Cryptographic protocols
Formal and Computational Proofs

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

September 30, 2009

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

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 [3], using formal methods. This attack is completely independent of the security of cryptographic primitives and of the actual implementation: the attack works even for perfect cryptography and a faithful implementation.

The protocol was corrected by G. Lowe himself [4]. The new protocol has been proved secure in a formal model by several authors. However, there is still an attack on this protocol, which we present next. The idea of the attack was suggested by J. Millen and is reported by B. Warinschi in [7].

The reason of the apparent paradox (the protocol is proved secure and there is an attack) is the discrepancy between the models that are used for the proof and for the attack respectively. More precisely, the security proof (that we will also give in these notes) assume a *perfect cryptography.*

Such a perfect cryptography does not exist in practice. Based on some hardness assumptions, there are however many "provably secure" encryption schemes. In our example, we are using public-key cryptography and the attack on the corrected protocol is based on a property of the El-Gamal encryption scheme. The El-Gamal public-key encryption scheme is provably secure for *chosen plaintext attacks (IND-CPA)* [6] (assuming the hardness of some discrete logarithm problem). However, it is not secure for some stronger notions of security, for instance the *Chosen Ciphertext Attacks (IND-CCA).* And it turns out that Lowe's protocol model for public key cryptography is only sound for IND-CCA secure encryption schemes.

In summary: we have two models; in the more abstract one there is a security proof, while in the less abstract one there is an attack. The two models are actually equivalent, provided that the encryption scheme is secure in a strong sense.

Similar problems may occur in actual implementations: a protocol can be proven secure while there is an attack on its actual implementation. Similarly, the soundness of the abstract proof of security requires some properties (typically memory access) of the implementations.

The ambition of this lecture is to cover all these aspects:

- Give a formal symbolic model for security protocols and proof techniques at this level of abstraction.
- Give a computational model of security protocols and proof techniques at this level.
- Provide with soundness results, showing under which assumptions symbolic proofs provide with computational security.
- Provide with correctness results of implementations, that show under which assumptions either the formal or the computational proof yields a security proof of the implementation.

1.1 An informal description of the Needham-Schroeder protocol

The protocol is a so-called "mutual authentication protocol": two parties A and B wish to agree on some value. For instance A and B wish to establish a shared secret that they will use later for fast confidential communication.

A and B only use a public communication channel (for instance the postal service, or the internet or the mobile phone). The transport of the messages on such channels is insecure: 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 messages, 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 key k is written $\{m\}_k$ (later we may wish to distinguish the algorithm that is used and add a third argument for a random input).

First A encrypts a fresh message M_A and her identity with the public-key of B and sends it on the public channel:

$$A \to B : \qquad \{A, M_A\}_{\mathsf{pk}(B)}$$

Only B, who owes the key can open this message. Upon reception, he gets M_A, generates his own message M_B and sends back the pair encrypted with the public key of A:

$$B \to A : \qquad \{M_A, M_B\}_{\mathsf{pk}(A)}$$

Only A is able to open this message. Furthermore, since only B was able to get M_A, inserting M_A in the plaintext is a witness that it comes from B. Finally, A, after decrypting, checks that the first component is M_A indeed and retrieves the second component M_B. As an acknowledgement, she sends back M_B encrypted by the public key of B:

$$A \to B : \qquad \{M_B\}_{\mathsf{pk}(B)}$$

When B receives this message, he checks that the content is M_B indeed. If this succeeds, it is claimed that, if A, B are honest, then both parties agreed on M_A and M_B (they share these values) and that noone else knows M_A, M_B.

This fails, because this protocol can be used by several parties. Assume that C is a dishonest agent and that A starts the protocol with C:

$$A \to C : \qquad \{A, M_A\}_{\mathsf{pk}(C)}$$

Then C, impersonating A, sends a message to B, starting another instance of the protocol:

$$C \to B : \qquad \{A, M_A\}_{\mathsf{pk}(B)}$$

When B receives this message, supposedly coming from A, he sends back:

$$B \to A : \qquad \{M_A, M_B\}_{\mathsf{pk}(A)}$$

A believes that this is the reply of C, hence continues:

$$A \to C : \qquad \{M_B\}_{\mathsf{pk}(C)}$$

Finally, C sends the expected reply to B:

$$C \to B : \qquad \{M_B\}_{\mathsf{pk}(B)}$$

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 M_B, nor M_B is a secret shared between A and B.

