Computational soundness of observational equivalence

Hubert Comon-Lundh, Véronique Cortier

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Computational soundness of observational equivalence

Hubert Comon-Lundh — Véronique Cortier

N° 6508

Avril 2008

Thème SYM

ISSN 0249-6399 ISRN INRIA/RR--6508--FR+ENG

Computational soundness of observational equivalence

Hubert Comon-Lundh*, Véronique Cortier†

Thème SYM — Systèmes symboliques
Équipes-Projets Cassis et Secsi

Rapport de recherche n° 6508 — Avril 2008 — 36 pages

Abstract: Many security properties are naturally expressed as indistinguishability between two versions of a protocol. In this paper, we show that computational proofs of indistinguishability can be considerably simplified, for a class of processes that covers most existing protocols. More precisely, we show a soundness theorem, following the line of research launched by Abadi and Rogaway in 2000: computational indistinguishability in presence of an active attacker is implied by the *observational equivalence* of the corresponding symbolic processes.
Up to our knowledge, the only result of this kind is Adão and Fournet [7], in which, however, cryptographic primitives are not part of the syntax. Otherwise, previous works either considered a passive attacker, or, in case of active attackers, proved a soundness result for properties that can be defined on execution traces of the protocol. Anonymity for instance does not fall in the latter category.
We prove our result for symmetric encryption, but the same techniques can be applied to other security primitives such as signatures and public-key encryption. The proof requires the introduction of new concepts, which are general and can be reused in other settings.

Key-words: computational soundness, cryptographic protocols, verification, communicating processes

This work has been partially supported by the grants ARA FormaCrypt and ANR AVOTÉ.

* ENS Cachan and Research Center for Information Security (RCIS), National Institute of Advanced Industrial Science and Technology (AIST)
† LORIA, CNRS & INRIA project Cassis

Centre de recherche INRIA Nancy – Grand Est
LORIA, Technopôle de Nancy-Brabois, Campus scientifique,
615, rue du Jardin Botanique, BP 101, 54602 Villers-Lès-Nancy
Téléphone : +33 3 83 59 30 00 — Télécopie : +33 3 83 27 83 19

Correction calculatoire de l'équivalence observationelle

Résumé : De nombreuses propriétés de sécurité s'expriment naturellement comme l'*indistingabilité* (c'est-à-dire l'impossibilité de distinguer) deux versions d'un protocole. Dans ce document, nous montrons que les preuves calculatoires de l'indistingabilité peuvent être considérablement simplifiées, pour une classe de processus qui couvrent la plupart des protocoles existants. Plus précisément, nous prouvons un théorème de correction - dans la lignée des travaux inités par Abadi et Rogaway en 2000: l'indistingabilité calculatoire en présence d'un intrus actif est assurée par l'*équivalence observationelle* des processus symboliques correspondants.
À notre connaissance, le seul résultat de cette sorte est celui de Adão et Fournet [7]. Cependant, dans cet article, la syntaxe ne comprend pas de primitives cryptographiques. En dehors de ce résultat, les travaux existants considèrent un attaquant passif ou, dans le cas d'attaquants actifs, obtiennent des résultats de correction pour des propriétés qui peuvent être définies sur les traces d'exécution d'un protocole; ce qui n'est pas le cas de l'anonymat par exemple.
Nous prouvons notre résultat pour le chiffrement asymétrique mais les mêmes techniques pourraient être appliquées aux autres primitives de sécurité telles que la signature ou le chiffrement à clef publique. Les techniques de preuves ont demandé l'introduction de nouveaux concepts généraux qui pourront être réutilisés dans d'autres contextes.

Mots-clés : correction calculatoire, protocoles cryptographiques, vérification, processus communicants

1 Introduction

Two families of models have been designed for the rigorous analysis of security protocols: the so-called Dolev-Yao (symbolic, formal) models on the one hand and the cryptographic (computational, concrete) models on the other hand. In symbolic models messages are formal terms and the adversary can only perform a fixed set of operations on them. The main advantage of the symbolic approach is its relative simplicity which makes it amenable to automated analysis tools (see, e.g., [17, 8]). In cryptographic models, messages are bit strings and the adversary is an arbitrary probabilistic polynomial-time (ppt) Turing machine. While the proofs in such models yield strong security guarantees, they are often quite involved and seldom suitable for automation.

Starting with the seminal work of Abadi and Rogaway [5], a lot of efforts has been directed to bridging the gap between the two approaches. The goal is to obtain the best of both worlds: simple, automated security proofs that entail strong security guarantees. The numerous relevant works can be divided into two categories. In the first one ([1, 15, 35] and many others), the authors generalize Abadi and Rogaway result, typically considering a larger set of security primitives. However, they still only consider a *passive adversary*. This rules out the so-called "man-in-the-middle attacks". Analyzing real protocols requires to consider active adversaries, which is the aim of the second category of papers (e.g. [10, 21, 24, 34]). It is also the aim of the present paper. We consider however a wider class of security properties.

Trace properties vs. Equivalence properties. We call here a *trace property* a formal statement that something bad never occurs on any trace of a protocol. (Formally, this is a property definable in linear time temporal logic). Integrity and authentication are examples of trace properties. That is why they were the first for which computational guarantees were derived out of symbolic ones [12, 36].
There are however several security properties, which cannot be defined (or cannot be naturally defined) as trace properties. We give now some examples, both in the computational and in the symbolic worlds.

- Anonymity states that any *two* execution traces, in which names are swapped, cannot be distinguished by an attacker. More precisely, anonymity requires two instances of the protocol P_{AB} and P_{BA}, the names A, B being switched in the second copy. An adversary interacting with one of the two copies should not be able to tell (with non-negligible probability) with which copy he is interacting. There is no known way to reduce this problem to a property of a single protocol copy.
  Privacy related properties involved in electronic voting protocols [25] also use an equivalence and cannot be expressed in linear temporal logic.

- Similarly, in the computational worlds, anonymity of group signatures [6] is defined through the indistinguishability of two games where different identities are used in each game. A similar definition is proposed for "blindness" of signatures in [31]. More details will be provided in the core of the paper.

- The "computational secrecy" states that the protocol does not leak any piece of the secret (this is sometimes called "real or random"). Such a property is naturally expressed as an indistinguishability property: the attacker cannot distinguish between two games, one of which is the real protocol, and, in the other one, the secret has been replaced by a random string. There are several works [36, 11, 24, 28, 21, 23, 30] showing how to soundly abstract it as a trace property in the symbolic model, in a number of particular cases. It is not clear, however, that such a property can be expressed as a trace property in general. Consider e.g. the case of a hash function and assume that a protocol reveals the hash $h(s)$ of some secret s. Then s cannot be computed (by one-wayness of h), which, from the trace property point of view, would be sufficient for confidentiality. On the other hand, an attacker certainly learns something about s and the computational secrecy is not preserved.

- Strong (also called "black-box") simulatability [13, 33], states that, given a real protocol P and an ideal functionality F, there is a simulator S such that P cannot be distinguished from $S\|F$ by any environment. Again, this is not a property of any particular trace, but rather a relationship between the real traces and the ideal ones. Various notions of *universal composability* [20, 22] can be defined in a similar way.

This shows the importance and generality of indistinguishability properties compared to trace properties.

The main question is then: "is it possible to get sound abstraction results for computational indistinguishability, analogous to the results obtained so far for trace properties ?" This is the question, which we want to address in this paper, for a sample set of cryptographic primitives.

Our contribution. There is a well-known similar notion in concurrency theory: observational equivalence, introduced by Milner and Hoare in the early 80s. Two processes P and Q are observationally equivalent, denoted by $P \sim_o Q$, if for any process O (a symbolic observer) the processes $P\|O$ and $Q\|O$ are equally able to emit on a given channel. This means that O cannot observe any difference between P and Q. Observational equivalence is therefore a natural candidate for the symbolic counterpart of indistinguishability, the attacker being replaced by the observer. And indeed, we show in this paper a result of the form "two networks of machines are indistinguishable if the processes they implement are observationally equivalent". As a consequence, proving computational indistinguishability can be reduced to proving observational equivalence (for a class of protocols and assuming some standard security hypotheses on the cryptographic primitives). This is a simpler task, which can be completely formalized and sometimes automated [18, 26].

We prove our result for symmetric encryption and pairing, using a fragment of the applied pi-calculus [3] for specifying protocols and relying on standard cryptographic assumptions (IND-CPA and INT-CTXT) as well as hypotheses, which are similar to those of [10]. The main difference with this latter work is that we prove the soundness of observational equivalence, which covers more properties. The fragment of applied pi-calculus we consider allows to express an arbitrary (unbounded) number of sessions of a protocol.

To prove our result, we need first to show that any computational trace is, with overwhelming probability, an instance of a symbolic one. This lemma is similar to [24, 28], though with different hypotheses and in a different model. A naive idea would be then to consider any pair of related symbolic traces: by observational equivalence (and actually labeled bisimilarity) the two traces are statically equivalent. Then we could try to apply soundness of static equivalence on these traces (using results in the passive case, e.g. [5, 1, 15, 35]). This idea does not work, since the computational traces could be spread over the symbolic ones: if there is only one computational trace corresponding to a given symbolic trace, then the symbolic traces indistinguishability does not tell us anything relevant on the computational ones.

That is why we need a new tool; the main technical ingredient of our proof is the introduction of *tree soundness* in the case of passive attackers and the use of intermediate structures, which we called *computation trees*: on one end such trees roughly correspond to the labeled transition semantics of some process algebra, and, on the other end, they are interpreted as an encryption oracle, scheduled by the attacker. These notions are defined independently of the cryptographic setting. Tree soundness captures the fact that even a passive attacker can *adaptively* choose its requests. It seems related to "adaptive soundness of static equivalence" as defined in [32] though no precise relationship has been established yet. We can then derive a general method for proving that observational equivalence implies computational indistinguishability. We believe our techniques are general and can be reused in other settings. In particular, using our generic approach, it should not be difficult to extend our result to other security primitives like asymmetric encryption and signatures.

Related Work. In a series of papers starting with Micciancio and Warinschi [36] and continued with e.g. [24, 28], the authors show trace mapping properties: for some selected primitives (public-key encryption and signatures in the above-cited papers) they show that a computational trace

is an instance of a symbolic trace, with overwhelming probability. We have a similar result for symmetric encryption in the present paper, but this is not our main contribution: this is only a step towards the lifting of indistinguishability.

There is a huge amount of work on simulatability/universal composability, especially the work of Backes *et. al.* and Canetti [20, 13, 12, 10, 11]. When the ideal functionality is the symbolic version of the protocol, then the black-box simulatability implies the trace mapping property [9], therefore showing a safe abstraction. But, again, this does not show that (and how) indistinguishability properties can be soundly abstracted.

Simulatability itself (this time, not necessarily with an ideal functionality, which is given by the symbolic protocol), given a simulator, is an indistinguishability property. So, our result could help simplifying and maybe automating simulatability proofs, provided the simulator is given. Indeed, thanks to our result, it is now possible to prove simulatability in the *symbolic* setting.

Our work can also be seen as a generalization of soundness results for static equivalence [5, 14, 15] from a passive attacker to an active one. However, as we sketched above and as we will see on an example later, these results cannot be used directly in the active attacker case, which is the one we consider.

In [21] the authors show how to encode an equivalence property (actually computational secrecy for a given set of primitives) in the symbolic trace, using patterns. This allows to show how an indistinguishability property can be lifted to the symbolic case. The method, contrary to ours, is however dedicated to this particular property.

The work of Mitchell *et. al.* [37] also aims at faithfully abstracting the model of interactive Turing machines. Their results concern general processes and not only a small fragment, as we do here. In this respect, they are much more general than us. However, on the other hand, they abstract much less: there are still computations, coin tossing and probabilistic transitions in their model. Our aim is really to show that it makes no difference if the attacker is given only a fixed set of operations (encryption, decryption, name generation...) and if there is no probabilities nor coin tossing. In the present paper, we start from a simple model of communicating Turing machines, but we could have started from a higher-level language such as the probabilistic process algebras of [37], without changing the core of our contribution.

To our knowledge, the only previous work formally connecting observational equivalence and computational indistinguishability is [7]. In this paper, the authors give soundness and completeness results of a cryptographic implementation of processes. The major difference with our work is that they do not have explicit cryptographic constructions in the formal model. For instance encryption keys cannot be sent or leaked since they are implicit. Most standard security protocols cannot be described at this level of abstraction without possibly missing attacks. The results of [7] are useful in designing secure implementations of abstract functionalities, not for the verification of existing protocols.

Finally, the work on automation and decision of static equivalence [2] and observational equivalence [27, 18, 26] is related to our work, as it shows that there exist systematic ways of deriving such equivalences in the symbolic world.

Organization of the paper. We first give the definitions of our computational model in section 2. Next we recall some of the general definitions of applied π-calculus in section 3. Note that, in the following, we only consider a fragment of the calculus for the protocol description (as usual), and we will only consider a particular equational theory corresponding to symmetric encryption. The relationship between the two models, as well as the protocol description language is given in section 4. In section 5 we give our main result and outline the proof. The notions of computation trees, tree oracles, tree soundness are introduced in section 6. Intermediate lemmas and full proofs are provided in section 7.

2 Communicating Turing machines

Randomized Turing machines are Turing machines with an additional *random tape*. We assume w.l.o.g. that these machine first draw an infinite random input string on the random tape, and then compute deterministically. *Communicating Turing machines* are randomized machines equipped with input/output tapes and two special instructions: send and receive. They are restricted to work in polynomial time *with respect to their original input* (see [13, 33] for a discussion). As in [13], we assume that, for each machine M, there is a polynomial P_M such that the number of computation steps of M is bounded by $P_M(\eta)$ where η is the length of the *initial input of* M. So, in our framework, a communicating Turing machine is actually a pair of a machine and a polynomial and, in an initialization phase, the machine will compute $P_M(\eta)$. In further computations M will subtract 1 from the result after each computation step (not including the very substraction operation, of course). When the machine reaches the computation bounds, it simply performs a fixed number of moves, typically writing an error message on the output tape, and stops. In order to keep light notations, we will abstract out this, essential but straightforward, issue in the following and forget about the polynomial P_M. Randomized machines equipped with such a time complexity control device will be called PPT.

Definition 1 *A* communicating Turing machine *(CTM) is a PPT, equipped with two additional tapes. One is the* sending tape *and is write only and the other is the* receiving tape *and is read only. The machine has two special instructions:* receive *and* send.

We assume that there are two final states: an accepting one and a failure one (written $\perp$*) and that final states can only be reached after a* send *action and that there is no deadlock, except in final states.*

The final assumptions are not restrictions as the machines can always be completed in order to fulfill these additional requirements.

