Systems (FroCoS'07)*, volume 4720 of *Lecture Notes in Artificial Intelligence*, pages 103–117, Liverpool, UK, Sept. 2007. Springer.

[26] L. Bachmair and H. Ganzinger. Resolution theorem proving. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume 1, chapter 2, pages 19–100. North Holland, 2001.

[27] M. Backes, D. Hofheinz, and D. Unruh. CoSP: A general framework for computational soundness proofs. In *ACM Conference on Computer and Communications Security (CCS'09)*, pages 66–78, Chicago, IL, USA, Nov. 2009. ACM.

[28] M. Backes, C. Hritcu, and M. Maffei. Automated verification of remote electronic voting protocols in the applied pi-calculus. In *Proc. 21st IEEE Computer Security Foundations Symposium, (CSF'08)*, pages 195–209. IEEE Comp. Soc. Press, 2008.

[29] M. Backes and P. Laud. Computationally sound secrecy proofs by mechanized flow analysis. In *Proceedings of 13th ACM Conference on Computer and Communications Security (CCS'06)*, pages 370–379, Alexandria, VA, Nov. 2006. ACM.

[30] M. Backes, M. Maffei, and D. Unruh. Zero-knowledge in the applied pi-calculus and automated verification of the direct anonymous attestation protocol. In *29th IEEE Symposium on Security and Privacy*, pages 202–215, Oakland, CA, May 2008. IEEE. Technical report version available at `http://eprint.iacr.org/2007/289`.

[31] M. Backes, M. Maffei, and D. Unruh. Computationally sound verification of source code. In *17th ACM Conference on Computer and Communications Security (CCS'10)*, pages 387–398, Chicago, IL, USA, Oct. 2010. ACM Press.

[32] M. Backes and B. Pfitzmann. Symmetric encryption in a simulatable Dolev-Yao style cryptographic library. In *17th IEEE Computer Security Foundations Workshop*, pages 204–218, Pacific Grove, CA, June 2004. IEEE.

[33] M. Backes and B. Pfitzmann. Relating symbolic and cryptographic secrecy. *IEEE Transactions on Dependable and Secure Computing*, 2(2):109–123, Apr. 2005.

[34] M. Backes, B. Pfitzmann, and M. Waidner. A composable cryptographic library with nested operations. In *10th ACM conference on Computer and communication security (CCS'03)*, pages 220–230, Washington D.C., Oct. 2003. ACM.

[35] B. Barak, R. Canetti, J. Nielsen, and R. Pass. Universally composable protocols with relaxed set-up assumptions. In *Proc. 45th Symposium on Foundations of Computer Science (FOCS'04)*, pages 186–195. IEEE Comp. Soc. Press, 2004.

[36] G. Barthe, M. Daubignard, B. Kapron, and Y. Lakhnech. Computational indistinguishability logic. In *17th ACM Conference on Computer and Communications Security (CCS'10)*, pages 375–386, Chicago, IL, USA, Oct. 2010. ACM Press.

[37] G. Barthe, B. Grégoire, S. Z. Béguelin, and Y. Lakhnech. Beyond provable security. Verifiable IND-CCA security of OAEP. In A. Kiayias, editor, *Topics in Cryptology - CT-RSA 2011*, volume 6558 of *Lecture Notes on Computer Science*, pages 180–196, San Francisco, CA, USA, Feb. 2011. Springer.

[38] G. Barthe, B. Grégoire, S. Heraud, and S. Z. Béguelin. Computer-aided security proofs for the working cryptographer. In *Advances in Cryptology – CRYPTO 2011*, Santa Barbara, CA, USA, Aug. 2011. To appear.

[39] G. Barthe, B. Grégoire, S. Heraut, and S. Z. Béguelin. Formal certification of ElGamal encryption. a gentle introduction to CertiCrypt. In P. Degano, J. Guttman, and F. Martinelli, editors, *5th International Workshop on Formal Aspects in Security and Trust, FAST 2008*, volume 5491 of *Lecture Notes on Computer Science*, pages 1–19, Malaga, Spain, 2009. Springer.

[40] G. Barthe, B. Grégoire, and S. Zanella. Formal certification of code-based cryptographic proofs. In *36th ACM SIGPLAN - SIGACT Symposium on Principles of Programming Languages (POPL'09)*, pages 90–101, Savannah, Georgia, Jan. 2009. ACM.

[41] M. 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.

[42] M. Baudet. *Sécurité des protocoles cryptographiques : aspects logiques et calcula toires.* Thèse de doctorat, LSV, ENS Cachan, France, Jan. 2007.

[43] M. Baudet, V. Cortier, and S. Delaune. YAPA: A generic tool for computing intruder knowledge. In R. Treinen, editor, *Proceedings of the 20th International Conference on Rewriting Techniques and Applications (RTA'09)*, volume 5595 of *Lecture Notes in Computer Science*, pages 148–163, Brasília, Brazil, June-July 2009. Springer.

[44] O. Baudron, D. Pointcheval, and J. Stern. Extended Notions of Security for Multicast Public Key Cryptosystems. In *Proc. of the 27th ICALP*, LNCS 1853, pages 499–511. Springer-Verlag, Berlin, 2000.

[45] S. Z. Béguelin, G. Barthe, S. Heraud, B. Grégoire, and D. Hedin. A machine-checked formalization of sigma-protocols. In *23rd Computer Security Foundations Symposium (CSF'10)*, pages 246–260, Edinburgh, UK, July 2010. IEEE.

[46] S. Z. Béguelin, B. Grégoire, G. Barthe, and F. Olmedo. Formally certifying the security of digital signature schemes. In *30th IEEE Symposium on Security and Privacy, S&P 2009*, pages 237–250, Oakland, CA, May 2009. IEEE.

[47] M. Bellare. Practice-Oriented Provable Security. In *ISW '97*, LNCS 1396. Springer-Verlag, Berlin, 1997.

[48] M. Bellare, A. Boldyreva, and S. Micali. Public-key Encryption in a Multi-User Setting: Security Proofs and Improvements. In *Eurocrypt '00*, LNCS 1807, pages 259–274. Springer-Verlag, Berlin, 2000.

[49] M. Bellare, A. Boldyreva, and A. Palacio. A Separation between the Random-Oracle Model and the Standard Model for a Hybrid Encryption Problem, 2003. Cryptology ePrint Archive 2003/077.