This attack cannot be mounted in the corrected version of the protocol, that can be summarized as follows:

$$\begin{array}{ll} A \to B : & \{A, M_A\}_{\mathsf{pk}(B)} \\ B \to A : & \{M_A, M_B, B\}_{\mathsf{pk}(A)} \\ A \to B : & \{M_B\}_{\mathsf{pk}(B)} \end{array}$$

In what follows, we will refer to the first instance as NS-protocol and to the second instance as NSL-protocol.

1.2 A more formal analysis

We have to make more precise the view of each agent. This amounts specifying the concurrent programs that are executed by each party.

"A" is actually a program executed by the agent a:

$$\begin{array}{lll} A(a,b) = & \nu M. & a \texttt{ generates a fresh message } M \\ & \overline{c}(\{M, a\}_{\mathsf{pk}(b)}). & \texttt{the message is sent on the channel } c \\ & c(x). & a \texttt{ waits for a message } x \texttt{ on channel } c \\ & \mathsf{let}\ x_0 = \mathsf{dec}(x, \mathsf{sk}(a))\ \mathsf{in} & a \texttt{ tries to decrypt the message with her} \\ & & \quad \texttt{secret key. In case of failure, she aborts} \\ & \mathsf{if}\ \pi_1(x_0) = M\ \mathsf{then} & a \texttt{ checks that the first component} \\ & & \quad \texttt{is her nonce } M\texttt{. If not, she aborts.} \\ & \mathsf{let}\ x_1 = \pi_2(x_0)\ \mathsf{in} & a \texttt{ retrieves the second component} \\ & \overline{c}(\{x_1\}_{\mathsf{pk}(b)}) & \end{array}$$

Note that we use here 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.

Similarly, the B process may look like this:

$$\begin{array}{ll} B(a,b) = & c(y).\ \mathsf{let}\ (a, y_0) = \mathsf{dec}(y, \mathsf{sk}(b))\ \mathsf{in} \\ & \nu M'.\ \overline{c}(\{y_0, M'\}_{\mathsf{pk}(a)}) \\ & c(y').\ \mathsf{if}\ \mathsf{dec}(y', \mathsf{sk}(b)) = M'\ \mathsf{then}\ \mathsf{Ok}. \end{array}$$

We give a sketch of the operational semantics of such processes a little later.

Any number of copies of A and B (with any parameter values) are running concurrently in a hostile environment. Such a hostile environment is modeled 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 keys (besides those that it generates) are known.

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, M' is unknown to the environment. We will also see later how to formalize 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.

We exhibit now a process that will yield the attack, assuming that d is a dishonest (or compromised) agent who leaked his secret key:

$$\begin{aligned} P = \quad & c(z).\ \mathsf{let}\ (a, z_0) = \mathsf{dec}(z, \mathsf{sk}(d))\ \mathsf{in} \\ & \overline{c}(\{a, z_0\}_{\mathsf{pk}(b)}). \\ & c(z').\ \overline{c}(z'). \\ & c(z'').\ \mathsf{let}\ z_1 = \mathsf{dec}(z'', \mathsf{sk}(d))\ \mathsf{in} \\ & \overline{c}(\{z_1\}_{\mathsf{pk}(b)}) \end{aligned}$$

In this sketched formal model, the NSL protocol is secure.

1.3 An attack on the NSL protocol

Up to now, the encryption is a black-box: nothing can be learned 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 $\{m\}_{\mathsf{pk}(a)} = \left\langle \mathsf{pk}(a)^r \times g^{m'}, g^r \right\rangle$. Decrypting the message consists in raising g^r to the power $\mathsf{sk}(a)$ and dividing the first component of the pair by $g^{r \times \mathsf{sk}(a)}$.

Assume now that we are using such an encryption scheme in the NSL 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 m'_1, m'_2).

The attacker 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:

$$a \xrightarrow{\{a,N\}_{\mathsf{pk}(d)}} d \xrightarrow{\{a,N\}_{\mathsf{pk}(b)}} b \xrightarrow{\{N,N',b\}_{\mathsf{pk}(a)}}$$

Now, the attacker intercepts this message $(\mathsf{pk}(a)^r \times g^{N + 2^\alpha \times N' + 2^{2\alpha} \times b}, g^r)$ where α is the length of a nonce (random numbers N, N'). The attacker knows g, α, b, hence can compute $g^{-2^{2\alpha} b} \times g^{2^{2\alpha} d}$ and multiply the first component, yielding $\{N, N', d\}_{\mathsf{pk}(a)}$. Then the attack can go on as before: a replies by sending $\{N'\}_{\mathsf{pk}(d)}$ and the attacker sends $\{N'\}_{\mathsf{pk}(b)}$ to b, impersonating a.