Definition 2 *The* adversary *is a PPT with three additional working tapes: a reading tape (*T_R*), a writing tape (*T_W*) and a scheduling tape (*T_S*) The adversary has three additional instructions:* new, send *and* receive. *A* network $\mathcal{F}\|A$ *consists of an adversary* A *and a family of CTMs* $\mathcal{F} = (\mathcal{M}_n)_{n\in\mathbb{N}}$. *Each machine* $\mathcal{M}_n$ *has a* type*, which is a positive integer. We also call* $\mathcal{F}$ *the* environment *of* A.

This model is a simplification of interactive Turing machines of [20], keeping only the essential features. The type of a machine represents the program implemented by the machine. For instance, protocols are usually described as parallel a composition of finitely many *roles*, each of which is a reactive program executed on a single machine. In this case, the type of the machine is the role, which is implemented by the machine.

In the *initial configuration*, each machine of the network has the security parameter in unary on its input tape and possibly other data on their input tapes such as secret keys. For simplicity we do not model here the key distribution. Moves between configurations are defined according to the attacker's view: in each configuration, the attacker decides to perform an internal move, to ask for the initialization of a new machine or to schedule a communication. In the latter case, the identity of the recipient is written on the scheduling tape and either a send or a receive action is performed. In case of a send, the contents of the sending tape is copied to the receiving tape of the scheduled recipient, who performs (in one single step) all its computations, until (s)he is waiting for another communication. In case of a receive action, the whole content of the sending tape of the scheduled machine is copied on the receiving tape of the attacker. More formally:

Definition 3 *A* configuration *of the network consists in*

1. *a local configuration* γ_n *for each machine* $\mathcal{M}_n$
2. *a local configuration* γ *of the adversary*
3. *a list of active machines* l_{am}*: this is a list of triples* (n, j, p) *where* n *is a machine number,* j *a machine type and* $p \in \mathbb{N}$ *a pid.*

Given a configuration $(\{\gamma_n\}_{n\in\mathbb{N}}, \gamma, l_{am})$ *the network* moves to *the configuration* $(\{\gamma'_n\}_{n\in\mathbb{N}}, \gamma', l'_{am})$, *which is defined as follows:*

- *If the move of the adversary A does not involve any of its special actions* new, send, receive, *then* $\gamma'_n = \gamma_n$ *for all* n, $l_{am} = l'_{am}$ *and* γ' *is obtained by the move of A*
- *If A performs a* receive *action, p is the content of the scheduling tape of A*
  - *if* $(i, j, p) \in l_{am}$, *and* M_i *is ready to perform a* send *action and w is the content of the sending tape of* M_i, *then* $\gamma'_n = \gamma_n$ *for all* $n \neq i$, T_R *is set to w and the control is given to* M_i, *which moves until it reaches a final state or is ready to perform a* receive *action or a* send *action. This yields a configuration* γ'_i *of* M_i, *A moves to* γ' *and* $l'_{am} = l_{am}$.
  - *Otherwise,* $\gamma'_n = \gamma_n$ *for all* n, $l'_{am} = l_{am}$, T_R *is set to* $\perp$ *and A moves to* γ'.
- *If A performs a* send *action, p is the content of the scheduling tape of A and w is the content of* T_W
  - *if* $(i, j, p) \in l_{am}$, *and* M_i *is ready to perform a* receive *action, then* $\gamma'_n = \gamma_n$ *for all* $n \neq i$, *w is copied on the receiving tape of* M_i *and the control is passed to* M_i, *which moves until it reaches a final state or is ready to perform a* receive *action or a* send *action. This yields a configuration* γ'_i *of* M_i, *A moves to* γ' *and* $l'_{am} = l_{am}$.
  - *Otherwise,* $\gamma'_n = \gamma_n$ *for all* n, $l'_{am} = l_{am}$, T_R *is set to* $\perp$ *and A moves to* γ'.
- *if A performs a* new *action, and j is on the scheduling tape of A,*
  - *if there is at least one machine* M_n *whose type is j and such that there is no triple* $(n, j, x) \in l_{am}$, *we let n be the minimal such index. Then, in the new configuration,* $\gamma'_n = \gamma_n$ *for all* n, T_R *is set to a pid p drawn (on the random tape), A moves to* γ' *and* $l'_{am} = l_{am} \cup \{(n, j, p)\}$
  - *otherwise,* T_R *is set to* $\perp$, l_{am} *and all* γ_i *are unchanged and A moves to* γ'.

The number of CTMs in the network is unbounded. Note that this setting does not allow dynamically corrupted parties as in most results relating symbolic and computational models; there is simply a type for corrupted parties playing a given role. Corrupted machines, in addition to the expected program implementing the role, also send all their private data on the network.

There is no limitation in the messages size, which are inputs of the machines. However, because of polynomial time restrictions, unexpectedly long messages cannot be read entirely anyway.

We say that a function $f : \mathbb{N} \to \mathbb{N}$ is *negligible* if, for every polynomial P, $\exists N \in \mathbb{N}, \forall \eta > N, f(\eta) < \frac{1}{P(\eta)}$. We write $\mathbf{Pr}\{x : P(x)\}$ the probability of event $P(x)$ when the sample x is drawn according to an appropriate distribution (the key distribution or the uniform distribution; this is kept implicit).

Two families of machines are indistinguishable if any adversary cannot tell with which family he is connected with non negligible probability:

Definition 4 *Two environments* $\mathcal{F}$ *and* $\mathcal{F}'$ *are* indistinguishable, *which we write* $\mathcal{F} \approx \mathcal{F}'$, *if, for every attacker A,*

$$|\boldsymbol{Pr}\{\overline{r}, r : (\mathcal{F}(\overline{r}) \parallel A(r))(0^\eta) = 1\} - \boldsymbol{Pr}\{\overline{r}, r : (\mathcal{F}'(\overline{r}) \parallel A(r))(0^\eta) = 1\}|$$

is negligible. $\overline{r}$ *is the sequence of random inputs of the machines in* $\mathcal{F}$ *(resp.* $\mathcal{F}'$*), including keys. r is the random input of the attacker.*

Example 2.1 *As described in introduction, the* computational secrecy *of s can be expressed as follows. In* $\mathcal{F}_0$, *the machines using s are set with* s_0 *while in* $\mathcal{F}_1$, *they are set with* s_1. *The values* s_0 *and* s_1 *could also be chosen by the attacker. Then the data s is computationally secret if* $\mathcal{F}_0 \approx \mathcal{F}_1$. *Note that the environments* $\mathcal{F}_b$ *may contain corrupted machines, not holding* s_i, *that start by leaking their private information to the adversary.*

Example 2.2 *Anonymity of group signatures [6] is defined through the following game: the adversary chooses a message m and two identities i_0 and i_1. Then in $\mathcal{F}_0$, the machines sign m with identity i_0 while in $\mathcal{F}_1$, the machines sign m with identity i_1. Again the property is defined by $\mathcal{F}_0 \approx \mathcal{F}_1$.*

Example 2.3 *More generally, security properties can be defined by specifying the ideal behavior P_{ideal} of a protocol P and requiring that the two protocols are indistinguishable. For example, in [4], authenticity is defined through the specification of a process where the party B magically received the message sent by the party A. This process should be indistinguishable from the initial one.*

3 The applied pi-calculus

We use the applied π-calculus of [3] as a symbolic model. We recall the definitions in this section. Note that we will only consider a small fragment of the applied-π-calculus for the protocol descriptions and only a particular equational theory for our main result.

3.1 Syntax

A *signature* is a finite set of function symbols with an arity. It represents the security primitives (e.g. encryption, pairing, decryption). Given a signature Σ, an infinite set $\mathcal{N}$ of *names* and an infinite set $\mathcal{X}$ of *variables*, $\mathcal{T}(\mathcal{N}, \mathcal{X})$ is the set of *terms*, defined as follows:

s, t, u ::=	terms
x, y, z	variable
a, b, c, k, n, r	name
$f(s_1, \ldots, s_k)$	function application

where $f \in \Sigma$ and k is the arity of f.

We assume that Σ always contains a binary pairing function $< u, v >$, the corresponding projections functions π_1, π_2 and constants new_j for $j \in \mathbb{N}$. We assume infinitely many names of any length. Terms represent messages and names stand for (randomly) generated data. $\alpha, \beta, \ldots$ are meta-variables that range over names and variables. $\overline{u}$ stands for a sequence $u_1, \ldots, u_n$. $\sigma = \{x_1 \mapsto s_1, \ldots, x_k \mapsto s_k\}$ is the substitution that replaces the variable x_i with the term s_i. The domain of σ, denoted by $\mathsf{dom}(\sigma)$ is the set $\{x_1, \ldots, x_k\}$.

Example 3.1 *Σ_0 is the signature consisting of the binary pairing $< \cdot, \cdot >$, the two associated projections π_1, π_2, the binary decryption* dec *and the ternary symbol $\{\cdot\}^{\cdot}_{\cdot}$ for symmetric encryption: $\{x\}^r_k$ stands for the encryption of x with the key k and the random r. Σ_0 also contains constants with in particular a constant 0^l of length l for every l.*

Terms are also (as usual) identified with ranked trees, i.e. mappings from a prefix closed set of positions $\mathrm{Pos}(t)$ to labels. $t(p)$ is then the label at position p and $t|_p$ is the subtree rooted at position p.

Example 3.2 *Let $t = \{<< k_1, k_1 >, k_2 >\}^r_{k_3}$. Then $Pos(t) = \{\epsilon, 1, 2, 3, 11, 12, 111, 112\}$, $t(\epsilon) = \{\cdot\}^{\cdot}_{\cdot}$, $t(3) = r$, $t(11) =< \cdot, \cdot >$, $t(111) = k_1$. $t|_{12} = k_3$, $t|_1 =<< k_1, k_1 >, k_2 >$.*

The syntax of processes and extended processes is displayed in Figure 1. In what follows, we restrict ourselves to processes with public channels, that is, there is no restriction on name channel. We assume a set $\mathcal{P}$ of predicate symbols with an arity. Such a definition, as well as its operational semantics coincides with [3], hence will not be recalled in details, except for one minor point: we consider conditionals with arbitrary predicates. This leaves some flexibility in modeling various levels of assumptions on the cryptographic primitives. Typical examples are the ability (or not) to check whether a decryption succeeds, or the ability (or not) to check that two ciphertexts are produced using the same encryption key. Other examples are typing predicates, which we

Φ_1, Φ_2 ::=	conditions
$p(s_1, \ldots, s_n)$	predicate application
$\Phi_1 \wedge \Phi_2$	conjunction
P, Q, R ::=	processes
$c(x).P$	input
$\overline{c}(s).P$	output
$\mathbf{0}$	terminated process
$P \parallel Q$	parallel composition
$!P$	replication
$(\nu\alpha)P$	restriction
`if` Φ `then` P `else` Q	conditional
A, B, C ::=	extended processes
P	plain process
$A \| B$	parallel composition
$(\nu\alpha)A$	restriction
$\{x \mapsto s\}$	active substitution

Figure 1: Syntax of processes

may want (or not). In [3] the condition is always an equality. Encoding the predicate semantics with equalities is often possible, at least when there is no negative condition: it suffices then to express when a predicate is true. It is however not possible in general to express equationally the truth and falsity of a predicate. We believe that predicates allow to better reflect the ability of the adversary, with less coding.

In the paper, we often confuse "process" an "extended process" (and do not follow the lexicographic convention $A, B...$ vs $P.Q, ...$).

3.2 Operational semantics

We briefly recall the operational semantics of the applied pi-calculus (see [3] for details). E is a set of equations on the signature Σ, defining an equivalence relation $=_E$ on $\mathcal{T}(\mathcal{N})$, which is closed under context. $=_E$ is meant to capture several representations of the same message. This yields a quotient algebra $\mathcal{T}(\mathcal{N})/=_E$, representing the messages. and under substitutions of terms for variables.

Predicate symbols are interpreted as relations over $\mathcal{T}(\mathcal{N})/=_E$. This yields a structure $\mathcal{M}$.

Example 3.3 *The equations E_0 corresponding to Σ_0 are*

$$\mathsf{dec}(\{x\}_y^z, y) = x \qquad \pi_1(< x, y >) = x \qquad \pi_2(< x, y >) = y$$

They can be oriented, yielding a convergent rewrite system: every term s has a unique normal form $s\downarrow$. We may also consider the following predicates:

- *M is unary and holds on a (ground) term s iff $s\downarrow$ does not contain any projection nor decryption symbols and for any $\{u\}_v^r$ subterm of s, v and r must be names. This forbids compound keys for instance.*
- *EQ is binary and holds on s, t iff $M(s), M(t)$ and $s\downarrow = t\downarrow$: this is a strict interpretation of equality.*
- *$\mathsf{P}_{\mathsf{samekey}}$ is binary and holds on ciphertexts using the same encryption key: $\mathcal{M} \models \mathsf{P}_{\mathsf{samekey}}(s, t)$ iff $\exists k, u, v, r, r'.EQ(s, \{u\}_k^r) \wedge EQ(t, \{v\}_k^{r'})$.*

$$\begin{array}{rcll} A \parallel \mathbf{0} & \equiv & A & \\ A \parallel B & \equiv & B \parallel A & \\ (A \parallel B) \parallel C & \equiv & A \parallel (B \parallel C) & \\ (\nu\alpha)(\nu\beta)A & \equiv & (\nu\beta)(\nu\alpha)A & \\ (\nu\alpha)(A \parallel B) & \equiv & A \parallel (\nu\alpha)B & \text{if } \alpha \notin \mathsf{fn}(A) \cup \mathsf{fv}(A) \\ (\nu x)\{x \mapsto s\} & \equiv & \mathbf{0} & \\ (\nu\alpha)\mathbf{0} & \equiv & \mathbf{0} & \\ !P & \equiv & P \parallel !P & \\ \{x \mapsto s\} \parallel A & \equiv & \{x \mapsto s\} \parallel A\{x \mapsto s\} & \\ \{x \mapsto s\} & \equiv & \{x \mapsto t\} & \text{if } s =_E t \end{array}$$

Figure 2: Structural equivalence

- *EL is binary and holds on s, t iff $M(s), M(t)$ and s, t have the same length.*

The structural equivalence is the smallest equivalence relation on processes that is closed under context application and that satisfies the relations of Figure 2. $\mathsf{fn}(P)$ (resp. $\mathsf{fv}(P)$) is the set of names (resp. variables) that occur free in P. Bound names are renamed thus avoiding captures. $P\{x \mapsto s\}$ is the process P in which free occurrences of x are replaced by s. An *evaluation context* is a process $C = (\nu\overline{\alpha})([\cdot] \parallel P)$ where P is a process. We write $C[Q]$ for $(\nu\overline{\alpha})(Q \parallel P)$. A context (resp. a process) C is *closed* when $\mathsf{fv}(C) = \emptyset$ (there might be free names).