[50] M. Bellare, A. Desai, E. Jokipii, and P. Rogaway. A concrete security treatment of symmetric encryption. In *Proceedings of the 38th Symposium on Foundations of Computer Science (FOCS'97)*, pages 394–403, Miami Beach, Florida, Oct. 1997. IEEE. Full paper available at `http://www-cse.ucsd.edu/users/mihir/papers/sym-enc.html`.

[51] M. Bellare, A. Desai, D. Pointcheval, and P. Rogaway. Relations among Notions of Security for Public-Key Encryption Schemes. In *Crypto '98*, LNCS 1462, pages 26–45. Springer-Verlag, Berlin, 1998.

[52] M. Bellare, J. Kilian, and P. Rogaway. The security of the cipher block chaining message authentication code. *Journal of Computer and System Sciences*, 61(3):362–399, Dec. 2000.

[53] M. Bellare and C. Namprempre. Authenticated encryption: relations among notions and analysis of the generic composition paradigm. In *Advances in Cryptology – ASIACRYPT 2000*, volume 1976 of *Lecture Notes in Computer Science*, pages 531–545, Kyoto, 2000.

[54] M. Bellare and A. Palacio. GQ and Schnorr Identification Schemes: Proofs of Security against Impersonation under Active and Concurrent Attacks. In *Crypto '02*, LNCS 2442, pages 162–177. Springer-Verlag, Berlin, 2002.

[55] M. Bellare, D. Pointcheval, and P. Rogaway. Authenticated Key Exchange Secure Against Dictionary Attacks. In *Eurocrypt '00*, LNCS 1807, pages 139–155. Springer-Verlag, Berlin, 2000.

[56] M. Bellare and P. Rogaway. Random Oracles Are Practical: a Paradigm for Designing Efficient Protocols. In *Proc. of the 1st CCS*, pages 62–73. ACM Press, New York, 1993.

[57] M. Bellare and P. Rogaway. Optimal Asymmetric Encryption – How to Encrypt with RSA. In *Eurocrypt '94*, LNCS 950, pages 92–111. Springer-Verlag, Berlin, 1995.

[58] M. Bellare and P. Rogaway. The Exact Security of Digital Signatures – How to Sign with RSA and Rabin. In *Eurocrypt '96*, LNCS 1070, pages 399–416. Springer-Verlag, Berlin, 1996.

[59] M. Bellare and P. Rogaway. The Game-Playing Technique and its Application to Triple Encryption, 2004. Cryptology ePrint Archive 2004/331.

[60] M. Bellare and P. Rogaway. The security of triple encryption and a framework for code-based game-playing proofs. In S. Vaudenay, editor, *Advances in Cryptology – Eurocrypt 2006 Proceedings*, volume 4004 of *Lecture Notes on Computer Science*, pages 409–426, Saint Petersburg, Russia, May 2006. Springer. Extended version available at `http://eprint.iacr.org/2004/331`.

[61] S. M. Bellovin and M. Merritt. Encrypted key exchange: Password-based protocols secure against dictionary attacks. In *Proc. Symposium on Security and Privacy (SP'92)*, pages 72–84. IEEE Comp. Soc., 1992.

[62] K. Bhargavan, R. Corin, C. Fournet, and A. Gordon. Secure sessions for web services. In *ACM Workshop on Secure Web Services (SWS'04)*, Washington DC, Oct. 2004.

[63] K. Bhargavan, R. Corin, C. Fournet, and E. Zălinescu. Cryptographically verified implementations for TLS. In *Proceedings of the 15th ACM Conference on Computer and Communications Security (CCS'08)*, pages 459–468. ACM, Oct. 2008.

[64] K. Bhargavan, C. Fournet, and A. Gordon. Verifying policy-based security for web services. In *ACM Conference on Computer and Communications Security (CCS'04)*, pages 268–277, Washington DC, Oct. 2004. ACM.

[65] K. Bhargavan, C. Fournet, and A. Gordon. Verified reference implementations of WS-Security protocols. In M. Bravetti, M. Núñez, and G. Zavattaro, editors, *3rd International Workshop on Web Services and Formal Methods (WS-FM 2006)*, volume 4184 of *Lecture Notes on Computer Science*, pages 88–106, Vienna, Austria, Sept. 2006. Springer.

[66] K. Bhargavan, C. Fournet, A. Gordon, and N. Swamy. Verified implementations of the information card federated identity-management protocol. In *ACM Symposium on Information, Computer and Communications Security (ASIACCS'08)*, pages 123–135, Tokyo, Japan, Mar. 2008. ACM.

[67] K. Bhargavan, C. Fournet, A. Gordon, and S. Tse. Verified interoperable implementations of security protocols. In *19th IEEE Computer Security Foundations Workshop (CSFW'06)*, pages 139–152, Venice, Italy, July 2006. IEEE Computer Society.

[68] K. Bhargavan, C. Fournet, A. D. Gordon, and R. Pucella. TulaFale: A security tool for web services. In F. S. de Boer, M. M. Bonsangue, S. Graf, and W.-P. de Roever, editors, *Formal Methods for Components and Objects (FMCO 2003)*, volume 3188 of *Lecture Notes on Computer Science*, pages 197–222, Leiden, The Netherlands, Nov. 2003. Springer. Paper and tool available at `http://securing.ws/`.

[69] E. Biham and A. Shamir. Differential Fault Analysis of Secret Key Cryptosystems. In *Crypto '97*, LNCS 1294, pages 513–525. Springer-Verlag, Berlin, 1997.

[70] B. Blanchet. *ProVerif Automatic Cryptographic Protocol Verifier User Manual.* http://www.proverif.ens.fr/proverif-manual.pdf.

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

[72] B. Blanchet. Automatic proof of strong secrecy for security protocols. In *IEEE Symposium on Security and Privacy*, pages 86–100, Oakland, California, May 2004.

[73] B. Blanchet. Security protocols: From linear to classical logic by abstract interpretation. *Information Processing Letters*, 95(5):473–479, Sept. 2005.

[74] B. Blanchet. Computationally sound mechanized proofs of correspondence assertions. In *20th IEEE Computer Security Foundations Symposium (CSF'07)*, pages 97–111, Venice, Italy, July 2007. IEEE. Extended version available as ePrint Report 2007/128, http://eprint.iacr.org/2007/128.