There are other examples, based on RSA and factorization [5], from which similar attacks could be mounted

This example is however a toy example since pairing could be implemented in another way. In [7] there is a real attack (independent on how pairing is implemented), based on weaknesses of El Gamal encryption scheme of the NSL protocol. This shows however 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 Some other simple examples

Here are some simple examples from [2].

Exercise 1
Consider the simple protocol:

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

M is generated by A and should be kept secret in any session between two honest parties.
Show that there is a very simple (man-in-the-middle) attack

Example 1 *The protocol is corrected as in the NS protocol:*

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

Exercise 2
For double security, all messages in the previous protocol are encrypted twice:

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

Show that the protocol then becomes insecure.

2 A small process calculus

We describe here a small process calculus and sketch its operational semantics. It is inspired by the applied π-calculus of [1]. It is a fragment of the calculus considered in the tool ProVerif. We will enrich the calculus later on.

2.1 Terms and equations

Messages are represented by *terms* that are built on a set of function symbols $\mathcal{F}$, each with a fixed arity and *names* out of a set $\mathcal{N}$. Names are considered as arity 0 function symbols (constants). In our examples we consider

$$\mathcal{F} = \{\langle \_, \_\rangle, \{\_\}^p_\_, \{\_\}^s_\_, \mathsf{sign}(\_, \_), \mathsf{pk}(\_), \mathsf{sk}(\_), \mathsf{vk}_\_, \pi_1(\_), \pi_2(\_), \mathsf{sdec}(\_, \_), \mathsf{pdec}(, \_), \mathsf{check}(\_, \_)\}$$

$T(\mathcal{F} \cup \mathcal{N})$ is the set of such messages. In addition, we may consider messages containing unknown (unspecified) pieces, that are represented using variables. Such templates are terms with variables in the set $T(\mathcal{F} \cup \mathcal{N} \cup \mathcal{X})$. *Ground terms* are terms without variables.

The messages are actually a quotient algebra; they satisfy a set $\mathcal{E}$ of equations. In our examples, we typically use the following equations:

$$\begin{array}{rclrcl} \pi_1(\langle x, y\rangle) & = & x & \mathsf{sdec}(\{x\}^s_y, y) & = & x \\ \pi_2(\langle x, y\rangle) & = & y & \mathsf{pdec}(\{x\}^p_{\mathsf{pk}(y)}, \mathsf{sk}(y)) & = & x \\ \mathsf{check}(\mathsf{sign}(x, y), \mathsf{vk}_y) & = & x & & & \end{array}$$

This set of equations can be oriented from left to right into a convergent term rewriting system. This means that every term t has a unique normal form $t\downarrow$.

Finally, we split the function symbols into constructors and destructors and between private and public symbols.

Constructors are $\mathcal{C} = \{\langle\_,\_\rangle, \{\_\}^p_\_, \{\_\}^s_\_, \mathsf{sign}(\_,\_), \mathsf{pk}(\_), \mathsf{sk}(\_), \mathsf{vk}\_\}$ and destructors are $\mathcal{D} = \{\pi_1(\_), \pi_2(\_), \mathsf{sdec}(\_,\_), \mathsf{pdec}(,\_), \mathsf{check}(\_,\_)\}$. A *constructor term* is a term in $T(\mathcal{C} \cup \mathcal{N} \cup \mathcal{X})$.

Non constructor ground terms in normal form can be considered as junk: this means that some operation failed.

2.2 Conditions

Conditions are Boolean combinations of atomic formulas $P(t_1, \ldots, t_n)$ where P is a predicate symbol and $t_1, \ldots, t_n$ are terms of $T(\mathcal{F} \cup \mathcal{N} \cup \mathcal{X})$.

There might be as many predicate symbols as needed, whose interpretation must be a recursive relation on terms. We consider in our examples a single predicate symbol $=$, whose interpretation is the expected one: if σ is a substitution that maps the free variables of s, t to ground terms, $\sigma \models s = t$ iff $s\sigma\downarrow$ is identical to $t\sigma\downarrow$.

For example, $\{x \mapsto \{n\}^s_k\} \models \mathsf{sdec}(x, k) = n$.