Possible evolutions of processes are captured by the relation $\rightarrow$, which is the smallest relation, compatible with the process algebra and such that:

$$\begin{array}{rrcll} \text{(Com)} & c(x).P \parallel \overline{c}(s).Q & \rightarrow & \{x \mapsto s\} \parallel P \parallel Q & \\ \text{(Cond1)} & \mathtt{if}\ \Phi\ \mathtt{then}\ P\ \mathtt{else}\ Q & \rightarrow & P & \text{if } \mathcal{M} \models \Phi \\ \text{(Cond2)} & \mathtt{if}\ \Phi\ \mathtt{then}\ P\ \mathtt{else}\ Q & \rightarrow & Q & \text{if } \mathcal{M} \not\models \Phi \end{array}$$

$\xrightarrow{*}$ is the smallest transitive relation on processes containing $\equiv$ and $\rightarrow$ and closed by application of contexts. We write $P \xrightarrow{c(t)} Q$ (resp. $P \xrightarrow{\overline{c}(t)} Q$) if there exists P' such that $P \xrightarrow{*} c(x).P'$ and $\{x \mapsto t\} \| P' \xrightarrow{*} Q$ (resp. $P \xrightarrow{*} \overline{c}(t).P'$ and $P' \xrightarrow{*} Q$).

Definition 5 *The* observational equivalence relation $\sim_o$ *is the largest symmetric relation $\mathcal{S}$ on closed extended processes such that $A\mathcal{S}B$ implies:*

1. *if, for some context C, term s and process A',*
   $A \xrightarrow{*} C[\overline{c}(s) \cdot A']$ then for some context C', term s' and process B', $B \xrightarrow{*} C'[\overline{c}(s') \cdot B']$.

2. *if $A \xrightarrow{*} A'$ then, for some B', $B \xrightarrow{*} B'$ and $A'\mathcal{S}B'$*

3. *$C[A]\mathcal{S}C[B]$ for all closed evaluation contexts C*

Example 3.4 (Group signature) *Group signature as defined in [6] can be modeled as observational equivalence as follows. Let $P(x, i)$ be the protocol for signing message x with identity i. Let $P_1 = c(y).P(\pi_1(y), \pi_1(\pi_2(y)))$ and $P_2 = c(y).P(\pi_1(y), \pi_2(\pi_2(y)))$. Intuitively, the adversary will send $< m, < i_1, i_2 >>$ where m is a message to be signed and i_1, i_2 are two identities. P_1 signs m with i_1 while P_2 signs m with i_2. Then P preserves anonymity if $P_1 \sim_o P_2$.*

3.3 Simple processes

We do not need the full applied pi-calculus to symbolically represent CTMs. For example, CTMs do not communicate directly: all communications are controlled by the attacker and there is

no private channel. Thus we consider the fragment of *simple processes* built on *basic processes*. Simple processes capture the usual fragment used for security protocols. A basic process represents a session of a protocol role where a party waits for a message of a certain form and when all equality tests succeed, outputs a message accordingly. Then the party waits for another message and so on. The sets of basic processes $\mathcal{B}(i, \overline{n}, \overline{x})$, where $\overline{x}$ is a variable sequence, i is a name, called *pid*, standing for the process id and $\overline{n}$ is a name sequence (including for instance fresh and long-term keys), are the least sets of processes such that $\mathbf{0} \in \mathcal{B}(i, \overline{n}, \overline{x})$ and

- If $B \in \mathcal{B}(i, \overline{n}, \overline{x})$, $s \in \mathcal{T}(\overline{n}, \overline{x})$, Φ is a conjunction of EQ and M atomic formulas such that $\mathsf{fn}(\Phi) \subseteq \overline{n}$ and $\mathsf{fv}(\Phi) \subseteq \overline{x}$, $\perp$ is a special error message, then

$$\texttt{if } \Phi \wedge M(s) \texttt{ then } \overline{\mathsf{c_{out}}}(s) \cdot B \texttt{ else } \overline{\mathsf{c_{out}}}(\perp) \cdot \mathbf{0} \quad \in \mathcal{B}(i, \overline{n}, \overline{x}).$$

  Intuitively, if all tests are satisfied and if s is a well-formed message (no destructors, no composed keys), the process sends a message depending on its inputs.

- if $B \in \mathcal{B}(i, \overline{n}, \overline{x}, x)$ and $x \notin \overline{x}$, then

$$\mathsf{c_{in}}(x).\ \texttt{if } EQ(\pi_1(x), i) \texttt{ then } B \texttt{ else } \overline{\mathsf{c_{out}}}(\perp) \cdot \mathbf{0} \quad \in \mathcal{B}(i, \overline{n}, \overline{x}).$$

  Intuitively, on input x, the basic process first checks that it is the expected recipient of the message, before processing it.

$\mathsf{c_{out}}$ and $\mathsf{c_{in}}$ are two special names, representing resp. the send tape and the receive tape.

Example 3.5 *The Wide Mouth Frog [19] is a simple protocol where a server transmits a session key from an agent A to an agent B.*

$$\begin{aligned} A \to S &: A, \{N_a, B, K_{ab}\}_{K_{as}} \\ S \to B &: \{N_b, A, K_{ab}\}_{K_{bs}} \end{aligned}$$

A session of role A played by agent a can be modeled by the basic process

$$A(a,b) \stackrel{def}{=} \texttt{if true then } \overline{\mathsf{c_{out}}}(< a, \{< n_a, < b, k_{ab} >>\}^r_{k_{as}} >) \cdot \mathbf{0} \texttt{ else } \overline{\mathsf{c_{out}}}(\perp) \cdot \mathbf{0}$$

Similarly a session of role S played for agents a, b can be modeled by

$$\begin{aligned} S(a,b) \stackrel{def}{=} \mathsf{c_{in}}(x).\ \texttt{if } & EQ(\pi_1(x), l) \texttt{ then} \\ & \texttt{if } \pi_1(\pi_2(x)) = a \ \wedge\ \pi_1(\pi_2(\mathsf{dec}_{k_{as}}(\pi_2(\pi_2(x))))) = b \texttt{ then} \\ & \quad \overline{\mathsf{c_{out}}}(\{< n_b, < a, \pi_2(\pi_2(\mathsf{dec}_{k_{as}}(\pi_2(\pi_2(x))))) >>\}^r_{k_{bs}}) \cdot \mathbf{0} \\ & \qquad \texttt{else } \overline{\mathsf{c_{out}}}(\perp) \cdot \mathbf{0} \texttt{ else } \overline{\mathsf{c_{out}}}(\perp) \cdot \mathbf{0} \end{aligned}$$

A *simple process* is obtained by composing and replicating basic processes and frames, hiding some names and all local variables:

$$(\nu \overline{n})[\ (\nu \overline{x_1}, \overline{n_1} B_1 \| \sigma_1) \| \cdots \| (\nu \overline{x_k}, \overline{n_k} B_k \| \sigma_k) \| \\ !(\nu \overline{y_1}, l_1, \overline{m_1 \mathsf{c_{out}}}(l_1) B'_1) \ \|\ \cdots\ \|\ !(\nu \overline{y_n}, l_n, \overline{m_n}.\overline{\mathsf{c_{out}}}(l_n) B'_n)\ ]$$

where $B_j \in \mathcal{B}(i_j, \overline{n} \uplus \overline{n_j}, \overline{x_j})$, $\mathsf{dom}(\sigma_j) \subseteq \overline{x_j}$, $B'_j \in \mathcal{B}(l_j, \overline{n} \uplus \overline{m_j}, \overline{y_j})$. Note that each basic process B'_j first publishes its identifier l_j, so that an attacker can communicate with it. Each process of the form $!((\nu \overline{y_i}, l_i)\overline{\mathsf{c_{out}}}(l_i).B'_i)$ is a *replicated process*.

In the definition of simple processes, we assume that for any subterm $\{t\}^v_k$ occurring in a simple process, v is a name that does not occur in any other term, and belongs to the set of restricted names $\overline{n}$. (Still, there may be several occurrences of $\{t\}^v_k$, unlike in [5]).

Example 3.6 *Continuing Example 3.5, a simple process representing unbounded number of sessions in which A plays a (with b) and s plays S (with a, b) for the Wide Mouth Frog protocol is*

$$\nu(k_{as}, k_{bs}) \quad \big(\quad !((\nu k_{ab}, n_a, r, l)\overline{\mathsf{c_{out}}}(l).A(a,b)) \| \quad !((\nu x, n_b, r, l)\overline{\mathsf{c_{out}}(l)}.S(a,b)) \quad \big)$$

For simplicity, we have only considered sessions with the same agent names a, b.

$$\frac{}{\sigma \vdash s}\ \begin{array}{c}\text{if } \exists x \in \mathsf{dom}(\sigma)\\ \text{s.t. } x\sigma = s\\ \text{or } s \in \mathcal{N} \setminus \overline{n}\end{array} \qquad \frac{\sigma \vdash s_1 \quad \ldots \quad \phi \vdash s_\ell}{\sigma \vdash f(s_1, \ldots, s_\ell)}\ f \in \Sigma$$

$$\frac{\sigma \vdash s}{\sigma \vdash s'}\ \mathcal{M} \models s = s'$$

Figure 3: Deduction rules

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\sigma \vdash < \{n_2\}^{r_2}_{n_1}, \{n_3\}^{r_3}_{n_2} >}}{\sigma \vdash \pi_2(< \{n_2\}^{r_2}_{n_1}, \{n_3\}^{r_3}_{n_2} >)}}{\sigma \vdash \{n_3\}^{r_3}_{n_2}} \qquad \dfrac{\dfrac{\dfrac{}{\sigma \vdash < \{n_2\}^{r_2}_{n_1}, \{n_3\}^{r_3}_{n_2} >}}{\sigma \vdash \pi_1(< \{n_2\}^{r_2}_{n_1}, \{n_3\}^{r_3}_{n_2} >)} \quad \dfrac{}{\sigma \vdash \{n_1\}^{r_1}_{k_1}} \quad \dfrac{}{\sigma \vdash k_1}}{\sigma \vdash \mathsf{dec}(\{n_1\}^{r_1}_{k_1}, k_1)}}{\sigma \vdash \{n_2\}^{r_2}_{n_1}} \qquad \dfrac{}{\sigma \vdash n_1}}{\sigma \vdash \mathsf{dec}(\{n_2\}^{r_2}_{n_1}, n_1)}}{\sigma \vdash n_2}}{\sigma \vdash \mathsf{dec}(\{n_3\}^{r_3}_{n_2}, n_2)}}{\sigma \vdash n_3}$$

Figure 4: An example of formal proof

3.4 Deduction and static equivalence

At a particular point in time, while engaging in one or more sessions of one or more protocols, an attacker may know a sequence of messages $s_1, \ldots, s_\ell$. This means that he knows each message but he also knows in which order he obtained the messages. So it is not enough for us to say that the attacker knows the set of terms $\{s_1, \ldots, s_\ell\}$ since the information about the order is lost. Furthermore, we should distinguish those names that the attacker knows from those that were freshly generated by others and which are *a priori* secret from the attacker; both kinds of names may appear in the terms.

As in the applied pi calculus [3], message sequences are recorded in *frames* $\phi = \nu\overline{n}.\sigma$, where $\overline{n}$ is a finite set of names, and σ is a ground substitution. The variables enable references to each sequence element. and we always assume that the terms s_i are ground. $\overline{n}$ stands for fresh names that are *a priori* unknown to the attacker.

Deduction. Given a frame $\phi = \nu\overline{n}.\sigma$ that represents the information available to an attacker, a ground term s is *deducible*, which we write $\nu\overline{n}.(\sigma \vdash s)$ if $\sigma \vdash s$ can be inferred using the rules displayed in Figure 3.

Example 3.7 *Consider the signature and equational theory of example 3.3.*

Let $\phi = \nu n_1, n_2, n_3, r_1, r_2, r_3.\sigma$ *with* $\sigma = \{x_1 \mapsto \{n_1\}^{r_1}_{k_1}, x_2 \mapsto < \{n_2\}^{r_2}_{n_1}, \{n_3\}^{r_3}_{n_2} >\}$. *Then* $\nu n_1, n_2, n_3, r_1, r_2, r_3.(\sigma \vdash n_3)$. *The full proof is displayed in figure 4.*

Static equivalence. Deduction is not sufficient for expressing the attacker's knowledge. We have also to consider its distinguishing capabilities. Using the predicate symbols, we get the following slight extension of static equivalence:

Definition 6 (static equivalence) *Let* ϕ *be a frame,* p *be a predicate and* $s_1, \ldots, s_k$ *be terms. We say that* $\phi \models p(s_1, \ldots, s_k)$ *if there exists* $\overline{n}$ *such that* $\phi = \nu\overline{n}.\sigma$, $\mathsf{fn}(s_i) \cap \overline{n} = \emptyset$ *for any* $1 \leq i \leq k$ *and* $\mathcal{M} \models p(s_1, \ldots, s_k)\sigma$. *We say that two frames* $\phi_1 = \nu\overline{n}.\sigma_1$ *and* $\phi_2 = \nu\overline{n'}.\sigma_2$ *are* statically equivalent, *and write* $\phi_1 \sim \phi_2$ *when* $\mathsf{dom}(\phi_1) = \mathsf{dom}(\phi_2)$, *and*

$$\forall s_1, \ldots, s_k \in T(\mathcal{N}, \mathcal{X}), \forall p \in \mathcal{P}. \qquad \phi_1 \models p(s_1, \ldots, s_k) \Leftrightarrow \phi_2 \models p(s_1, \ldots, s_k).$$

If the only predicate is the equality predicate modulo the equational theory, we retrieve exactly the definition of static equivalence [3].

Example 3.8 *Consider (again for the theory of Example 3.3).*

The two frames $\phi_1 = \nu n_1, r_1, r_2.\{x \mapsto \{\{k\}_{n_1}^{r_1}\}_{n_1}^{r_2}\}$ *and* $\phi_2 = \nu n_2, r_3.\{x \mapsto \{s\}_{n_2}^{r_3}\}$*. If* s *has the same length as* $\{k\}_{n_1}^{r_1}$*, then the two frames are statically equivalent, as there is no way to access the term below the encryption with* n_1 *(resp.* n_2*).*

It also sometimes convenient to forget the variable names in a frame, when they are irrelevant: if $\mathcal{L}_1 = (s_1^1, \ldots, s_m^1), \mathcal{L}_2 = (s_1^2, \ldots, s_m^2)$ are two sequences of terms, $\nu\overline{n_1}.\mathcal{L}_1 \sim \nu\overline{n_2}.\mathcal{L}_2$ if, by definition,

$$\nu\overline{n_1}\,\{x_1 \mapsto s_1^1; \ldots; x_n \mapsto s_m^1\} \sim \nu\overline{n_2}\,\{x_1 \mapsto s_1^2; \ldots; x_n \mapsto s_m^2\}$$

We also sometimes confuse $\nu\overline{n}.(s_1, \ldots, s_m)$ with the frame $\nu\overline{n}.\{x_1 \mapsto s_1; \ldots, x_m \mapsto s_m\}$, for a (fixed) given sequence of variables x_i, $i \in \mathbb{N}$.