[75] B. Blanchet. Automatic verification of correspondences for security protocols. Report arXiv:0802.3444v1, 2008. Available at http://arxiv.org/abs/0802.3444v1.

[76] B. Blanchet. A computationally sound mechanized prover for security protocols. *IEEE Transactions on Dependable and Secure Computing*, 5(4):193–207, Oct.–Dec. 2008.

[77] B. Blanchet. Automatic verification of correspondences for security protocols. *Journal of Computer Security*, 17(4):363–434, July 2009.

[78] B. Blanchet, M. Abadi, and C. Fournet. Automated verification of selected equivalences for security protocols. *Journal of Logic and Algebraic Programming*, 75(1):3–51, Feb.–Mar. 2008.

[79] B. Blanchet and A. Chaudhuri. Automated formal analysis of a protocol for secure file sharing on untrusted storage. In *IEEE Symposium on Security and Privacy*, pages 417–431, Oakland, CA, May 2008. IEEE.

[80] B. Blanchet, A. D. Jaggard, A. Scedrov, and J.-K. Tsay. Computationally sound mechanized proofs for basic and public-key Kerberos. In *ACM Symposium on Information, Computer and Communications Security (ASIACCS'08)*, pages 87–99, Tokyo, Japan, Mar. 2008. ACM.

[81] B. Blanchet and A. Podelski. Verification of cryptographic protocols: Tagging enforces termination. *Theoretical Computer Science*, 333(1-2):67–90, Mar. 2005. Special issue FoSSaCS'03.

[82] B. Blanchet and D. Pointcheval. Automated security proofs with sequences of games. In C. Dwork, editor, *Advances in Cryptology – CRYPTO 2006*, volume 4117 of *Lecture Notes on Computer Science*, pages 537–554, Santa Barbara, CA, Aug. 2006. Springer.

[83] D. Bleichenbacher. Generating El Gamal Signatures without Knowing the Secret Key. In *Eurocrypt '96*, LNCS 1070, pages 10–18. Springer-Verlag, Berlin, 1996.

[84] D. Bleichenbacher. A Chosen Ciphertext Attack against Protocols based on the RSA Encryption Standard PKCS #1. In *Crypto '98*, LNCS 1462, pages 1–12. Springer-Verlag, Berlin, 1998.

[85] C. Bodei. *Security Issues in Process Calculi.* PhD thesis, Università di Pisa, Jan. 2000.

[86] D. Boneh, R. DeMillo, and R. Lipton. On the Importance of Checking Cryptographic Protocols for Faults. In *Eurocrypt '97*, LNCS 1233, pages 37–51. Springer-Verlag, Berlin, 1997.

[87] J. Borgström. Static equivalence *is* harder than knowledge. *Electr. Notes Theor. Comput. Sci.*, 154(3):45–57, 2006.

[88] X. Boyen. The uber-assumption family. In *Pairing '08*, LNCS 5209, pages 39–56. Springer-Verlag, Berlin, 2008.

[89] E. Brickell, D. Pointcheval, S. Vaudenay, and M. Yung. Design Validations for Discrete Logarithm Based Signature Schemes. In *PKC '00*, LNCS 1751, pages 276–292. Springer-Verlag, Berlin, 2000.

[90] D. R. L. Brown and D. B. Johnson. Formal Security Proofs for a Signature Scheme with Partial Message Recovery. In *CT – RSA '01*, LNCS 2020, pages 126–142. Springer-Verlag, Berlin, 2001.

[91] M. Burrows, M. Abadi, and R. Needham. A logic of authentication. *Proceedings of the Royal Society of London A*, 426(1871):233–271, dec 1989. A preliminary version appeared as Digital Equipment Corporation Systems Research Center report No. 39, February 1989.

[92] R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Proc. 42nd Annual Symposium on Foundations of Computer Science (FOCS'01)*, pages 136–145. IEEE Comp. Soc., 2001.

[93] R. Canetti, Y. Dodis, R. Pass, and S. Walfish. Universally composable security with global setup. In *Proc. 4th Theory of Cryptography Conference, (TCC'07)*, LNCS, pages 61–85. Springer, 2007.

[94] R. Canetti, O. Goldreich, and S. Halevi. The Random Oracle Methodology, Revisited. In *Proc. of the 30th STOC*, pages 209–218. ACM Press, New York, 1998.

[95] R. Canetti and J. Herzog. Universally composable symbolic analysis of mutual authentication and key exchange protocols. In S. Halevi and T. Rabin, editors, *Proceedings, Theory of Cryptography Conference (TCC'06)*, volume 3876 of *Lecture Notes on Computer Science*, pages 380–403, New York, NY, Mar. 2006. Springer. Extended version available at `http://eprint.iacr.org/2004/334`.

[96] R. Canetti, C. Meadows, and P. F. Syverson. Environmental requirements for authentication protocols. In *Proc. Symposium on Software Security – Theories and Systems*, volume 2609 of *LNCS*, pages 339–355. Springer, 2002.

[97] R. Canetti and T. Rabin. Universal composition with joint state. In *Proc. 23rd International Cryptology Conference (CRYPTO'03)*, LNCS, pages 265–281. Springer, 2003.

[98] L. Cardelli, G. Ghelli, and A. D. Gordon. Secrecy and group creation. In C. Palamidessi, editor, *CONCUR 2000: Concurrency Theory*, volume 1877 of *Lecture Notes on Computer Science*, pages 365–379. Springer, Aug. 2000.

[99] S. Cavallar, B. Dodson, A. K. Lenstra, W. Lioen, P. L. Montgomery, B. Murphy, H. te Riele, K. Aardal, J. Gilchrist, G. Guillerm, P. Leyland, J. Marchand, F. Morain, A. Muffett, C. Putnam, C. Putnam, and P. Zimmermann. Factorization of a 512-bit RSA Modulus. In *Eurocrypt '00*, LNCS 1807, pages 1–18. Springer-Verlag, Berlin, 2000.

[100] Y. Chevalier, R. Küsters, M. Rusinowitch, and M. Turuani. Deciding the security of protocols with Diffie-Hellman exponentiation and products in exponents. In P. K. Pandya and J. Radhakrishnan, editors, *FST TCS 2003: Foundations of Software Technology and Theoretical Computer Science, 23rd Conference*, volume 2914 of *Lecture Notes on Computer Science*, pages 124–135, Mumbai, India, Dec. 2003. Springer.