2.3 Simple threads

For simplicity we consider first some simple threads:

$$
\begin{array}{lcll}
T & ::= & \mathbf{0} & \text{the terminated process} \\
 & | & \mathsf{in}(c, s).T & \text{input of a message matching } s \text{ on channel } c \\
 & | & \mathsf{out}(c, t).T & \text{output of a message } t \text{ on channel } c \\
 & | & \mathsf{if}\ \phi\ \mathsf{then}\ T\ \mathsf{else}\ T & \text{conditional branching}
\end{array}
$$

c is a channel name.

s is a pattern: a constructor term with variables. These variables are bound by the actual input term. The terms t are any constructor terms whose variables are bound before. Formally, we define inductively the set of general threads T with bound variables $B(T)$ and free variables $F(T)$; the threads are processes without free variables.

- $\mathbf{0}$ is a general thread and $B(\mathbf{0}) = \emptyset$, $F(\mathbf{0}) = \emptyset$
- if T is a general thread and t is a constructor term such that $B(T) \cap \mathcal{V}(t) = \emptyset$, then $\mathsf{in}(c, t).T$ is a general thread and $B(\mathsf{in}(c, t).T) = \mathcal{V}(t) \cup B(T)$, $F(\mathsf{in}(c, t).T) = F(T) \setminus \mathcal{V}(t)$
- if T is a general thread and t is a term, then $\mathsf{out}(c, t).T$ is a general thread with $F(\mathsf{out}(c, t).T) = \mathcal{V}(t) \cup F(T)$ and $B(\mathsf{out}(c, t).T) = B(T)$.
- if T_1, T_2 are general threads, and $\mathcal{V}(\phi) \cap (B(T_1) \cup B(T_2)) = \emptyset$, then $T = \mathsf{if}\ \phi\ \mathsf{then}\ T_1\ \mathsf{else}\ T_2$ is a general thread and $F(T) = \mathcal{V}(\phi) \cup F(T_1) \cup F(T_2)$ and $B(T) = B(T_1) \cup B(T_2)$

In this definition, we consider both pattern inputs and conditionals, which is redundant: 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 $P_A(a,b,n_a) = \mathit{out}(c, \{a, n_a\}_{\mathsf{pk}(b)}).\mathit{in}(c, \{n_a, x\}_{\mathsf{pk}(a)}).\mathit{out}(c, \{x\}_{\mathsf{pk}(b)}).\boldsymbol{0}$

$P_B(a,b,n_b) = \mathit{in}(c, \{a, x\}_{\mathsf{pk}(b)}).\mathit{out}(c, \{x, n_b\}_{\mathsf{pk}(b)}).\mathit{in}(c, \{n_b\}_{\mathsf{pk}(b)}).\boldsymbol{0})$

In the alternative formalism of explicit destructors (that we will eventually use), the same example becomes:

Example 3

$$\begin{array}{rl} P_A(a,b,n_a) = & \mathit{out}(c, \{a, n_a\}_{\mathsf{pk}(b)}).\mathit{in}(c, x). \\ & \mathit{if}\ \pi_1(\mathit{dec}(x, \mathsf{sk}(a))) = n_a \\ & \quad \mathit{then}\ \mathit{out}(c, \{\pi_2(\mathit{dec}(x, \mathsf{sk}(a))\}_{\mathsf{pk}(b)}).\boldsymbol{0} \end{array}$$

$$\begin{array}{rl} P_B(a,b,n_b) = & \mathit{in}(c, y). \\ & \mathit{if}\ \pi_1(\mathit{dec}(y, \mathsf{sk}(b))) = a \\ & \quad \mathit{then}\ \mathit{out}(c, \{\pi_2(\mathit{dec}(y, \mathsf{sk}(b))), n_b\}_{\mathsf{pk}(b)}).\mathit{in}(c, z). \\ & \qquad \mathit{if}\ \mathit{dec}(y, \mathsf{sk}(b)) = n_b\ \mathit{then}\ \boldsymbol{0} \end{array}$$

Exercise 3
Give a reasonable formalisation of the protocols of section 1.4.

2.4 Processes

We now compose the threads and embed them in a hostile environment.

$$\begin{array}{rcll} P & ::= & T & \\ & | & P \| Q & \\ & | & \nu n.P & \\ & | & !P & \\ & | & \mathsf{out}(c, s).\ P & \text{if } s \text{ is a ground constructor term.} \end{array}$$

We could (will) also allow more constructions later on, adding convenient constructions and also events that will allow to define more security properties.