4 Computational interpretation

4.1 Computational interpretation of terms and conditions

Given a security parameter η and a mapping τ from names to actual bitstrings of the appropriate length, which depends on η, the computational interpretation $[\![s]\!]_\eta^\tau$ of a term s is defined as a $\mathcal{F}$-homomorphism: for each function symbol f there is a polynomially computable function $[\![f]\!]$ and $[\![f(t_1, \ldots, t_n)]\!]_\eta^\tau \stackrel{\text{def}}{=} [\![f]\!]([\![t_1]\!]_\eta^\tau, \ldots, [\![t_n]\!]_\eta^\tau)$.

In addition, for names, $[\![n]\!]_\eta^\tau \stackrel{\text{def}}{=} \tau(n)$. Such an interpretation must be compatible with the equational theory: $\forall s, t, \eta, \tau.\ s =_E t \Rightarrow [\![s]\!]_\eta^\tau = [\![t]\!]_\eta^\tau$.

Similarly, each predicate symbol p gets a computational interpretation $[\![p]\!]$ as a PPT that outputs a Boolean value. This is extended to conditions, using the standard interpretation of logical connectives.

Given the structure $\mathcal{M}$, which defines the symbolic interpretation of the predicates symbols,

Definition 7 $[\![p]\!]$ *is an* implementation *of* $p \subseteq (T(\mathcal{N}))^n$*, if, for every* $t_1, \ldots, t_n \in T(\mathcal{N})$*, which are in normal form,*

$$\boldsymbol{Pr}\{(x_1, \ldots, x_n) \stackrel{R}{\leftarrow} [\![t_1, \ldots, t_n]\!]_\eta : [\![p]\!](x_1, \ldots, x_n) = 1 - b\}$$

is negligible, where $b = 1$ *if* $\mathcal{M} \models p(t_1, \ldots, t_n)$ *and 0 otherwise.*

We assume that, for every predicate symbol p, $[\![p]\!]$ is an implementation of p.

Example 4.1 *Consider the predicate symbols of Example 3.3. Assume that the decryption and projection functions return an error message* $\perp$ *when they fail. Then here are possible interpretations of some predicates:*

- $[\![M]\!]$ *is the set of bitstrings, which are distinct from* $\perp$*.* $[\![M]\!]$ *implements* M *if the encryption scheme is* confusion-free *(a consequence of INT-CTXT [35]).*
- $[\![EQ]\!]$ *is the set of pairs of identical bitstrings, which are distinct from* $\perp$*. It is an implementation of* EQ *as soon as* $[\![M]\!]$ *implements* M*.*

The interpretation of terms with variables also depends on the variable bindings: if σ is a substitution (and assuming that there is no reminding free variables), $[\![t]\!]_\eta^{\tau,\sigma}$ is defined as $[\![t\sigma]\!]_\eta^\tau$.

Now, conditions are interpreted as expected:

$$[\![\Phi_1 \wedge \Phi_2]\!]_\eta^{\tau,\sigma} \stackrel{\text{def}}{=} [\![\Phi_1]\!]_\eta^{\tau,\sigma} \wedge [\![\Phi_2]\!]_\eta^{\tau,\sigma}$$

and

$$[\![p(s_1, \ldots, s_n)]\!]_\eta^{\tau,\sigma} \stackrel{\text{def}}{=} [\![p]\!]([\![s_1]\!]_\eta^{\tau,\sigma}, \ldots, [\![s_n]\!]_\eta^{\tau,\sigma}).$$

Finally, if $\mathcal{L}$ is a list of ground terms whose free names are in the domain of τ, $[\![\nu\overline{n}.\mathcal{L}]\!]^\tau \stackrel{\text{def}}{=} [\![\mathcal{L}]\!]^\tau$.

4.2 Computational interpretation of basic processes

Given a basic process $B \in \mathcal{B}(i, \overline{n}, \overline{x})$, a substitution σ whose domain is contained in $\overline{x}$ and codomain does not contain free variable, a security parameter η and a binding τ for all free names occurring in B, we define the (deterministic) machine $[\![B\|\sigma]\!]^\tau_\eta$ as follows. Irrelevant details are omitted and (relevant information on the) configurations of the machine are written as triples $[q, \tau_0, \theta]$ where q is the state of the machine and τ_0, θ represent the (relevant part of) the working tape.

- The initial configuration of the machine is $[B, \tau_0, [\![\sigma]\!]^\tau]$: the machine is in state B and its working tape contains τ_0, the restriction of τ to names and random symbols occurring in B, and the binding for variables in $\overline{x}$; $[\![\{x_1 \mapsto t_1, \dots x_n \mapsto t_n\}]\!]^\tau$ is, by definition, $\{x_1 \mapsto [\![t_1]\!]^\tau, \dots x_n \mapsto [\![t_n]\!]^\tau\}$. Note that τ_0 includes the random input tape of the machine.
- $[\mathbf{0}, \tau_0, \theta]$ is a final configuration: $\mathbf{0}$ is a final state
- If the machine is in a configuration

  $$[\mathsf{c_{in}}(y).\ \mathtt{if}\ EQ(\pi_1(y), i)\ \mathtt{then}\ B_1\|\{x \mapsto \pi_2(y)\}\ \mathtt{else}\ B_2, \tau_0, \sigma]$$

  then it may (only) perform a receive action (which will be triggered only when the attacker schedules it, see Definition 3) then copies the content m of the reading tape on its working tape, keeping thereof the binding for y in its memory, yielding a configuration

  $$[\mathtt{if}\ EQ(\pi_1(y), i)\ \mathtt{then}\ B_1\|\{x \mapsto \pi_2(y)\}\ \mathtt{else}\ B_2, \tau_0, \sigma \uplus \{y \mapsto m\}]$$

  Next, the machine checks that m is a pair $([\![i]\!]^{\tau_0}, m_1)$. If so, it moves to the configuration

  $$[B_1, \tau_0, \sigma \uplus \{y \mapsto m, x \mapsto m_1\}]$$

  Otherwise it moves to $[B_2, \tau_0, \sigma]$.

  If x or y was bound in σ, the new bindings replace the former ones.
- If the machine is in configuration $[\mathtt{if}\ \ \Phi\ \ \mathtt{then}\ \ \overline{\mathsf{c_{out}}}(< i, s >).B_1\ \ \mathtt{else}\ \ B_2,\ \tau_0, \sigma]$ it first computes $[\![\Phi]\!]^{\tau_0,\sigma}_\eta$. If some variables are unbound or if the evaluation result is not 1, then it moves to $[B_2, \tau_0, \sigma]$, otherwise it computes $[\![s]\!]^{\tau,\sigma}_\eta$, writes it on the sending tape and performs a send action (which will be triggered when the attacker schedules it). This yields the configuration $[B_1, \tau_0, \sigma]$.

4.3 Computational interpretation of simple processes

Consider now a simple process

$$P \equiv (\nu\overline{n})[(\nu\overline{x_1}, \overline{n_1}.B_1\|\sigma_1)\| \cdots \nu\overline{x_k}, \overline{n_k}.B_k\|\sigma_k)\| \\ !(\nu\overline{y_1}, l_1, \overline{m_1}.\overline{\mathsf{c_{out}}}(l_1).B'_1)\| \cdots !(\nu\overline{y_n}, l_n, \overline{m_n}.\overline{\mathsf{c_{out}}}(l_n).B'_n)]$$

such that $B_i \in \mathcal{B}(j_i, \overline{n} \cup \overline{n_i}, \overline{x_i})$ and $\mathsf{dom}(\sigma_i) \subseteq \overline{x_i}$, $B'_i \in \mathcal{B}(l_i, \overline{n} \cup \overline{m_i}, \overline{y_j})$.

Then, given a security parameter η, $[\![P]\!]_\eta$ is (the distribution of) an infinite family of CTMs, obtained, for each sample τ of $\overline{n}, \overline{n_1}, \dots, \overline{n_k}, \overline{m_{1,\lambda}}, \dots, \overline{m_{n,\lambda}}, l_{1,\lambda}, \dots, l_{n,\lambda}$, $\lambda \in \mathbb{N}$, (depending on the security parameter η) by:

- interpreting each $B_i\|\sigma_i$ by $[\![B_i\|\sigma_i]\!]^\tau_\eta$
- interpreting each $!(\nu\overline{y_i}, l_i, \overline{m_i}.\overline{\mathsf{c_{out}}}(l_i).B'_i)$ as an infinite network of machines $M_{i,\lambda} = [\![\mathsf{if}\ x = \mathsf{new}_i\ \mathsf{then}\ \overline{\mathsf{c_{out}}}(l_{i,\lambda}).B'_{i,\lambda}\mathsf{else}\ \mathbf{0}]\!]^\tau_\eta$, where each $B'_{i,\lambda}$ is obtained from B'_i by renaming l_i in $l_{i,\lambda}$, $\overline{m_i}$ in $\overline{m_{i,\lambda}}$ and $\overline{y_i}$ in $\overline{y_{i.\lambda}}$, for $\lambda \in \mathbb{N}$, where each of these machine has type i.

5 Main result

In this section, we state our main result and give an outline of the proof. Fist, we start by stating our assumptions.

5.1 Assumptions

As in [10, 29], we avoid the so-called commitment problem by requiring that honest keys are never revealed. This hypothesis is implied by our notion of key hierarchy that is defined next.

Cryptographic assumptions Let us start with the assumptions on the encryption scheme: we assume that it is IND-CPA (more precisely "type 3"-secure of [5]) and INT-CTXT, as defined in [16]. In what follows, for randomized machines, we distinguish the input tape and the random tape, by separatin the inputs with a |. For instance $\mathcal{A}(m \mid r)$ is the result of the computation of $\mathcal{A}$, on input m, with random input r.

Definition 8 *A symmetric encryption scheme is IND-CPA, if, for any non-null polynomial P with positive integer coefficients, for every PPT A with one oracle, there is a N such that, for all $\eta > N$,*

$$|\boldsymbol{Pr}\{k \xleftarrow{R} \mathcal{K}_G(\eta), r, \overline{r} \xleftarrow{R} U : A^{\mathcal{O}_k}(0^\eta \mid r) = 1\} - \boldsymbol{Pr}\{k \xleftarrow{R} \mathcal{K}_G(\eta), r, \overline{r} \xleftarrow{R} U : A^{\mathcal{O}'_k}(0^\eta \mid r) = 1\}| \leq \frac{1}{P(\eta)}$$

where $\overline{r}$ is the sequence of random inputs $r_1, \ldots, r_n$ of the oracles, $\mathcal{O}_k$ is the oracle, which, on request x, returns $\mathcal{E}(x, k \mid r_i)$ and $\mathcal{O}'_k$ is the oracle which, on request x, returns $\mathcal{E}(0^{l(x)}, k \mid r_i)$.

Definition 9 ([16]) *A symmetric encryption scheme is INT-CTXT if, for any non-null polynomial P with positive integer coefficients, for every PPT A with one oracle, there is a N such that, for all $\eta > N$,*

$$\boldsymbol{Pr}\{k \xleftarrow{R} \mathcal{K}_G(\eta), r, \overline{r} \xleftarrow{R} U : \exists x, r'. A^{\mathcal{O}_k}(0^\eta \mid r) = \{x\}_k^{r'} \wedge x \notin S\} \leq \frac{1}{P(\eta)}$$

where $S = \{x_1, \ldots, x_n\}$ is the sequence of requests to the oracle, $\mathcal{O}_k(x_i) = \{x_i\}_k^{r_i}$ and $\overline{r}$ is the sequence $(r_1, \ldots, r_n)$.

Adversary's key generation. We assume the adversary only uses correctly generated keys. The parties are supposed to check that the keys they are using have been properly generated. The assumption could be achieved by assuming that keys are provided by a trusted server that properly generates keys together with a certificate. Then when a party receives a key, it would check that it comes with a valid certificate, guaranteeing that the key has been issued by the server. Of course, the adversary could obtain from the server as many valid keys as he wants.

However, this assumption is rather strong compared to usual implementation of symmetric keys. It is however necessary to deal with natural examples of protocols like the following one. A sends out a message of the form $\{c\}_{K_{ab}}$ where c is a constant. Note that c could also be a ciphertext. This can be formally represented by the process

$$A = (\nu r)\overline{\mathsf{c_{out}}}(< c, \{c\}^r_{k_{ab}} >).\mathbf{0}$$

B expects a key y and a message of the form $\{\{b\}_y\}_{K_{ab}}$ where b is the identity of B, in which case, it sends out a secret (or goes in a bad state). This can be formally modeled by the process

$$B = \mathsf{c_{in}}(z).\ \texttt{if}\ EQ(b, \mathsf{dec}(\mathsf{dec}(\pi_2(z), k_{ab}), \pi_1(z))\ \texttt{then}\ \overline{\mathsf{c_{out}}}(s)\ \texttt{else}\ \mathbf{0}$$

Then symbolically, the process $(\nu k_{ab})(\nu s)A\|B$ would never emit s. However, a computational adversary can forge a key k such that any bitstring can be successfully decrypted to b using k. In particular, at the computational level, we have $\mathsf{dec}(c,k) = b$. Thus by sending $< k, \{c\}^r_{k_{ab}} >$ to B, the adversary would obtain the secret s. This is due to the fact that security of encryption schemes only provide guarantees on *properly generated* keys.

We do not know how this counter-example is handled in[10] and [29].

Key hierarchy We say that a term u is a *plaintext subterm* of t if it is a subterm of t such that at least one occurrence of u in t is not at a key or a randomness position. More formally, the set $\mathsf{pst}(t)$ of plaintext subterms of t is defined as follows: $\mathsf{pst}(a) = \{a\}$ if a is a name, a constant or a variable, $\mathsf{pst}(< t_1, t_2 >) = \mathsf{pst}(t_1) \cup \mathsf{pst}(t_2)$ and $\mathsf{pst}(\{u\}^r_v) = \mathsf{pst}(u)$. For example, $\mathsf{pst}(< k_1, \{\{a\}^{r_2}_{k_2}\}^{r_1}_{k_1} >) = \{a, k_1\}$.

We say that k *encrypts* k' in a set of terms S if S contains a subterm of the form $\{u\}^r_k$ such that k' is a plaintext subterm of u. There is a *key cycle* in S if there exist $k_1, \ldots, k_n$ such that k_{i+1} encrypts k_i for $1 \leq i \leq n-1$ and k_1 encrypts k_n.