[101] Y. Chevalier, R. Küsters, M. Rusinowitch, and M. Turuani. An NP decision procedure for protocol insecurity with XOR. *Theoretical Computer Science*, 338(1–3):247–274, June 2005.

[102] B. Chor and R. L. Rivest. A Knapsack Type Public Key Cryptosystem based on Arithmetic in Finite Fields. In *Crypto '84*, LNCS 196, pages 54–65. Springer-Verlag, Berlin, 1985.

[103] Ş. Ciobâcă and V. Cortier. Protocol composition for arbitrary primitives. In *Proceedings of the 23rd IEEE Computer Security Foundations Symposium (CSF'10)*, Edinburgh, Scotland, UK, July 2010. IEEE Computer Society Press. To appear.

[104] Ş. Ciobâcă, S. Delaune, and S. Kremer. Computing knowledge in security protocols under convergent equational theories. In R. Schmidt, editor, *Proceedings of the 22nd International Conference on Automated Deduction (CADE'09)*, Lecture Notes in Artificial Intelligence, pages 355–370, Montreal, Canada, Aug. 2009. Springer.

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

[106] J. Clulow. The design and analysis of cryptographic APIs for security devices. Master's thesis, University of Natal, Durban, South Africa, 2003. Chapter 3.

[107] E. Cohen. Proving protocols safe from guessing. In *Foundations of Computer Security*, Copenhagen, Denmark, July 2002.

[108] H. Comon-Lundh and V. Cortier. New decidability results for fragments of first-order logic and application to cryptographic protocols. In R. Nieuwenhuis, editor, *14th Int. Conf. Rewriting Techniques and Applications (RTA'2003)*, volume 2706 of *Lecture Notes on Computer Science*, pages 148–164, Valencia, Spain, June 2003. Springer.

[109] H. Comon-Lundh and V. Cortier. Security properties: two agents are sufficient. *Science of Computer Programming*, 50(1-3):51–71, 2004.

[110] H. Comon-Lundh and V. Cortier. Computational soundness of observational equivalence. In *Proceedings of the 15th ACM Conference on Computer and Communications Security (CCS'08)*, pages 109–118, Alexandria, Virginia, USA, Oct. 2008. ACM Press.

[111] H. Comon-Lundh, V. Cortier, and E. Zlinescu. Deciding security properties of cryptographic protocols. application to key cycles. *Transaction on Computational Logic*, 11(2), 2010.

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

[113] R. Corin, J. Doumen, and S. Etalle. Analysing password protocol security against off-line dictionary attacks. *ENTCS*, 121:47–63, 2005.

[114] R. Corin, S. Malladi, J. Alves-Foss, and S. Etalle. Guess what? here is a new tool that finds some new guessing attacks. In R. Gorrieri, editor, *Workshop on Issues in the Theory of Security (WITS'03)*, Warsaw, Poland, Apr. 2003.

[115] J.-S. Coron. On the Exact Security of Full-Domain-Hash. In *Crypto '00*, LNCS 1880, pages 229–235. Springer-Verlag, Berlin, 2000.

[116] V. Cortier, J. Delaitre, and S. Delaune. Safely composing security protocols. In *Proc. 27th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'07)*, volume 4855 of *LNCS*, pages 352–363. Springer, 2007.

[117] V. Cortier and S. Delaune. Deciding knowledge in security protocols for monoidal equational theories. In N. Dershowitz and A. Voronkov, editors, *Proceedings of the 14th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR'07)*, volume 4790 of *Lecture Notes in Artificial Intelligence*, pages 196–210, Yerevan, Armenia, Oct. 2007. Springer.

[118] V. Cortier and S. Delaune. A method for proving observational equivalence. In *Proceedings of the 22nd IEEE Computer Security Foundations Symposium (CSF'09)*, pages 266–276, Port Jefferson, NY, USA, July 2009. IEEE Computer Society Press.

[119] V. Cortier and S. Delaune. Safely composing security protocols. *Formal Methods in System Design*, 34(1):1–36, Feb. 2009.

[120] V. Cortier, S. Delaune, and P. Lafourcade. A survey of algebraic properties used in cryptographic protocols. *Journal of Computer Security*, 14(1):1–43, 2006.

[121] V. Cortier, H. Hördegen, and B. Warinschi. Explicit randomness is not necessary when modeling probabilistic encryption. In C. Dima, M. Minea, and F. Tiplea, editors, *Workshop on Information and Computer Security (ICS 2006)*, volume 186 of *Electronic Notes in Theoretical Computer Science*, pages 49–65, Timisoara, Romania, Sept. 2006. Elsevier.

[122] V. Cortier and B. Warinschi. Computationally sound, automated proofs for security protocols. In M. Sagiv, editor, *Proc. 14th European Symposium on Programming (ESOP'05)*, volume 3444 of *Lecture Notes on Computer Science*, pages 157–171, Edimbourg, U.K., Apr. 2005. Springer.

[123] J. Courant, M. Daubignard, C. Ene, P. Lafourcade, and Y. Lakhnech. Towards automated proofs for asymmetric encryption schemes in the random oracle model. In *Proceedings of the 15th ACM conference on Computer and communications security (CCS'08)*, pages 371–380, Alexandria, Virginia, USA, Oct. 2008. ACM.

[124] J. Courant, M. Daubignard, C. Ene, P. Lafourcade, and Y. Lakhnech. Automated proofs for asymmetric encryption. In D. Dams, U. Hannemann, and M. Steffen, editors, *Concurrency, Compositionality, and Correctness*, volume 5930 of *Lecture Notes on Computer Science*, pages 300–321. Springer, 2010.

[125] J. Courant, C. Ene, and Y. Lakhnech. Computationally sound typing for non-interference: The case of deterministic encryption. In V. Arvind and S. Prasad, editors, *27th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'07)*, volume 4855 of *Lecture Notes on Computer Science*, pages 364–375, New Delhi, India, Dec. 2007. Springer.

[126] R. Cramer and V. Shoup. A Practical Public Key Cryptosystem Provably Secure against Adaptive Chosen Ciphertext Attack. In *Crypto '98*, LNCS 1462, pages 13–25. Springer-Verlag, Berlin, 1998.

[127] A. Datta, A. Derek, J. C. Mitchell, and A. Roy. Protocol composition logic (PCL). *Electr. Notes Theoretical Computer Science*, 172:311–358, 2007.