Example 4 *Coming back to the NS protocol, we may define several execution scenarii:*

- *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:*

$$(\nu n_a.\ P_A(a, d, n_a)) \| (\nu n_b.\ P_B(a, b, n_b)) \| \mathit{out}(c, \mathsf{sk}(d)).\boldsymbol{0}$$

- *Now, we wish a to execute P_A, 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.*

$$(\nu n_a.\ \mathit{in}(c, x_b).\ P_A(a, x_b, n_a)) \| (\nu n_b.\ \mathit{in}(c, x_a).\ P_B(x_a, b, n_b)) \| \mathit{out}(c, \mathsf{sk}(d)).\boldsymbol{0}$$

$$\frac{x \quad y}{\langle x, y\rangle}\,P \qquad \frac{x \quad y}{\{x\}^s_y}\,E \qquad \frac{x \quad \mathsf{pk}(y)}{\{x\}^p_{\mathsf{pk}(y)}}\,PKE \qquad \frac{x}{\mathsf{pk}(x)}\,PK$$

$$\frac{x \quad y}{\mathsf{sign}(x, y)}\,S \qquad \frac{\langle x, y\rangle}{x}\,\pi_1 \qquad \frac{\langle x, y\rangle}{y}\,\pi_2 \qquad \frac{\{x\}^s_y \quad y}{x}\,D$$

$$\frac{\{x\}^p_{\mathsf{pk}(y)} \quad \mathsf{sk}(y)}{x}\,PKD \qquad \frac{\mathsf{sign}(x, y)}{x}\,C$$

Figure 1: Intruder deduction rules

- *In the previous examples a was only able to engage the protocol once (and b was only able to engage once in a response). We may wish a and b be able to execute any number of instances of P_A and P_B:*

$$!((\nu n_a.\ \mathit{in}(c, x_b).\ P_A(a, x_b, n_a))\|(\nu n_b.\ \mathit{in}(c, x_a).\ P_B(x_a, b, n_b)))\|\mathit{out}(c, \mathsf{sk}(d)).\mathbf{0}$$

- *Finally, in general, P_A could be executed by any agent, including b and P_B 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:*

$$\begin{array}{l} !\ \ (\ \nu a.\nu b.\ \mathit{out}(c, a).\ \mathit{out}(c, b). \\ \qquad !((\nu n_a.\ \mathit{in}(c, x_b).\ P_A(a, x_b, n_a)) \\ \qquad\quad \|(\nu n_b.\ \mathit{in}(c, x_a).\ P_B(x_a, b, n_b)))) \\ \|!(\nu d.\mathit{out}(c, d).\ \mathit{out}(c, \mathsf{sk}(d)).\mathbf{0}) \end{array}$$

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

Exercise 4
In this last scenario of the above example, there is actually a missing case. What is missing ? How is it possible to fix this ?

2.5 Intruder capabilities

Frames record the sequences of messages, together with the hidden information in these messages. We write a frame $\nu\overline{n}.\sigma$ where $\overline{n}$ is a sequence of (hidden) names and σ is a substitution.

There are several possible ways of defining the intruder capabilities, we choose here the "implicit destructors" formulation, in which the destructors do not appear. The rules are displayed in figure 1.

A *proof of t with hypotheses H* built using these inference rules is a tree Π labeled with terms such that

- the labels of the leaves of Π belong to H
- the root is labeled with t

- For any subtree Π' of Π whose root is labeled t' and sons of the root are labeled $t_1, \ldots, t_n$, there is an inference rule whose an instance is $\dfrac{t_1 \quad \cdots \quad t_n}{t'}$.

$\mathsf{Hyp}(\Pi)$ is the set of labels of the leaves of 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. Given a set of terms T, $\mathsf{St}(T)$ is the union of the subterms of terms in T. The number of elements in $\mathsf{St}(T)$ is also the measure of the size of T, as this is (up to a constant factor) the size of a DAG representation of T.

We will only use the rules of figure 1 in the next section. However, for equivalence properties, that will be considered later in this lecture, we need to extend the deduction relation to frames. Basically, the names that occur free in a frame can be seen from the outside:

$$\frac{\nu\overline{n}.\ \phi}{m} \text{ if } m \in \mathcal{N} \setminus \overline{n}$$

and any term that is part of the frame is deducible:

$$\frac{\nu\overline{n}.\ \sigma}{x_i\sigma} \text{ if } x_i \in \mathsf{Dom}(\sigma)$$

Then, we may confuse the frame with the hypotheses of the proof.