We require a hierarchy on keys to prevent key cycles. To define an order on keys, we need some convention on the renaming of names in processes. This renaming occurs in particular when a process is replicated: $!((\nu n)P) \to (\nu n)P\|!((\nu n)P)$. We will assume by convention that the name n is successively renamed in $n_1, n_2, n_3, \ldots$

More formally, let

$$P = (\nu \overline{n}[B_1\|\sigma_1\| \cdots \|B_k\|\sigma_k\|S_1\| \cdots \|S_n])$$

be a simple process where the B_i are basic processes and $S_i = !((\nu \overline{k^{S_i}})S'_i)$ are replicated processes. We assume by renaming that all names occurring in P (bounded or free) are distinct. Let $K = \bigcup_{i=1}^n \overline{k^{S_i}}$ be the set of bounded names occuring under a replication and N be the set of all remaining names of P that are not in K. The set of *extended names* of P is $N \cup \{k^i \mid k \in K, i \in \mathbb{N}\}$. By convention, we assume that each time a replicated process S_i is duplicated: $S_i \to (\nu \overline{k^{S_i}})S'_i\|S_i$, the process $(\nu \overline{k^{S_i}})S'_i$ is renamed in $(\nu \overline{k^{S_i,p}})S'_i[k_j^{S_i} \mapsto k_j^{S_i,p}]$ where p is the least integer that has not been used yet for renaming the names of $\overline{k^{S_i}}$.

We say that v *visibly occurs* in u if $u = v$ or $u =< u_1, u_2 >$ and v *visibly occurs* in u_1 or u_2. We say that a key is *immediately revealed* if it is (visibly) disclosed before being used. More formally,

Definition 10 *A key k is* immediately revealed *in a simple process*

$$P = (\nu \overline{n}[B_1\|\sigma_1\| \cdots \|B_k\|\sigma_k\|S_1\| \cdots \|S_n])$$

if, whenever $\overline{\mathsf{c_{out}}}(s)$ occurs in some B_i or S_i with k occurring in s, that it B_i or S_i is of the form $B = C[\overline{\mathsf{c_{out}}}(s).B']$ for some context C then k visibly occurs in s or there is a term u such that k visibly occurs in u and $\overline{\mathsf{c_{out}}}(u)$ occurs before in B, that is, $C = C'[\overline{\mathsf{c_{out}}}(u).C''[\_]]$ for some contexts C', C''.

Immediately revealed keys typically correspond to keys of corrupted parties. If $k \in K$ is immediately revealed then we also say that k_i is immediately revealed for any $i \in \mathbb{N}$.

The *extended honest names* of P are the extended names of P which are not immediately revealed.

Definition 11 *We say that a simple process P admits a* key hierarchy *if there exists an strict ordering $<$ on the extended honest names of P such that, for any extended honest names k, k', whenever k encrypts k' in the image of σ, for some closed evaluation context C, for some process Q such that $C[P] \xrightarrow{*} Q$ and such that $\nu\overline{n}.\sigma$ is a frame of Q, then $k' < k$. We say that $\nu\overline{n}.\sigma$* observes *the key hierarchy $<$.*

It is easy to see that if there exists a key hierarchy, no key cycle can be created. Moreover, if there exists a key hierarchy then no (extended) honest keys can be learned by the adversary.

Indeed, if an honest key k were learned by the adversary, he would be able to forge $\{k\}_k$ and thus to contradict the key hierarchy property.

Compared to [29], our hypothesis on keys is more general. In [29], the order is imposed by the order in which keys are generated and a key k cannot encrypt keys generated before k. If a protocol satisfies the hypothesis of [29], it also admits a key hierarchy for our definition.

Compared to [10], our hypothesis is more flexible since we do not impose a fixed order. However, the two hypotheses are incomparable. In [10], he order is imposed by the order in which keys are used for encrypting: if k_1 is used for encrypting before k_2 is used for encrypting than $k_1 > k_2$. (Actually, they require $k_1 < k_2$ but they conversely impose that whenever k encrypts k' then $k < k'$.) This definition allows that the order on the keys depends on the execution of the protocol.

Consider the protocol P_1 informally described as follows.

$$\begin{array}{lcl} B \to A & : & N \\ A \to B & : & \{N, K_1, K_2\}_{K_{ab}}, \{N, K_2, K_1\}_{K_{ab}} \\ B \to A & : & \{K_1\}_{K_2} \end{array}$$

Then P_1 satisfies the requirement of [10], even if, once A has sent $\{N, K_1, K_2\}_{K_{ab}}, \{N, K_2, K_1\}_{K_{ab}}$, the adversary can choose to obtain $\{K_1\}_{K_2}$ or $\{K_2\}_{K_1}$. (Note that he cannot obtain both.) The protocol P_1 does not admit a key hierarchy with our definition.

Consider now the protocol P_2 informally described as follows.

$$\begin{array}{lcl} A \to B & : & \{N, K_1, K_2\}_{k_{ab}}, \{N\}_{K_1} \\ B \to A & : & \{K_1\}_{K_2} \end{array}$$

Then P_2 does not satisfy the requirement of [10] while it does admit a key hierarchy by requiring $N < K_1 < K_2$.

Parsing To ease parsing operations, we assume that the pairing, key generation and encryption functions add a typing tag (which can be changed by the attacker), which includes which key is used in case of encryption. This can be easily achieved by assuming that a symmetric key k consists of two parts (k_1, k_2), k_1 being generated by some standard key generation algorithm and k_2 selected at random. Then one encrypts with k_1 and tags the ciphertext with k_2.

5.2 Main theorem

We are now ready to state our main theorem: observational equivalence implies indistinguishability.

Theorem 12 *Let P_1 and P_2 be two simple processes such that each P_i admits a key hierarchy. Assume that the encryption scheme is joint IND-CPA and INT-CTXT. Then $P_1 \sim_o P_2$ implies that $[\![P_1]\!] \approx [\![P_2]\!]$.*

For example, anonymity of group signature as defined in section 2 is soundly abstracted by the property defined in Example 3.4. Computational secrecy as defined in section 2 can be soundly abstracted by strong secrecy: a secret x is *strongly secret* in P if $P(s) \sim_o P(s')$ for any term s, s'. Note however that strong secrecy implies computational secrecy but might be too strong since in the computational games corresponding to strong secrecy, the secret values are given to the adversary *before* it starts interacting with the protocol while these values are given at the end of the interaction in the definition of computational secrecy.

5.3 Overview of the proof

The rest of the paper is devoted to the proof of Theorem 12.

A first approach Let us first show why the naive ideas do not work. Assume we have proved that any computational trace is an interpretation (for some sample input) of a symbolic trace. Assume moreover that we have a soundness result showing that, if $s_1, \ldots, s_n$ and $u_1, \ldots, u_n$ are two equivalent sequences of terms, then the distributions $[\![s_1, \ldots, s_n]\!]$ and $[\![u_1, \ldots, u_n]\!]$ are indistinguishable. Assume finally that the traces of P_1 and the traces of P_2 can be pairwise associated in statically equivalent traces (as a consequence of observational equivalence).

One could think that it is possible to conclude, pretending that $[\![P_2]\!] \approx [\![P_1]\!]$ since $[\![t_1]\!] \approx [\![t_2]\!]$ for each trace t_1 of P_1 and the corresponding trace t_2 of P_2. This is however incorrect. Indeed, an attacker can choose his requests (hence the trace) depending on the sample input. In the equivalence $[\![t_1]\!] \approx [\![t_2]\!]$, we use an average on the samples, while the choice of t_1 (and t_2), may depend on this sample: there is a circular dependency.

To be more concrete, here is a toy example. Assume that an attacker, given a random input τ, first gets $[\![s]\!]^\tau$ (in both experiments) and then, schedules his requests depending on the ith bit of $[\![s]\!]^\tau$: at the ith step, he will get t_i^j (resp. u_i^j in the second experiment), where j is the ith bit of $[\![s]\!]^\tau$. Assume that, for any sequence of bits $j_1, \ldots, j_n$,

$$[\![s, t_1^{j_1}, \ldots, t_n^{j_n}]\!] \approx [\![s, u_1^{j_1}, \ldots, u_n^{j_n}]\!]$$

but that, for the particular sample τ such that $[\![s]\!]^\tau = j_1 \cdots j_n$, the attacker outputs 1 on input $[\![s, t_1^{j_1}, \ldots, t_n^{j_n}]\!]^\tau$ and outputs 0 on input $[\![s, u_1^{j_1}, \ldots, u_n^{j_n}]\!]^\tau$. This may happen as the distributions could be indistinguishable while distinguished on one particular sample value. Note that, in the distribution equivalence, we draw again a sample, while the choice of $j_1, ..., j_n$ depended precisely of that sample. Then the attacker always outputs 1 in the first experiment since he precisely chose from τ the sequence $j_1, ...j_n$. Similarly, he always outputs 0 in the second experiment: he gets a significant advantage, distinguishing the two processes.

The example shows that we cannot simply use the soundness of static equivalence on traces. The idea is to consider trees labeled with terms, instead of sequences of terms. Then we do not commit first to a particular trace (as choosing $j_1, ..., j_n$ above). Considering such trees requires an extension of the results of Abadi and Rogaway, which are proved for sequences of terms.

Actually standard security definitions such as IND-CPA, IND-CCA, can be recasted as an interaction of the attacker with a tree of possible oracle calls. The tree structure reflects the adaptivity of the attacker. If we look carefully, in [5], the proof does not use the adaptivity of the attacker. Indeed, from a term sequence, all oracle calls are known beforehand and the simulator does not need to adapt his oracle calls to the replies.

Proof sketch We associate a tree T_P with each process P, which we call *process computation tree* and define symbolic and computational equivalences (denoted respectively $\sim$ and $\approx$) on process computation trees (see the definitions in the section 6). Such trees record the possible behaviors of the symbolic process, depending on (non-deterministic) choices of the symbolic attacker. We use process computation trees as an intermediate step and show the following implications:

$$P \sim_o Q \Rightarrow T_P \sim T_Q \Rightarrow T_P \approx T_Q \Rightarrow [\![P]\!] \approx [\![Q]\!]$$

$P \sim_o Q \Rightarrow T_P \sim T_Q$: (Lemma 15) It holds for any term algebra, relying however on the particular fragment of process algebra (simple processes). This is similar to the classical characterization of observational equivalence as labeled bisimilarity.

$T_P \sim T_Q \Rightarrow T_P \approx T_Q$: (Lemma 19) It uses the (tree) soundness in the ground case. This is a new concept, which generalizes the soundness of static equivalence from sequences to trees. It is necessary for the preservation of trace equivalences.

$T_P \approx T_Q \Rightarrow [\![P]\!] \approx [\![Q]\!]$ (Lemma 25) It uses trace lifting: we need to prove that a computational trace is, with an overwhelming probability, an instance of a symbolic trace.

The two last implications are proved here in the context of pairing and symmetric encryption only. However, we believe that the use of computation trees and the way we get rid of encryption, can be extended to other primitives.

6 Computation Trees

Previous results on proving soundness of symbolic models hold either for static equivalences [15] or for relating symbolic and computational traces [24]. However, combining soundness of static equivalence with trace lifting is not sufficient for relating observational equivalence with indistinguishability. Instead of defining the static equivalence on frames (related to only one, linear, execution trace), we move to an equivalence on trees, which records execution traces.

6.1 General Computation Trees

Let $S = T(\mathcal{N})$ be a set of labels, typically a pair $< i, u >$ of a pid and a term for a request u to the process i, or a request to start a new process.

If $\mathcal{L}_1, \mathcal{L}_2$ are two sequences of terms $\nu\overline{n_1}.\mathcal{L}_1 \subseteq \nu\overline{n_2}.\mathcal{L}_2$ if, by definition,

- $\overline{n_1} \subseteq \overline{n_2}$
- $\exists \mathcal{L}$, $\mathcal{L}_2 = \mathcal{L}_1 \cdot \mathcal{L}$ ($\mathcal{L}_2$ is obtained from $\mathcal{L}_1$ by appending a list of terms)
- $(\overline{n_2} \setminus \overline{n_1}) \cap \mathsf{fn}(\mathcal{L}_1) = \emptyset$: newly restricted names are also newly introduced.

We also write $t \in \nu\overline{n}.\mathcal{L}$ if t is a member of the list $\mathcal{L}$.

Definition 13 *A* computation tree T *is a mapping from a prefix closed subset of* S^* *(*$Pos(T)$*, the* positions *of* T*) to pairs* (P, ϕ) *where* P *is a simple process and* $\phi = \nu\overline{n}.\mathcal{L}$ *is a ground frame over* $\mathcal{T}(\mathcal{N})$*. If* $p \in Pos(T)$ *and* $T(p) = (P, \phi)$*, we write* $\phi(T, p)$ *the frame* ϕ*. Moreover* T *must satisfy the following conditions:*

- *for every positions* p, q*, if* $p > q$*, then* $\phi(T, q) \subseteq \phi(T, p)$
- *for every position* $p \cdot t$*,* $t \in \phi(T, p \cdot t)$
- *for every positions* $p \cdot t, p \cdot u$*, if* $t =_E u$*, then* $t = u$*. This ensures that there is no two branches labeled with the same (equivalent) message.*

The trees need not to be finite: they may be both infinitely branching and contain infinite paths. Finally, positions need not to be closed lexicographically.

The definition of static equivalence $\sim$ is extended to computation trees. $T|_p$ is the sub-tree of T at position p.

Definition 14 $\sim$ *is the largest equivalence relation on computation trees such that if* $T_1 \sim T_2$*, then* $\phi(T_1, \epsilon) \sim \phi(T_2, \epsilon)$ *and there is an one-to-one mapping* β *from* $T(\mathcal{N})$ *to itself such that, for any length 1 position* t *of* T_1*,* $T_1|_t \sim T_2|_{\beta(t)}$*.*

Typically, requests sent by the adversary need not to be identical, but must be equivalent. Then β is a mapping, which depends on T_1, T_2, which associates the messages in the labels of T_1 with equivalent messages labeling T_2.

6.2 Process computation trees

We organize all possible symbolic executions of a simple process P in a tree T_P. Each node of T_P is labeled by (Q, ϕ) where Q is the current state of the process and ϕ represents the sequence of messages sent over the network by P so far.