[128] A. Datta, A. Derek, J. C. Mitchell, V. Shmatikov, and M. Turuani. Probabilistic polynomial-time semantics for a protocol security logic. In L. Caires and L. Monteiro, editors, *ICALP 2005: the 32nd International Colloquium on Automata, Languages and Programming*, volume 3580 of *Lecture Notes on Computer Science*, pages 16–29, Lisboa, Portugal, July 2005. Springer.

[129] A. Datta, A. Derek, J. C. Mitchell, and B. Warinschi. Computationally sound compositional logic for key exchange protocols. In *Proceedings of 19th IEEE Computer Security Foundations Workshop (CSFW'06)*, pages 321–334, Venice, Italy, July 2006. IEEE Computer Society.

[130] H. de Nivelle. *Ordering Refinements of Resolution.* PhD thesis, Technische Universiteit Delft, Oct. 1995.

[131] S. Delaune and F. Jacquemard. A theory of dictionary attacks and its complexity. In *17th IEEE Computer Security Foundations Workshop*, pages 2–15, Pacific Grove, CA, June 2004. IEEE.

[132] S. Delaune and F. Jacquemard. Decision procedures for the security of protocols with probabilistic encryption against offline dictionary attacks. *Journal of Automated Reasoning*, 36(1-2):85–124, Jan. 2006.

[133] S. Delaune, S. Kremer, and M. D. Ryan. Composition of password-based protocols. In *Proceedings of the 21st IEEE Computer Security Foundations Symposium (CSF'08)*, pages 239–251, Pittsburgh, PA, USA, June 2008. IEEE Computer Society Press.

[134] S. Delaune, S. Kremer, and M. D. Ryan. Symbolic bisimulation for the applied pi calculus. *Journal of Computer Security*, 2009. To appear.

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

[136] G. Denker, J. Meseguer, and C. Talcott. Protocol specification and analysis in Maude. In N. Heintze and J. Wing, editors, *Workshop on Formal Methods and Security Protocols*, Indianapolis, Indiana, 25 June 1998.

[137] D. E. Denning and G. M. Sacco. Timestamps in key distribution protocols. *Commun. ACM*, 24(8):533–536, Aug. 1981.

[138] W. Diffie and M. E. Hellman. New Directions in Cryptography. *IEEE Transactions on Information Theory*, IT–22(6):644–654, November 1976.

[139] D. Dolev, C. Dwork, and M. Naor. Non-Malleable Cryptography. *SIAM Journal on Computing*, 30(2):391–437, 2000.

[140] D. Dolev and A. C. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, IT-29(12):198–208, Mar. 1983.

[141] P. H. Drielsma, S. Mödersheim, and L. Viganò. A formalization of off-line guessing for security protocol analysis. In F. Baader and A. Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning: 11th International Conference, LPAR 2004*, volume 3452 of *Lecture Notes on Computer Science*, pages 363–379, Montevideo, Uruguay, Mar. 2005. Springer.

[142] N. A. Durgin, P. D. Lincoln, J. C. Mitchell, and A. Scedrov. Undecidability of bounded security protocols. In *Workshop on Formal Methods and Security Protocols (FMSP'99)*, Trento, Italy, 5 July 1999.

[143] T. El Gamal. A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. *IEEE Transactions on Information Theory*, IT–31(4):469–472, July 1985.

[144] F. J. T. Fábrega, J. C. Herzog, and J. D. Guttman. Strand spaces: Proving security protocols correct. *Journal of Computer Security*, 7(2/3):191–230, 1999.

[145] A. Fiat and A. Shamir. How to Prove Yourself: Practical Solutions of Identification and Signature Problems. In *Crypto '86*, LNCS 263, pages 186–194. Springer-Verlag, Berlin, 1987.

[146] C. Fournet and M. 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.

[147] E. Fujisaki, T. Okamoto, D. Pointcheval, and J. Stern. RSA–OAEP is Secure under the RSA Assumption. In *Crypto '01*, LNCS 2139, pages 260–274. Springer-Verlag, Berlin, 2001. Also appeared as *RSA–OAEP is Still Alive* in the Cryptology ePrint Archive 2000/061. November 2000.
Available from `http://eprint.iacr.org/`.

[148] E. Fujisaki, T. Okamoto, D. Pointcheval, and J. Stern. RSA–OAEP is Secure under the RSA Assumption. *Journal of Cryptology*, 17(2):81–104, 2004.

[149] J. C. Godskesen. Formal verification of the ARAN protocol using the applied pi-calculus. In *Proceedings of the Sixth International IFIP WG 1.7 Workshop on Issues in the Theory of Security (WITS'06)*, pages 99–113, Vienna, Austria, Mar. 2006.

[150] O. Goldreich, S. Goldwasser, and S. Micali. How to Construct Random Functions. *Journal of the ACM*, 33(4):792–807, 1986.

[151] S. Goldwasser and S. Micali. Probabilistic Encryption. *Journal of Computer and System Sciences*, 28:270–299, 1984.

[152] S. Goldwasser, S. Micali, and C. Rackoff. The Knowledge Complexity of Interactive Proof Systems. In *Proc. of the 17th STOC*, pages 291–304. ACM Press, New York, 1985.

[153] S. Goldwasser, S. Micali, and R. Rivest. A "Paradoxical" Solution to the Signature Problem. In *Proc. of the 25th FOCS*, pages 441–448. IEEE, New York, 1984.

[154] S. Goldwasser, S. Micali, and R. Rivest. A Digital Signature Scheme Secure Against Adaptative Chosen-Message Attacks. *SIAM Journal of Computing*, 17(2):281–308, April 1988.

[155] L. Gong. Verifiable-text attacks in cryptographic protocols. In *INFOCOM '90, The Conference on Computer Communications*, pages 686–693, San Francisco, CA, June 1990. IEEE.

[156] L. Gong, T. M. A. Lomas, R. M. Needham, and J. H. Saltzer. Protecting poorly chosen secrets from guessing attacks. *IEEE Journal on Selected Areas in Communications*, 11(5):648–656, June 1993.

[157] L. Gong and P. Syverson. Fail-stop protocols: An approach to designing secure protocols. In *Proc. 5th International Working Conference on Dependable Computing for Critical Applications*, pages 44–55, 1995.