An example: $\nu a, \nu b.\ \left\langle \{s\}^s_{\langle a,b\rangle}, a \right\rangle, \{b\}^s_a \vdash s$,

$$\dfrac{\dfrac{\left\langle \{s\}^s_{\langle a,b\rangle}, a \right\rangle}{\{s\}^s_{\langle a,b\rangle}} \qquad \dfrac{\dfrac{\left\langle \{s\}^s_{\langle a,b\rangle}, a \right\rangle}{a} \qquad \dfrac{\{b\}^s_a \qquad \dfrac{\left\langle \{s\}^s_{\langle a,b\rangle}, a \right\rangle}{a}}{b}}{\langle a, b\rangle}}{s}$$

Similarly, $\nu a.\ \left\langle \{s\}^s_{\langle a,b\rangle}, a \right\rangle \vdash s$,

The rules of figure 1 define a deduction relation $\vdash$ that enjoys good properties.

Theorem 2.1 *$\vdash$ is decidable in linear time. Deciding the membership to $\vdash$ is PTIME-complete (under LOGSPACE-reductions).*

We use the property that HORN-SAT is decidable in linear time and is PTIME-complete. The equivalence with HORN-SAT relies on the following lemma, that states some "subformula property".

Let us call a *constructor rule* an inference rule that ends with a construction (either a pairing or an encryption). Let *destructor rules* be the other inference rules.

Definition 2.2 *A proof π of $H \vdash t$ is* local *if, for any subproof π' of π,*

-
  - *either the last rule of π' is a constructor rule and $\mathit{Conc}(\pi') \in \mathit{St}(H) \cup \mathit{St}(\mathit{Conc}(\pi))$*
  - *or else $\mathit{Conc}(\pi') \in \mathit{St}(H)$.*

- *for any subproof π'' of π', $\mathit{Conc}(\pi') \neq \mathit{Conc}(\pi'')$.*

Lemma 2.3 *If there is a proof of t with hypotheses H, then there is a local proof of $H \vdash t$. In particular, this proof π satisfies $\mathsf{Steps}(\pi) \subseteq \mathsf{St}(H) \cup \mathsf{St}(t)$.*

We prove the lemma in case of symmetric encryption and pairing.

Proof:. We prove, by induction on the size of proof Π of $H \vdash t$ that there is a local proof of $H \vdash t$.

In the base case, the proof is reduced to a leaf, which is a local proof.

For the induction step, let

$$\Pi = \cfrac{\begin{array}{ccc}\pi_1 & & \pi_n \\ t_1 & \cdots & t_n\end{array}}{t}\ R$$

If $t \in \mathsf{Steps}(\pi_1) \cup \ldots \cup \mathsf{Steps}(\pi_n)$, we replace Π with this strict subproof of t and apply the induction hypothesis..

Otherwise, if R is a constructor rule, then $t_1, \ldots, t_n$ are subterms of t and it is sufficient to apply the induction hypothesis.

If R is a projection rule (say π_1), then $t_1 = \langle t, u \rangle$. There are then three cases:

1. $t_1 \in H$ and then $t \in \mathsf{St}(H)$.
2. the last rule in the proof π_1 of t_1 is a destructor rule and, by induction hypothesis, $\mathsf{Steps}(\pi_1) \subseteq \mathsf{St}(H)$, hence $\mathsf{Steps}(\Pi) = \{t\} \cup \mathsf{Steps}(\Pi_1) \subseteq \mathsf{St}(H)$
3. the last rule of the proof π_1 of t_1 is a constructor rule. Then (because the top symbol of t_1 is a pair), it must be a pairing:

$$\Pi_1 = \cfrac{\begin{array}{ccc}\vdots & & \vdots \\ t & \cdots & u\end{array}}{\langle t, u \rangle}$$

Then, there is a proof of t, which is strictly smaller than Π, a case we already considered

If R is a decryption rule, then $t_1 = \{t\}^s_u$. Again, if either $t_1 \in H$ or t_1 is obtained by a destructor rule, we immediately get $t \in \mathsf{St}(H)$ (by induction hypothesis). If the proof of t_1 is ending with a constructor rule, it must be a symmetric encryption. Then, as above, there is a smaller proof of t, to which we can apply the induction hypothesis. □