Let $\mathcal{P} \equiv \nu\overline{n}, \nu\overline{x}.\ Q_1 \| \sigma_1 \| \cdots \| Q_N \| \sigma_N \| S$ be a simple process where $S = S_1 \| \cdots \| S_k$ is the composition of a finite number of replicated processes S_i and every $Q_i \in \mathcal{B}(l_i, \overline{n_i}, \overline{x_i})$ is either $\mathbf{0}$ or a basic process $\mathsf{c_{in}}(x_i).P_i$ and σ_i is a ground substitution whose domain contains only free variables of P_i. Note that l_i is the pid of Q_i. The *process computation tree* $T_{\mathcal{P}}$ is defined as follows. The

labeling and positions are defined by induction on the position length: $T_{\mathcal{P}}(\epsilon) = (\mathcal{P}, \mathtt{id})$ where $\mathtt{id}$ denotes the empty frame, and, for any $p \in \mathsf{Pos}(T_{\mathcal{P}})$, let

$$T_{\mathcal{P}}(p) = (\nu\overline{n}\nu\overline{x}.\ Q_1^p \| \sigma_1^p \| \cdots \| Q_N^p \| \sigma_N^p \| S\ ,\ \ \nu\overline{n'}.\mathcal{L})$$

where each Q_j^p is either $\mathbf{0}$ or $\mathsf{c}_{\mathsf{in}}(x_j^p).P_j^p$. Then $q = p \cdot \alpha \in \mathsf{Pos}(T_{\mathcal{P}})$ if $\alpha =< l_i, u >$, $Q_i^p \not\equiv \mathbf{0}$, $\nu\overline{n'}\sigma \vdash \alpha$ and

$$Q_i^p \xrightarrow{\mathsf{c}_{\mathsf{in}}(\alpha)} \xrightarrow{\overline{\mathsf{c}_{\mathsf{out}}}(\alpha_1)\cdots\overline{\mathsf{c}_{\mathsf{out}}}(\alpha_m)} Q_i^q \| \sigma_i^p \| \{x_i^p \mapsto \alpha\}$$

in which case

$$T_{\mathcal{P}}(p \cdot \alpha) = (\nu\overline{n}\nu\overline{x}.Q_1^q \| \sigma_1^q \| \cdots \| Q_N^q \| \sigma_N^q, \nu\overline{n'}.\ \mathcal{L} \cdot (\alpha, \alpha_1, \ldots, \alpha_m))$$

where, for every $j \neq i$, $Q_j^q = Q_j^p$, $\sigma_j^q = \sigma_j^p$, Q_i^q is either $\mathbf{0}$ or $\mathsf{c}_{\mathsf{in}}(x_i^q).P(x_i^q)$ and $\sigma_i^q = \sigma_i^p \circ \{x_i^p \mapsto \alpha\}$. This corresponds to the case where an attacker sends a message to an active process Q of pid l_i. The attacker may also ask for the initialization of a process. $S = S_1 \| \cdots \| S_k$ and there is a fixed ordering on the S_j (which correspond to the roles of the protocol). Then $q = p \cdot \mathsf{new}_j \in \mathsf{Pos}(T_{\mathcal{P}})$, where new_j is a special constant ($1 \leq j \leq k$). Let $S_j =!(\nu\overline{y}, l, \overline{n_j}\ \overline{\mathsf{c}_{\mathsf{out}}}(l).B)$. Then

$$T_{\mathcal{P}}(p \cdot \mathsf{new}_j) = (\nu\overline{n}\nu\overline{x}\nu\overline{y}\nu l\nu\overline{n_j}.\ Q_1^p \| \sigma_1^p \| \cdots \| Q_N^p \| \sigma_N^p \| B \| S\ ,\ \ \nu\overline{n'}\nu\overline{n_j}.\ \mathcal{L} \cdot l)$$

assuming by renaming that the names of $\overline{y}, \overline{n_j}$ do not appear free in $Q_1^p \| \sigma_1^p \| \cdots \| Q_N^p \| \sigma_N^p$. Note that by construction, $B \in \mathcal{B}(l, \overline{n_j} \cup \overline{n}, \overline{y_j})$ thus l is the identifier of B. l is first published such that the intruder can use it to schedule B.

A process computation tree is a computation tree. Observational equivalence yields equivalence between the corresponding process computation trees:

Lemma 15 *Let P and Q be two simple processes. If $P \sim_o Q$ then $T_P \sim T_Q$.*

This does not follow directly from [3] since we slightly changed the process algebra, adding predicates. Moreover, the tree equivalence does not correspond exactly to labeled bisimilarity. Actually, we only prove an implication here (which is all we need).

Proof: Assume that $T_P \not\sim T_Q$ and let us construct a process, which distinguishes between P and Q. We construct by induction on the length of $p \in \mathrm{Pos}(T_P)$ a family of one-to-one and onto functions (from $\mathcal{T}(\mathcal{N})/=_E$ to itself) β_p such that

- if β_p is defined, then, for any $q < p$, β_q is defined. We define then (inductively) B by $B(p \cdot \alpha) = B(p) \cdot \beta_p(\alpha)$.
- if β_p is defined, then, for any $q \leq p$, $\phi(T_P, q) \sim \phi(T_Q, B(q))$
- either $\phi(T_P, p) \not\sim \phi(T_Q, B(p))$, in which case β_q is undefined for $q \geq p$,
- or β_q is defined for any $q < p$ and $\phi(T_P, p) \sim \phi(T_Q, B(p))$, in which case β_p is defined and $T_P|_p \not\sim T_Q|_{B(p)}$ implies that, for some α, $T_P|_{p\cdot\alpha} \not\sim T_Q|_{B(p\cdot\alpha)}$.

If β is defined on any strict prefix of p and $T_P|_p \sim T_Q|_{B(p)}$, we simply use the definition of $\sim$: there is a one-to-one and onto function β_p such that, for every $p \cdot \alpha \in \mathrm{Pos}(T_P)$, $T_P|_{p\cdot\alpha} \sim T_Q|_{B(p)\cdot\beta_p(\alpha)}$.

If β is defined on any strict prefix of p and $T_P|_p \not\sim T_Q|_{B(p)}$ and $\phi(T_P, p) \sim \phi(T_Q, B(p))$, then we define β_p as follows: for every α, if $p \cdot \alpha \in \mathrm{Pos}(T_P)$, then let $\phi(T_P, p) = \nu\overline{n_1}.\sigma_1^p$. By definition of T_P, $\sigma_1^p \vdash \alpha$, therefore there is a term t_α such that $\mathsf{fn}(t_\alpha) \cap \overline{n_1} = \emptyset$ and $\mathcal{M} \models t_\alpha\sigma_1^p = \alpha$. If $\phi(T_Q, B(p)) = \nu\overline{n_2}.\sigma_2^p$, then we let $\beta_p(\alpha) = t_\alpha\sigma_2^p$. Let us show that β_p thus defined is one-to-one on $\mathcal{T}(\mathcal{N})/=_E$, hence can be extended to a one-to-one mapping. If $\beta_p(\alpha) = \beta_p(\alpha')$, then there are terms t, u such that $\mathcal{M} \models t\sigma_2^p = u\sigma_2^p$ and $\mathcal{M} \models t\sigma_1 = \alpha \wedge u\sigma_1 = \alpha'$. Since $\phi(T_P, p) \sim \phi(T_Q, B(p))$, $\mathcal{M} \models t\sigma_2 = u\sigma_2$ implies $\mathcal{M} \models t\sigma_1 = u\sigma_1$, hence $\mathcal{M} \models \alpha = \alpha'$.

This completes the construction of the family β_p.

Since $T_P \not\sim T_Q$, there must be a minimal position p for which β_p is undefined, otherwise the family β_p yields an equivalence relation R satisfying the conditions of definition 14, hence $T_P \sim T_Q$ since $\sim$ is the largest equivalence relation satisfying these conditions.

Let $p = p_1 \cdot \alpha$, $p_2 = B(p_1)$.

α cannot be a constant new_j since, in that case, $\nu\overline{n_1}.\mathcal{L}_1 = \phi(T_P, p_1) \sim \phi(T_Q, p_2) = \nu\overline{n_2}.\mathcal{L}_2$ implies $\phi(T_P, p_1 \cdot \mathsf{new}_j) = \nu\overline{n'_1}.\mathcal{L}_1 \cdot l_1 \sim \nu\overline{n'_2}.\mathcal{L}_2 \cdot l_2 = \phi(T_Q, p_2 \cdot \mathsf{new}_j)$.

Then let

- $T_P(p_1) = \nu\overline{n_1}.P_1, \nu\overline{n'_1}.\ \mathcal{L}_1$
- $T_Q(p_2) = \nu\overline{n_2}.P_2, \nu\overline{n'_2}.\ \mathcal{L}_2$
- $\phi_1 = \nu\overline{n'_1}.\ \mathcal{L}_1 \sim \nu\overline{n'_2}.\ \mathcal{L}_2 = \phi_2$.
- $P_1 = (\nu\overline{n_1}\nu\overline{x})Q_1\|\theta_1^1\| \cdots \|Q_N\|\theta_N^1\|S_1$ where $Q_i \equiv c(x_i).Q'_i$
- $Q'_i \xrightarrow{\mathsf{c_{in}}(\alpha)} \xrightarrow{\overline{\mathsf{c_{out}}}(\alpha_1),\ldots\overline{\mathsf{c_{out}}}(\alpha_m)} Q''_i\|\theta_i^1\|\{x_i \mapsto \alpha\}$
- $\phi(T_P, p_1 \cdot \alpha) = \nu\overline{n'_1}.\ \mathcal{L}_1 \cdot (\alpha, \alpha_1, \ldots, \alpha_m)$.
- $P_2 = (\nu\overline{n_2}\nu\overline{x_2})R_1\|\theta_1^2\| \cdots \|R_N\|\theta_N^2\|S_2$ where $R_j = \mathsf{c_{in}}(y_j).R'_j$
- $R'_j \xrightarrow{\mathsf{c_{in}}(\beta_{p_1}(\alpha))} \xrightarrow{\overline{\mathsf{c_{out}}}(\alpha'_1),\ldots,\overline{\mathsf{c_{out}}}(\alpha'_{m'})} R''_j$ where R''_j is $\mathbf{0}$ or ready to receive a message.

We first construct a process I_{p_1} with free variables $\overline{y} = \{y_1, \ldots, y_M\}$, where M is the length of the list $\mathcal{L}_1$ (which is also the length of $\mathcal{L}_2$), such that

$$I_{p_1}\|(\nu\overline{n'_1}\ \mathcal{L}_1)\|P_1 \xrightarrow{*} \overline{c}(a).I_1$$

while, for any I_2,

$$I_{p_1}\|(\nu\overline{n'_2}.\ \mathcal{L}_2)\|P_2 \not\xrightarrow{*} \overline{c}(a).I_2$$

Since $\phi(T_P, p_1 \cdot \alpha) \not\sim \phi(T_Q, p_2 \cdot \beta_{p_1}(\alpha))$, either $m \neq m'$, in which case we may assume w.l.o.g that $m > m'$ (possibly exchanging the roles of the processes) and we construct a process

$$I_{p_1} \equiv \nu\overline{x}.\overline{\mathsf{c_{in}}}(t_\alpha) \cdot \mathsf{c_{out}}(x_1). \cdots \mathsf{c_{out}}(x_m).\overline{c}(a) \cdot \mathbf{0}$$

such that $fn(I_{p_1}) \cap \overline{n'_1} = \emptyset$ (following $fn(t_\alpha) \cap \overline{n'_1} = \emptyset$). Then

$$I_{p_1}\|(\nu\overline{n'_1}.\ \mathcal{L}_1)\|P_1 \xrightarrow{*} \overline{c}(a).I_1,$$

while the process reaches a deadlock before being able to emit on c, when combined with P_2.

In case $m = m'$, there are terms $u_1, \ldots, u_n$ s.t. $\mathsf{fn}(u_i) \cap (\overline{n'_1} \cup \overline{n'_2}) = \emptyset$ and a predicate symbol p such that $\sigma_1 \models p(u_1, \ldots, u_n)$ while $\sigma_2 \not\models p(u_1, \ldots, u_n)$ (or the converse), if we let σ_1, σ_2 be the canonical substitutions associated respectively with the lists $\mathcal{L}_1 \cdot (\alpha, \alpha_1, \ldots, \alpha_m)$ and $\mathcal{L}_2 \cdot (\beta(\alpha), \alpha'_1, \ldots, \alpha'_m)$.

We construct a process I_{p_1}, which constructs t_α as above, receives the m messages and then distinguishes the two processes using an appropriate test using the predicate p:

$$I_{p_1} \equiv \nu\overline{x}.\overline{\mathsf{c_{in}}}(t_\alpha) \cdot \mathsf{c_{out}}(x_1) \cdots \mathsf{c_{out}}(x_m) \cdot \ \mathsf{if}\ p(u_1, \ldots, u_n)\ \mathsf{then}\ \overline{c}(a) \cdot \mathbf{0}\ \mathsf{else}\ \mathbf{0}$$

Now that we have dinstinguished the processes (and frames) at position p_1, p_2, we have reconstruct, moving up along the paths p_1, p_2, a process which distinguishes the labels of the nodes, until we reach the root of the trees, thus distinguishing P and Q.

For every prefix q of p_1, let

$$T_P(q) = ((\nu\overline{n_{1,q}}\nu\overline{x_{1,q}})\ P_{1,q}, \nu\overline{n'_{1,q}}.\ \mathcal{L}_{1,q}) \quad \text{and} \quad T_Q(q) = ((\nu\overline{n_{2,q}}\nu\overline{x_{2,q}})\ P_{2,q}, \nu\overline{n'_{2,q}}.\ \mathcal{L}_{2,q}).$$

We construct, by induction on $|p_1| - |q|$, a context I_q, which distinguishes between

$$((\nu\overline{n_{1,q}}\nu\overline{x_{1,q}})\ P_{1,q}) \| (\nu\overline{n'_{1,q}}.\ \mathcal{L}_{1,q}) \quad \text{and} \quad ((\nu\overline{n_{2,q}}\nu\overline{x_{2,q}})\ P_{2,q}) \| (\nu\overline{n'_{2,q}}.\ \mathcal{L}_{2,q}).$$

I_{p_1} is the process, which we constructed above.

Let $q \cdot \alpha$ be a length $n+1$ prefix of p. Assume $\mathcal{L}_{1,q\cdot\alpha} = \mathcal{L}_{1,q} \cdot (\alpha, \alpha_1, \ldots, \alpha_m)$, $\mathcal{L}_{2,q\cdot\alpha} = \mathcal{L}_{2,q} \cdot (\beta_q(\alpha), \alpha'_1, \ldots, \alpha'_m)$ and let σ_1^q, σ_2^q be the two substitutions associated with $\mathcal{L}_{1,q}$ and $\mathcal{L}_{2,q}$ respectively. There is a term t_α such that $fn(t_\alpha) \cap (\overline{n'_{1,q}} \cup \overline{n'_{2,q}}) = \emptyset$ and $\mathcal{M} \models t_\alpha\sigma_1^q = \alpha \wedge t_\alpha\sigma_2^q = \beta_q(\alpha)$. I_q will then be the process defined by:

$$I_q \equiv \overline{\mathsf{c_{in}}}(t_\alpha) \cdot \mathsf{c_{out}}(y_1) \cdots \mathsf{c_{out}}(y_m) \cdot I_{q\cdot\alpha}$$

Then, for every $i = 1, 2$,

$$I_q \| ((\nu\overline{n_{i,q}}\nu\overline{x_{i,q}})P_{i,q}) \| (\nu\overline{n'_{i,q}}.\ \mathcal{L}_{i,q}) \ \xrightarrow{*}\ I_{q\cdot\alpha} \| ((\nu\overline{n_{i,q\cdot\alpha}}\nu\overline{x_{i,q\cdot\alpha}})\ P_{i,q\cdot\alpha}) \| (\nu\overline{n'_{i,q\cdot\alpha}}.\ \mathcal{L}_{i,q\cdot\alpha})$$

And therefore, we conclude by the induction hypothesis.