[158] A. Gordon and A. Jeffrey. Authenticity by typing for security protocols. *Journal of Computer Security*, 11(4):451–521, 2003.

[159] J. Goubault-Larrecq. Deciding $\mathcal{H}_1$ by resolution. *Information Processing Letters*, 95(3):401–408, Aug. 2005.

[160] J. Goubault-Larrecq and F. Parrennes. Cryptographic protocol analysis on real C code. In R. Cousot, editor, *Proceedings of the 6th International Conference on Verification, Model Checking and Abstract Interpretation (VMCAI'05)*, volume 3385 of *Lecture Notes on Computer Science*, pages 363–379, Paris, France, Jan. 2005. Springer.

[161] J. D. Guttman and F. J. Thayer. Protocol independence through disjoint encryption. In *Proc. 13th Computer Security Foundations Workshop (CSFW'00)*, pages 24–34. IEEE Comp. Soc. Press, 2000.

[162] S. Halevi. A plausible approach to computer-aided cryptographic proofs. Cryptology ePrint Archive, Report 2005/181, June 2005. Available at `http://eprint.iacr.org/2005/181`.

[163] C. Hall, I. Goldberg, and B. Schneier. Reaction Attacks Against Several Public-Key Cryptosystems. In *Proc. of ICICS '99*, LNCS, pages 2–12. Springer-Verlag, 1999.

[164] J. Håstad. Solving Simultaneous Modular Equations of Low Degree. *SIAM Journal of Computing*, 17:336–341, 1988.

[165] J. Heather, G. Lowe, and S. Schneider. How to prevent type flaw attacks on security protocols. In *Proc. 13th Computer Security Foundations Workshop (CSFW'01)*, pages 255–268. IEEE Comp. Soc. Press, 2000.

[166] R. Janvier, Y. Lakhnech, and L. Mazaré. Completing the picture: Soundness of formal encryption in the presence of active adversaries. In M. Sagiv, editor, *Proc. 14th European Symposium on Programming (ESOP'05)*, volume 3444 of *Lecture Notes on Computer Science*, pages 172–185, Edimbourg, U.K., Apr. 2005. Springer.

[167] A. Joux and R. Lercier. Improvements to the general Number Field Sieve for discrete logarithms in prime fields. *Mathematics of Computation*, 2000. to appear.

[168] M. Joye, J. J. Quisquater, and M. Yung. On the Power of Misbehaving Adversaries and Security Analysis of the Original EPOC. In *CT – RSA '01*, LNCS 2020, pages 208–222. Springer-Verlag, Berlin, 2001.

[169] M. Kallahalla, E. Riedel, R. Swaminathan, Q. Wang, and K. Fu. Plutus: Scalable secure file sharing on untrusted storage. In *2nd Conference on File and Storage Technologies (FAST'03)*, pages 29–42, San Francisco, CA, Apr. 2003. Usenix.

[170] KCDSA Task Force Team. The Korean Certificate-based Digital Signature Algorithm. Submission to IEEE P1363a. August 1998.
Available from `http://grouper.ieee.org/groups/1363/`.

[171] J. Kelsey, B. Schneier, and D. Wagner. Protocol interactions and the chosen protocol attack. In *Proc. 5th International Workshop on Security Protocols*, volume 1361 of *LNCS*, pages 91–104. Springer, 1997.

[172] H. Khurana and H.-S. Hahm. Certified mailing lists. In *Proceedings of the ACM Symposium on Communication, Information, Computer and Communication Security (ASIACCS'06)*, pages 46–58, Taipei, Taiwan, Mar. 2006. ACM.

[173] T. Kleinjung, K. Aoki, J. Franke, A. Lenstra, E. Thomé, J. Bos, P. Gaudry, A. Kruppa, P. Montgomery, D. A. Osvik, H. te Riele, A. Timofeev, and P. Zimmermann. Factorization of a 768-bit RSA modulus, 2010. Cryptology ePrint Archive 2010/006.

[174] D. E. Knuth and P. B. Bendix. Simple word problems in universal algebras. In J. Leech, editor, *Computational Problems in Abstract Algebra*, pages 263–297. Pergamon Press, Oxford, U.K., 1970.

[175] P. C. Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In *Crypto '96*, LNCS 1109, pages 104–113. Springer-Verlag, Berlin, 1996.

[176] P. C. Kocher, J. Jaffe, and B. Jun. Differential Power Analysis. In *Crypto '99*, LNCS 1666, pages 388–397. Springer-Verlag, Berlin, 1999.

[177] H. Krawczyk. SKEME: A versatile secure key exchange mechanism for internet. In *Internet Society Symposium on Network and Distributed Systems Security*, Feb. 1996. Available at `http://bilbo.isu.edu/sndss/sndss96.html`.

[178] S. Kremer and M. D. Ryan. Analysis of an electronic voting protocol in the applied pi-calculus. In M. Sagiv, editor, *Programming Languages and Systems — Proceedings of the 14th European Symposium on Programming (ESOP'05)*, volume 3444 of *Lecture Notes in Computer Science*, pages 186–200, Edinburgh, U.K., Apr. 2005. Springer.

[179] R. Küsters and T. Truderung. Reducing protocol analysis with XOR to the XOR-free case in the Horn theory based approach. In *Proceedings of the 15th ACM conference on Computer and communications security (CCS'08)*, pages 129–138, Alexandria, Virginia, USA, Oct. 2008. ACM.

[180] R. Küsters and T. Truderung. Using ProVerif to analyze protocols with Diffie-Hellman exponentiation. In *22nd IEEE Computer Security Foundations Symposium (CSF'09)*, pages 157–171, Port Jefferson, New York, USA, July 2009. IEEE.

[181] P. Laud. Handling encryption in an analysis for secure information flow. In P. Degano, editor, *Programming Languages and Systems, 12th European Symposium on Programming, ESOP'03*, volume 2618 of *Lecture Notes on Computer Science*, pages 159–173, Warsaw, Poland, Apr. 2003. Springer.

[182] P. Laud. Symmetric encryption in automatic analyses for confidentiality against active adversaries. In *IEEE Symposium on Security and Privacy*, pages 71–85, Oakland, California, May 2004.

[183] P. Laud. Secrecy types for a simulatable cryptographic library. In *12th ACM Conference on Computer and Communications Security (CCS'05)*, pages 26–35, Alexandria, VA, Nov. 2005. ACM.