Finally, applying the result when $q = \epsilon$, we get a process I_ϵ, which distinguishes between P and Q.

□

6.3 Scheduled computation trees

When all behaviors of a concrete attacker are instances of behaviors of a symbolic attacker, the concrete attacker can be seen as a machine which schedules the behavior of the symbolic attacker. That is what we try to capture here.

We assume that, given a security parameter η and a mapping τ from names to actual bitstrings, there is a *parsing function* κ_η^τ that maps bitstring to their symbolic representation. This parsing function is assumed to be total, using possibly constants or new names of the appropriate length. In the case of symmetric encryption, we exhibit (in the proof of Lemma 19) such a parsing function, computable in polynomial time.

For any symbolic computation tree T, and random tape τ, we let $\mathcal{O}_{T,\tau}$ be an oracle, whose replies depend on the tree T and the sample τ. Initially, the tree T is a process computation tree. Then the oracle can be simply understood as simulating the network and answering to attacker's messages. This is convenient for an intuitive understanding of $\mathcal{O}_{T,\tau}$, but we will transform the tree T later on. That is why we need a general definition.

Intuitively, the tree specifies how the oracle can be adaptively queried and what are its answers. This is formalized as follows.

When queried with m_n, after being queried successively with $m_1, \ldots, m_{n-1}$,

- the oracle first computes $\kappa_\eta^\tau(m_n)$. Let $r_j = \kappa_\eta^\tau(m_j)$ for $1 \le j \le n$.
- the oracle returns 0 if $r_1 \cdots r_n$ is not a position of T
- otherwise, let $\phi_1 = \nu\overline{n_1}\sigma_1 = \phi(T, r_1 \cdots r_{n-1})$, $\phi_2 = \nu\overline{n_2}\sigma_2 = \phi(T, r_1 \cdots r_n)$ be the labels of the two successive nodes of T. For any name k that occurs in σ_2 and not in σ_1, the oracle draws a random number $\tau(k)$ using its random tape τ. If $k \notin \overline{n_2}$ then the oracle returns $\tau(k)$ (the value is public in that case).

  Next, the oracle returns $[\![x\sigma_2]\!]^\tau$ for all $x \in \mathsf{dom}(\sigma_2) \setminus \mathsf{dom}(\sigma_1)$. Intuitively, the oracle replies by sending back the (interpretation of the) terms labeling the target node of the tree that have not been already given.

In the last case, the oracle answer corresponds, in case T is a process computation tree, to the messages sent by the process answering the attacker's message.

Definition 16 *Given two symbolic computation trees T_1, T_2, the two trees are* computationally indistinguishable*, which we write $T_1 \approx T_2$ if, for any PPT A,*

$$|\boldsymbol{Pr}\{\tau, r : A^{\mathcal{O}_{T_1,\tau}}(0^\eta \mid r) = 1\} - \boldsymbol{Pr}\{\tau, r : A^{\mathcal{O}_{T_2,\tau}}(0^\eta \mid r) = 1\}|$$

is negligible.

7 Main steps of the proof

From now on, we fix the signature Σ_0, the equations E_0 and the predicate set $\mathcal{P}_0$, defined in Example 3.3 and in the following examples of Section 3.2.

The set of equations in E_0 can be turned in a convergent term rewriting system: we let $t\downarrow$ be the unique normal form of t with respect to this rewrite system.

7.1 Getting rid of encryption

To prove indistinguishability of processes, we replace ciphertexts by encryption of zeros. We follow the key hierarchy, starting from a maximal key. The function Ψ_k on trees replaces terms under encryption by constants 0^l of the same length:

$$\begin{array}{rcl} \Psi_k(n) & = & n \quad \text{if } n \text{ is a name or a constant} \\ \Psi_k(< t_1, t_2 >) & = & < \Psi_k(t_1), \Psi_k(t_2) > \\ \Psi_k(\{t\}^r_k) & = & \{0^{l(t)}\}^r_k \\ \Psi_k(\{t\}^r_{k'}) & = & \{\Psi_k(t)\}^r_{k'} \quad \text{if } k \neq k' \end{array}$$

Note that Ψ_k is an injective mapping on terms, essentialy because we assumed that two random components r of encryptions are distinct as soon as the corresponding plaintext are distinct (this correspond to two distinct calls to the encryption algorithm, which draws a random input r each time it is called).

Then Ψ_k is extended to computation trees by applying Ψ_k on the requests and frames. Intuitively, the underlying process remains the same but the adversary is given a view of the execution where any encryption by k has been replaced by an encryption of zeros by k.

If k is not deducible, then an intruder cannot tell whether the encryptions by the key k have been replaced by encryptions of zeros:

Lemma 17 *For any computation tree T and for any name k such that k is not deducible from any frame labeling a node of T, then $T \sim \Psi_k(T)$.*

Proof: β is chosen to be Ψ_k. It is a one-to-one function on labels, since Ψ_k is one-to-one. It remains to show that for any frame ψ such that $\psi \not\vdash k$, we have $\psi \sim \Psi_k(\psi)$. For any context C, $C[t_1, \ldots, t_n]\downarrow[\{u\}^r_k \mapsto \{0^{l(u)}\}^r_k] = C[t_1, \ldots, t_n][\{u\}^r_k \mapsto \{0^{l(u)}\}^r_k]\downarrow$ by induction on the length of a rewrite sequence to the normal form. Moreover, the predicates $M, EQ, EL, \mathsf{P}_{\mathsf{samekey}}$ are stable by replacement of zeros under encryption: $\mathcal{M} \models P(t_1, t_2)[\{u\}^r_k \mapsto \{0^{l(u)}\}^r_k] \Leftrightarrow P(t_1, t_2)$ This yields the desired property. □

Now, once every encryption has been replaced by encryption of zeros then static equivalence coincides with equality up-to name renaming:

Lemma 18 *Let ϕ_1 and ϕ_2 be two frames such that for any subterm of ϕ_1 or ϕ_2 of the form $\{u\}^r_k$, we have $u = 0^l$ for some $l \in \mathbb{N}$. If $\mathsf{P}_{\mathsf{samekey}}$ is in the set of predicates, then $\phi_1 \sim \phi_2$ iff ϕ_1 and ϕ_2 are equal up-to name renaming.*

Proof: Only the direction "only if" needs a proof. Assume $\nu\overline{n_1}\sigma_1 \sim \nu\overline{n_2}\sigma_2$. Let the size of $\sigma = \{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$ be the sum of the sizes of the terms t_i. The size of a term is 1 when it is a constant (or a name) and, if $t = f(t_1, \ldots, t_n)$, then the size of t is $|t| = 1 + |t_1| + \cdots + |t_n|$. We reason by induction on the size of σ_1. When σ_1 is empty then σ_2 must be empty too. Now if, for some x, u, v, $x\sigma_1 =< u_1, v_1 >$, then, since $\mathcal{M} \models M(\pi_1(x\sigma_1))$, $\phi_1 \sim \phi_2$ implies $\mathcal{M} \models M(\pi_1(x\sigma_2))$, hence $x\sigma_2$ is a pair $< u_2, v_2 >$. If we let $\phi_i' = \nu\overline{n_i}.\sigma_i'$ be the frame defined by $x_1\sigma_i' = u_i$, $x_2\sigma_1' = v_i$, $y\sigma_i' = y\sigma_i$ if $y \in \mathsf{dom}(\sigma_i)\backslash\{x\}$ and whose domain is $(\mathsf{dom}(\sigma_i)\backslash\{x\}) \uplus \{x_1, x_2\}$, we have $\phi_1' \sim \phi_2'$: for every term t, $t\sigma_i' = (t\{x_1 \mapsto \pi_1(x), x_2 \mapsto \pi_2(x)\})\sigma_i$. Then we can apply the induction hypothesis and conclude. Now, we can assume that no variable in either frame is mapped to a pair. This means that, for every variable x, $x\sigma_i$ is either a ciphertext, a name, or a constant.

If, for some x, $x\sigma_1$ is a constant c, then $\mathcal{M}, \phi_1 \models EQ(x, c)$, hence $\mathcal{M}, \phi_2 \models EQ(x, c)$: $x\sigma_2 = c$. Then simply remove x from the domains of the two frames and apply the induction hypothesis.

If, for some x, y, $x \neq y$, $x\sigma_1 = y\sigma_1$, then $\mathcal{M}, \phi_1 \models EQ(x, y)$, hence $\mathcal{M}, \phi_2 \models EQ(x, y)$ and therefore $x\sigma_2 = y\sigma_2$. Then simply remove x from the domain of σ_i and apply the induction hypothesis. From now on, we may assume that $x\sigma_i \neq y\sigma_i$ as soon as $x \neq y$.

If, for some variable x, $x\sigma_1$ is a name k_1, then $x\sigma_2$ must be a name k_2; this can be checked in several ways.For instance, if it was a ciphertext then $\mathsf{P}_{\mathsf{samekey}}(x\sigma_2, x\sigma_2)$ would hold, while it would not be true for $x\sigma_1$. Consider then all variables whose image in σ_1 is an encryption with k_1: by renumbering we may assume it is $x_1, \ldots, x_m$. $x_i\sigma_1 = \{u_i^1\}_{k_1}^{r_i^1}$. $\mathcal{M}, \phi_1 \models M(\mathsf{dec}(x_i, x))$, hence $\mathcal{M}, \phi_2 \models M(\mathsf{dec}(x_i, x))$: there are terms such that $x_i\sigma_2 = \{u_i^2\}_{k_2}^{r_i^2}$. Then replacing σ_j with σ_j' such that $x_i'\sigma_j = u_i^j$ and replacing x_i with x_i', we have $\phi_1' \sim \phi_2'$ since $t\sigma_i' = (t\{x_1' \mapsto \mathsf{dec}(x_1, x), \ldots, x_m' \mapsto \mathsf{dec}(x_m, x)\})\sigma_i$.

If k_1 occurs only once in σ_1 (in this case $m = 0$), we simply remove x from the domains of σ_1 and σ_2: by induction hypothesis, the resulting frames are identical, up to renaming. Then adding $k_1 \mapsto k_2 = x\sigma_2$ to this renaming, we get a renaming of σ_1 into σ_2.

Now, assume k_1 occurs more than once. We can apply the induction hypothesis to ϕ_1', ϕ_2': these two frames are identical, up to a renaming ρ. In particular, we must have $k_1\rho = k_2$. In addition, by consistency of random numbers and since all variable images are distinct, the names $r_1^1, \ldots, r_m^1$ and $r_1^2, \ldots, r_m^2$ occur only once in their respective frames. Let $\rho' = \rho \uplus \{r_1^1 \mapsto r_1^2, \ldots, r_m^1 \mapsto r_m^2\}$. ρ' is a renaming of ϕ_1 into ϕ_2.

Finally consider the case where, for every x, $x\sigma_1$ is a ciphertext. Similarly, $x\sigma_2$ must be a ciphertext (we can apply the above reasoning switching ϕ_1 and ϕ_2). Let k_1 be a key occurring in σ_1 (there must be at least one). As above, let $x_1, \ldots, x_m$ be the variables such that $x_i\sigma_1 = \{u_i^1\}_{k_1}^{r_i^1}$. Now, let $x_1\sigma_2 = \{u_1^2\}_{k_2}^{r_1^2}$. $\mathcal{M}, \phi_1 \models \mathsf{P}_{\mathsf{samekey}}(x_1, x_i)$ for every $i = 2, \ldots, m$. It follows that $\mathcal{M}, \phi_2 \models \mathsf{P}_{\mathsf{samekey}}(x_1, x_i)$ for $i = 2, ..., m$. This implies that $x_i\sigma_2 = \{u_i^2\}_{k_2}^{r_i^2}$ for some r_i^2, u_i^2. Now, all symbols r_i^j must occur only once, by consistency of the random numbers and since any ciphertext occurs at most once in each frame. In addition, since ϕ_1, ϕ_2 satisfy the same EL predicates, for every i, the length of u_i^1 equals the length of u_i^2. On the other hand, we assumed that only terms 0^l have been encrypted. It follows that $u_i^1 = u_i^2$ for every i. Now, if we consider the frames ϕ_1', ϕ_2' obtained by removing $x_1, \ldots, x_m$ from the domain of σ_1, σ_2, there is no longer any occurrence of k_1, k_2, r_i^1, r_i^2 in ϕ_1', ϕ_2'. By induction hypothesis, there is a renaming ρ such that $\rho(\phi_1') = \phi_2'$. Then, extending ρ with $\{k_1 \mapsto k_2, r_1^1 \mapsto r_1^2, \ldots r_m^1 \mapsto r_m^2\}$, we get a renaming of ϕ_1 in ϕ_2.

□

7.2 Soundness of static equivalence on trees

Static equivalence on process trees can be transferred at a computational level.

Lemma 19 *Let P_1 and P_2 be two simple processes such that each P_i admits a key hierarchy. Let T_{P_i} be the process computation tree associated to P_i. If $T_{P_1} \sim T_{P_2}$ then $T_{P_1} \approx T_{P_2}$ or the encryption scheme is not joint IND-CPA and INT-CTXT.*

This key lemma is proved by applying the functions Ψ_k following the key ordering, to the trees T_{P_i}. We preserve equivalence on trees thanks to Lemma 17. If we find a key k such that $\Psi_k(T_{P_i}) \not\approx T_{P_i}$, then we can construct an attacker who breaks IND-CPA. Otherwise we are left to trees labeled with frames whose only subterms of the form $\{u\}^r_k$ are such that $u = 0^l$ for some l. In this case, we show that equivalence of such frames coincides with equality using Lemma 18.

Proof:

Assume there exists a PPT A that distinguishes T_{P_1} from T_{P_2} with non negligible probability. A makes at most a polynomial number of queries of the form (j, new). Thus a polynomial number n_1 (resp. n_2) of keys can be generated when A is interacting with T_{P_1} (resp. T_{P_2}).

Moreover, when A interacts with T_{P_i}, it only makes requests up to a certain depth $p(\eta)$ where p is a polynomial. Let $T_P/_n$ denote the tree obtained from T_P by removing nodes at depth greater than n. Due to our convention about the renaming of keys when processes are duplicated (see section 5.1), we have that $T_{P_1}/_{p(\eta)}$ (resp. $T_{P_2}/_{p(\eta)}$) contains at most n_1 (resp. n_2) distinct keys.

Let $<_i$ be the key ordering of P_i for $i \in \{1, 2\}$ and let $k_i^1, \ldots, k_i^{n_i}$ be the names of P_i such that $k_i^1 <_i k_i^2 <_i \cdots <_i k_i^{n_i}$. Let $T_i^1 = T_{P_i}$ and $T_i^{j+1} = \Psi_{k_i^j}(T_i^j)$ for $1 \leq j \leq n_j$. By Lemma 17, we have $T_i^{j+1} \sim T_i^j$. By transitivity of $\sim$, we deduce $T_1^{n_1} \sim T_2^{n_2}$. Then either A distinguishes $T_1^{n_1}$ from $T_2^{n_2}$ or A distinguishes T_i^{j+1} from T_i^j for some i, j with non negligible probability.