[184] P. Laud and I. Tšahhirov. A user interface for a game-based protocol verification tool. In P. Degano and J. Guttman, editors, *6th International Workshop on Formal Aspects in Security and Trust (FAST2009)*, volume 5983 of *Lecture Notes on Computer Science*, pages 263–278, Eindhoven, Netherlands, Nov. 2009. Springer.

[185] P. Laud and V. Vene. A type system for computationally secure information flow. In M. Liśkiewicz and R. Reischuk, editors, *15th International Symposium on Fundamentals of Computation Theory (FCT'05)*, volume 3623 of *Lecture Notes on Computer Science*, pages 365–377, Lübeck, Germany, Aug. 2005. Springer.

[186] A. Lenstra and H. Lenstra. *The Development of the Number Field Sieve*, volume 1554 of *Lecture Notes in Mathematics*. Springer-Verlag, 1993.

[187] A. Lenstra and E. Verheul. Selecting Cryptographic Key Sizes. In *PKC '00*, LNCS 1751, pages 446–465. Springer-Verlag, Berlin, 2000.

[188] H. Lenstra. On the Chor-Rivest Knapsack Cryptosystem. *Journal of Cryptology*, 3:149–155, 1991.

[189] G. Lowe. An attack on the Needham-Schroeder public-key authentication protocol. *Information Processing Letters*, 56(3):131–133, 1996.

[190] G. Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In Margaria and Steffen, editors, *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 1055 of *Lecture Notes in Computer Science*, pages 147–166, 1996.

[191] G. 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.

[192] G. Lowe. Towards a completeness result for model checking of security protocols. *Journal of Computer Security*, 7(1), 1999.

[193] G. Lowe. Analysing protocols subject to guessing attacks. *Journal of Computer Security*, 12(1):83–98, 2004.

[194] K. D. Lux, M. J. May, N. L. Bhattad, and C. A. Gunter. WSEmail: Secure internet messaging based on web services. In *International Conference on Web Services (ICWS'05)*, pages 75–82, Orlando, Florida, July 2005. IEEE Computer Society.

[195] C. Lynch. Oriented equational logic programming is complete. *Journal of Symbolic Computation*, 21(1):23–45, 1997.

[196] J. Manger. A Chosen Ciphertext Attack on RSA Optimal Asymmetric Encryption Padding (OAEP) as Standardized in PKCS #1. In *Crypto '01*, LNCS 2139, pages 230–238. Springer-Verlag, Berlin, 2001.

[197] D. McAllester. Automatic recognition of tractability in inference relations. *J. ACM*, 40(2), 1993.

[198] C. Meadows and P. Narendran. A unification algorithm for the group Diffie-Hellman protocol. In *Workshop on Issues in the Theory of Security (WITS'02)*, Portland, Oregon, Jan. 2002.

[199] D. Micciancio and B. Warinschi. Completeness theorems for the abadi-rogaway language of encrypted expressions. *Journal of Computer Security*, 2004.

[200] D. Micciancio and B. Warinschi. Soundness of formal encryption in the presence of active adversaries. In M. Naor, editor, *Theory of Cryptography Conference (TCC'04)*, volume 2951 of *Lecture Notes on Computer Science*, pages 133–151, Cambridge, MA, USA, Feb. 2004. Springer.

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

[202] G. Miller. Riemann's Hypothesis and Tests for Primality. *Journal of Computer and System Sciences*, 13:300–317, 1976.

[203] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, i & ii. *Inf. Comput.*, 100(1):1–77, 1992.

[204] J. C. Mitchell, A. Ramanathan, A. Scedrov, and V. Teague. A probabilistic polynomial-time calculus for the analysis of cryptographic protocols. *Theoretical Computer Science*, 353(1–3):118–164, Mar. 2006.

[205] D. M'Raïhi, D. Naccache, D. Pointcheval, and S. Vaudenay. Computational Alternatives to Random Number Generators. In *Fifth Annual Workshop on Selected Areas in Cryptography (SAC '98)*, LNCS 1556, pages 72–80. Springer-Verlag, Berlin, 1998.

[206] M. Naor and M. Yung. Public-Key Cryptosystems Provably Secure against Chosen Ciphertext Attacks. In *Proc. of the 22nd STOC*, pages 427–437. ACM Press, New York, 1990.

[207] V. I. Nechaev. Complexity of a Determinate Algorithm for the Discrete Logarithm. *Mathematical Notes*, 55(2):165–172, 1994.

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

[209] R. M. Needham and M. D. Schroeder. Authentication revisited. *Operating Systems Review*, 21(1):7, 1987.

[210] J. B. Nielsen. Separating Random Oracle Proofs from Complexity Theoretic Proofs: The Non-committing Encryption Case. In *Crypto '02*, LNCS 2442, pages 111–126. Springer-Verlag, Berlin, 2002.

[211] NIST. Digital Signature Standard (DSS). Federal Information Processing Standards PUBlication 186, November 1994.

[212] NIST. Secure Hash Standard (SHS). Federal Information Processing Standards PUBlication 180–1, April 1995.

[213] D. Nowak. A framework for game-based security proofs. In *Information and Communications Security, 9th International Conference, ICICS 2007*, volume 4861 of *Lecture Notes on Computer Science*, pages 319–333, Zhengzhou, China, Dec. 2007. Springer.

[214] D. Nowak. On formal verification of arithmetic-based cryptographic primitives. In *Information Security and Cryptology - ICISC 2008, 11th International Conference*, volume 5461 of *Lecture Notes on Computer Science*, pages 368–382, Seoul, Korea, Dec. 2008. Springer.

[215] K. Ohta and T. Okamoto. On Concrete Security Treatment of Signatures Derived from Identification. In *Crypto '98*, LNCS 1462, pages 354–369. Springer-Verlag, Berlin, 1998.

[216] T. Okamoto and D. Pointcheval. REACT: Rapid Enhanced-security Asymmetric Cryptosystem Transform. In *CT – RSA '01*, LNCS 2020, pages 159–175. Springer-Verlag, Berlin, 2001.

[217] T. Okamoto and D. Pointcheval. The Gap-Problems: a New Class of Problems for the Security of Cryptographic Schemes. In *PKC '01*, LNCS 1992. Springer-Verlag, Berlin, 2001.

[218] D. Otway and O. Rees. Efficient and timely mutual authentication. *Operating Systems Review*, 21(1):8–10, 1987.

[219] L. C. Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Computer Security*, 6(1–2):85–128, 1998.

[220] D. Pointcheval and J. Stern. Security Proofs for Signature Schemes. In *Eurocrypt '96*, LNCS 1070, pages 387–398. Springer-Verlag, Berlin, 1996.

[221] D. Pointcheval and J. Stern. Security Arguments for Digital Signatures and Blind Signatures. *Journal of Cryptology*, 13(3):361–396, 2000.

[222] D. Pointcheval and S. Vaudenay. On Provable Security for Digital Signature Algorithms. Technical Report LIENS-96-17, LIENS, October 1996.

[223] J. M. Pollard. Monte Carlo Methods for Index Computation (mod p). *Mathematics of Computation*, 32(143):918–924, July 1978.

[224] C. Rackoff and D. R. Simon. Non-Interactive Zero-Knowledge Proof of Knowledge and Chosen Ciphertext Attack. In *Crypto '91*, LNCS 576, pages 433–444. Springer-Verlag, Berlin, 1992.

[225] R. Ramanujam and S. P. Suresh. Tagging makes secrecy decidable for unbounded nonces as well. In *Proc. 23rd Conference on Foundations of Software Technology and Theoretical Computer Science (FST&TCS'03)*, volume 2914 of *LNCS*, pages 363–374. Springer, 2003.

[226] R. Ramanujam and S. P. Suresh. Decidability of context-explicit security protocols. *Journal of Computer Security*, 13(1):135–165, 2005.

[227] R. Rivest. The MD5 Message-Digest Algorithm. RFC 1321, The Internet Engineering Task Force, April 1992.

[228] R. Rivest, A. Shamir, and L. Adleman. A Method for Obtaining Digital Signatures and Public Key Cryptosystems. *Communications of the ACM*, 21(2):120–126, February 1978.

[229] RSA Data Security, Inc. Public Key Cryptography Standards – PKCS.
Available from `http://www.rsa.com/rsalabs/pubs/PKCS/`.

[230] M. Rusinowitch and M. 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.

[231] 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.

[232] C. P. Schnorr. Efficient Identification and Signatures for Smart Cards. In *Crypto '89*, LNCS 435, pages 235–251. Springer-Verlag, Berlin, 1990.

[233] C. P. Schnorr. Efficient Signature Generation by Smart Cards. *Journal of Cryptology*, 4(3):161–174, 1991.

[234] C. P. Schnorr and M. Jakobsson. Security of Signed ElGamal Encryption. In *Asiacrypt '00*, LNCS 1976, pages 458–469. Springer-Verlag, Berlin, 2000.

[235] D. Shanks. Class Number, a Theory of Factorization, and Genera. In *Proceedings of the Symposium on Pure Mathematics*, volume 20, pages 415–440. AMS, 1971.

[236] H. Shimizu. On the Improvement of the Håstad Bound. In *1996 IEICE Fall Conference*, Volume A-162, 1996. In Japanese.

[237] V. Shoup. Lower Bounds for Discrete Logarithms and Related Problems. In *Eurocrypt '97*, LNCS 1233, pages 256–266. Springer-Verlag, Berlin, 1997.

[238] V. Shoup. OAEP Reconsidered. In *Crypto '01*, LNCS 2139, pages 239–259. Springer-Verlag, Berlin, 2001. Also appeared in the Cryptology ePrint Archive 2000/060. November 2000.
Available from `http://eprint.iacr.org/`.

[239] V. Shoup. OAEP Reconsidered. *Journal of Cryptology*, 15(4):223–249, September 2002.

[240] V. Shoup. Sequences of games: a tool for taming complexity in security proofs, 2004. Cryptology ePrint Archive 2004/332.

[241] G. Smith and R. Alpízar. Secure information flow with random assignment and encryption. In *4th ACM Workshop on Formal Methods in Security Engineering (FMSE'06)*, pages 33–43, Alexandria, Virginia, Nov. 2006.

[242] C. Sprenger, M. Backes, D. Basin, B. Pfitzmann, and M. Waidner. Cryptographically sound theorem proving. In *19th IEEE Computer Security Foundations Workshop (CSFW-19)*, pages 153–166, Venice, Italy, July 2006. IEEE.

[243] J. Stern, D. Pointcheval, J. Malone-Lee, and N. Smart. Flaws in Applying Proof Methodologies to Signature Schemes. In *Crypto '02*, LNCS 2442, pages 93–110. Springer-Verlag, Berlin, 2002.

[244] I. Tšahhirov and P. Laud. Application of dependency graphs to security protocol analysis. In G. Barthe and C. Fournet, editors, *3rd Symposium on Trustworthy Global Computing (TGC'07)*, volume 4912 of *Lecture Notes on Computer Science*, pages 294–311, Sophia-Antipolis, France, Nov. 2007. Springer.

[245] Y. Tsiounis and M. Yung. On the Security of El Gamal based Encryption. In *PKC '98*, LNCS. Springer-Verlag, Berlin, 1998.

[246] S. Vaudenay. Cryptanalysis of the Chor-Rivest Scheme. In *Crypto '98*, LNCS 1462, pages 243–256. Springer-Verlag, Berlin, 1998.

[247] B. Warinschi. A computational analysis of the Needham-Schröeder-(Lowe) protocol. In *Proceedings of the 16th IEEE Computer Security Foundations Workshop (CSFW'03)*. IEEE Computer Society, 2003.

[248] C. Weidenbach. Towards an automatic analysis of security protocols in first-order logic. In H. Ganzinger, editor, *16th International Conference on Automated Deduction (CADE-16)*, volume 1632 of *Lecture Notes in Artificial Intelligence*, pages 314–328, Trento, Italy, July 1999. Springer.

[249] T. Y. C. Woo and S. S. Lam. Authentication for distributed systems. *Computer*, 25(1):39–52, Jan. 1992.

[250] T. Y. C. Woo and S. S. Lam. A semantic model for authentication protocols. In *Proceedings IEEE Symposium on Research in Security and Privacy*, pages 178–194, Oakland, California, May 1993.

[251] T. Y. C. Woo and S. S. Lam. Authentication for distributed systems. In D. Denning and P. Denning, editors, *Internet Besieged: Countering Cyberspace Scofflaws*, pages 319–355. ACM Press and Addison-Wesley, Oct. 1997.