We first consider the case where A distinguishes T_i^{j+1} from T_i^j for some i, j with non negligible probability. We show that we can break IND-CPA or INT-CTXT for the key k_i^j. W.l.o.g, we assume $i = 1$. We construct a PPT B that simulates the oracle $\mathcal{O}_{T_1^j}$ in polynomial time. Initially B knows the process P_1 and generates all the keys $k_1^1, \ldots, k_1^{n_i}$ by itself except k_1^j. Then, each time A queries w where w is a bitstring then B computes the symbolic representation u of w as follows:

- Either w is the interpretation of new_j, and then the computation result is new_j
- Or w is a pair $w_1 \| w_2$, in which case B computes the symbolic representation of w_1 and w_2.
- Or w is an encryption with some key k. If B has already seen w then B already knows its symbolic interpretation. Otherwise, B decrypts w by using its own key if $k \neq k_1^j$ or by submitting w to the INT-CTXT oracle. If the decryption fails, B creates three new names u, k_0, r and remembers the association between w and its symbolic representation $\{k\}^r_{k_0}$. Otherwise, the decryption yields to a bitstring w' for which B computes a symbolic representation u. Then B creates a new name r and associate to w the symbolic representation $\{u\}^r_k$.
- Or w is a key. There are two cases, either B has already seen w in which case B already knows its symbolic interpretation or B creates a new name u and remembers the association between w and its symbolic representation u.
- Otherwise, we are left to the case where w does not start with the pair tag neither with the cyphertext tag or the key tag. There are two cases, either B has already seen w in which case B already knows its symbolic interpretation or B creates a new name u and remembers the association between w and its symbolic representation u.

Note that parsing performed by B has the following property (bottom-up parsing):

$$\text{if } u \text{ is a subterm of } \kappa^\tau_\eta(m), \text{ then } \kappa^\tau_\eta([\![u]\!]^\tau_\eta) = u. \tag{P}$$

Once B has computed the symbolic representation u of w, B looks up at the symbolic representation P of the current state of the process and computes the next state Q that is reached:

$P \xrightarrow{\mathsf{c_{in}}(u)} \xrightarrow{\overline{\mathsf{c_{out}}}(u_1)\cdots\overline{\mathsf{c_{out}}}(u_m)} Q$. It remains to B to give the computational counterpart of $u_1, \ldots, u_m$. B first identifies the maximal subterms of u_i for which it already knows the computational counterpart. Then the computational counterpart of $u_1, \ldots, u_m$ is computed by pairing and encrypting as follows: for each subterm of the form $\{v\}^r_k$:

- If $k <_1 k_i^j$, then B encrypts $[\![v]\!]$ with k yielding to a ciphertext c and remembers the association between c and $\{v\}^r_k$. Note that v cannot contain k as plaintext subterm due to the key hierarchy.
- If $k >_1 k_i^j$, then B encrypts $0^{l(v)}$ with k yielding to a ciphertext c and remembers the association between c and $\{v\}^r_k$.
- If $k = k_i^j$ then B submits to IND-CPA oracle the request $0^{l(v)}, [\![v]\!]$ and gets a ciphertext c. Again, B remembers the association between c and $\{v\}^r_k$.

When A halts outputting b then B halts outputting b. Then $A^{\mathcal{O}_{T_i^j}}$ behaves like $B^{\mathcal{O}_{k_1^j}}$ and $A^{\mathcal{O}_{T_i^{j+1}}}$ behaves like $B^{\mathcal{O}'_{k_1^j}}$, except when B submits an encryption to the INT-CTXT oracle. We deduce that B breaks the joint INT-CTXT and IND-CPA game.

We are left to the case where A distinguishes $T_1^{n_1}$ from $T_2^{n_2}$ with non negligible probability. Let β_1 be the one-to-one function from $T_1^{n_1}$ to $T_2^{n_2}$ and β_2 be the one-to-one function from $T_2^{n_2}$ to $T_1^{n_1}$. For any position p of $T_1^{n_1}$ not greater than $p(\eta)$, the frame of $T_1^{n_1}(p)$ is equivalent to the frame $T_2^{n_2}(\beta_1(p))$. From Lemma 18, we deduce that the two frames are equal modulo name renaming. Since the frames contain the adversary requests at the same position, it implies that β_1 is the identity (modulo name renaming). Symmetrically β_2 is the identity thus we deduce that $T_1^{n_1}$ and $T_2^{n_2}$ are identical up-to renaming and up-to depth $p(\eta)$. Thus they are *a fortiori* indistinguishable by A. □

7.3 Relating concrete and symbolic traces

We need here to show that concrete traces are, with overwhelming probability, interpretations of symbolic ones.

We let $\mathcal{P}$ be a simple process, τ be a sample of names and $\mathcal{F}$ be the family of CTMs obtained by interpreting $\mathcal{P}$, with respect to the sample τ (which we write by abuse of notations $[\![\mathcal{P}]\!]^\tau_\eta$). We define what it means for a concrete execution to be *fully abstracted* by a path in a process tree.

First, given $\mathcal{P}, \eta, \tau, A$, the behavior of the network $[\![\mathcal{P}]\!]^\tau_\eta \| \mathcal{A}$ is deterministic.

We let $\mathsf{Messages}(\mathcal{P}, \eta, \tau, A)$ be the sequence defined as follows: let γ_n^A be the sequence of configurations of A (along its deterministic computation), on the sample input τ, when interacting with $[\![\mathcal{P}]\!]^\tau_\eta$. Let δ_n^A be the subsequence of γ_n^A of configurations following a new, send or $\mathsf{receive}$ action.

- If δ_n^A follows a new action, let j be the content of the scheduling tape in δ_n^A and l be the content of the receiving tape in the same configuration. We let then $a_n = s(\mathsf{new}_j) \cdot r(l)$.
- If δ_n^A follows a send action of A, we let $a_n = s(< i, m >)$ where i is the content of the scheduling tape of A in configuration δ_n^A and m is the content of the sending tape in δ_n^A.
- If δ_n^A follows a $\mathsf{receive}$ action, then $a_n = r(m)$ where m is the content of the receiving tape in δ_n^A.

$\mathsf{Messages}(\mathcal{P}, \eta, \tau, A)$ is the sequence a_n obtained in this way.

Lemma 20 *We may assume w.l.o.g. (by changing the adversary) that* $\mathsf{Messages}(\mathcal{P}, \eta, \tau, A)$ *is a sequence* $s(m_1) \cdot R_1 \cdot s(m_2) \cdot R_2 \cdots s(m_n) \cdot R_n$ *where every* R_i *is a sequence of messages* $r(m'_1) \cdots r(m'_{n_i})$. *In other words, after a modification of the scheduling tape, there cannot be a* $\mathsf{receive}$ *event before a* send *event has been completed.*

In such a situation, $\mathsf{Messages}(\mathcal{P},\eta,\tau,A)$ can be more conveniently represented as a sequence $L_1 \xrightarrow{m_1} L_2 \cdots \xrightarrow{m_n} L_n$ where L_i is the increasing sequence of messages: $L_{i+1} = L_i \cdot R_i \cdot m_i$. We also denote the sequence of messages sent by A by $\mathsf{SMessages}(\mathcal{P},\eta,\tau,A) = (m_1,\ldots,m_n\ldots)$.

As for process computation trees, we will also insert in the sequence the configurations of the network in order to keep a stronger invariant relating the symbolic processes and their implementations along the computation. If

$$\mathcal{P} = \nu\overline{n},\forall\overline{x}.P_1\|\cdots\|P_n\|S_1\|\cdots\|S_k,$$

where $P_1,\ldots,P_n$ are basic processes, possibly with substitutions, and $S_1,\ldots,S_k$ are replicated processes, $\gamma([\![P_i]\!]^\tau)$ is the corresponding configuration $[q,\tau_i,\sigma]$ of the machine, as defined in section 4.

The configuration γ of the network is then given (besides the configuration of A) by the sequence $\gamma([\![P_i]\!]^\tau)$ for $i = 1,...,n$ (replicated processes do not play any role as long as there is no process activation; when a process is activated, it moves out of the scope of the bang).

Now the computation of the $A\|[\![\mathcal{P}]\!]^\tau$ can be split as follows:

$$\gamma_1, L_1 \xrightarrow{m_1} \gamma_2, L_2 \cdots \xrightarrow{m_n} \gamma_n, L_n$$

If T is a process computation tree and $p \in Pos(T)$, p *abstracts* such a computation sequence if $p = \alpha_1\cdots\alpha_n$ and, for every $j \le n$, $[\![\alpha_j]\!]^\tau = m_j$ and, if $T(\alpha_1\cdots\alpha_j) = (Q_j,\phi_j)$ $(= (Q_j,\nu\overline{n_j}.\,\mathcal{L}_j))$ then $[\![\phi_j]\!]^\tau$ $(= [\![\mathcal{L}_j]\!]^\tau) = L_j$ and $[\![Q_j]\!]^\tau = \gamma_j$.

Furthermore, p *fully abstracts* the computation if γ_n corresponds to a final configuration. (In which case, we succeeded in lifting the concrete trace to a symbolic one).

In other words, p fully abstracts a computation sequence if it defines a symbolic trace whose interpretation is that computation sequence.

The following characterizes situations in which computations cannot be fully abstracted:

Lemma 21 *Let $\mathcal{P}$ be a simple process and $T_\mathcal{P}$ its process computation tree. Given a sample τ and an attacker A, we let Γ be the computation sequence of $A\|[\![\mathcal{P}]\!]^\tau$.*

There is a path p in $Pos(T_\mathcal{P})$ such that one of the following holds:

1. *p fully abstracts Γ*
2. *or else $T_\mathcal{P}(p) = (Q,\phi)$, there is a a transition $\gamma_{n-1}, L_{n-1} \xrightarrow{m} \gamma_n, L_n$ in Γ such that $[\![\phi]\!]^\tau = L_{n-1}$ and there is a message m_1, which is polynomially computable from L_{n-1}, which is neither a pair, nor an encryption with a known key and such that $\phi \not\vdash \kappa_\eta^\tau(m_1)$.*
3. *or else there is a subterm v of $\phi \cup \{\kappa_\eta^\tau(m)\}$ and a name k such that $\mathsf{dec}(v,k)$ is in normal form, while $D([\![v]\!]_\eta^\tau, \tau(k)) \neq \bot$.*
4. *Or else there is a name k occurring in $\mathcal{P}$ and a term u occurring as a subterm in $\kappa_\eta^\tau(\mathsf{SMessages}(\mathcal{P},\eta,\tau A))$ such that $\tau(k) = [\![u]\!]_\eta^\tau$ and $k \neq u$.*
5. *Or else there is a term $\{v\}_k^r$ occurring as a subterm in $\kappa_\eta^\tau(\mathsf{SMessages}(\mathcal{P},\eta,\tau,A))$ and a term $\{v\}_k^{r'}$ such that $[\![\{v\}_k^r]\!]_\eta^\tau = [\![\{v\}_k^{r'}]\!]_\eta^\tau$.*

The proof relies on a number of simplification lemmas for conditions, whose proofs are given in appendix A and which we state below.

Lemma 22 *Every condition Φ is (effectively) logically equivalent to a conjunction of atomic formulas*

- *$M(s_1),\ldots M(s_n)$ such that, for every i, s_i does not contain any pairing or encryption symbol and does not contain any subterm $\mathsf{dec}(k,u)$ where k is a name*

- $EQ(t_1, u_1), \dots EQ(t_m, u_m)$ *such that, for every* $j = 1, ...n$ *there is a term* C *not containing any decryption or projection symbol and there are indexes* $i, i_1, \dots, i_k$ *such that* $\{t_j, u_j\} = \{s_i, C[s_{i_1}, \dots, s_{i_k}]\}$.

Lemma 23 *If* s *is a term in normal form and* $M(s)$ *is a simplified condition (as in lemma 22) and* θ *is a substitution such that, for every variable* x, $\mathcal{M} \models M(x\theta)$, *then, for every subterm* v *of* $s\theta\downarrow$, *if one of the following holds:*

- $v = \mathsf{dec}(w, k)$ *for some* w, k, *and* $\mathcal{M} \models M(w)$,
- $v = \pi_i(w)$ *for some* w *and* $\mathcal{M} \models M(w)$,

then there is a variable x *such that* w *is a subterm of* $x\theta$

Proof: **(Of lemma 21)**
Let Γ be the computation sequence

$$\gamma_1, L_1 \xrightarrow{m_1} \gamma_2, L_2 \cdots \xrightarrow{m_n} \gamma_n, L_n$$

For simplicity, we assume that there is no message new_j in the sequence of messages: these steps cannot have any influence on the trace extensibility. We first show that, if $\forall j \leq n-1$,

- $T_{\mathcal{P}}(\kappa_\eta^\tau(m_1) \cdots \kappa_\eta^\tau(m_j)) = (Q_j, \phi_j)$,
- $[\![\phi_j]\!] = L_j, [\![Q_j]\!] = \gamma_j$ and $\phi_j \vdash \kappa_\eta^\tau(m_{j+1})$,
- $Q_j \equiv R_j \| \sigma_j$ and $\sigma_j = \{x_1 \mapsto \kappa_\eta^\tau(m_1), \dots x_{n-1} \mapsto \kappa_\eta^\tau(m_{n-1})\}$

then $T_{\mathcal{P}}(\kappa_\eta^\tau(m_1) \cdots \kappa_\eta^\tau(m_n)) = (Q_n, \phi_n)$ with $[\![Q_n]\!] = \gamma_n$, $[\![\phi_n]\!] = L_n$ and $Q_n = R_n \| \sigma_n$ and $\sigma_n = \sigma_{n-1} \uplus \{x_n \mapsto \kappa_\eta^\tau(m_n)\}$ For, let $p = \kappa_\eta^\tau(m_1) \cdots \kappa_\eta^\tau(m_{n-1})$,

- $T_{\mathcal{P}}(p) = (Q_{n-1}, L_{n-1})$ and $Q_{n-1} \equiv Q \| \mathsf{c_{in}}(x_i).P_i \| \sigma_{n-1}$,
- in configuration γ_{n-1}, the machine M_i, whose pid is the interpretation of P_i's pid, is in state $[\mathsf{c_{in}}(x_i) \cdot P_i, \tau_i, [\![\theta_i]\!]^\tau]$, where θ_i is the restriction of σ_{n-1} to the variables of its domain that occur free in P_i.
- $P_i = \mathsf{if}\ \Phi\ \mathsf{then}\ \overline{\mathsf{c_{out}}}(s) \cdot Q\ \mathsf{else}\ \overline{\mathsf{c_{out}}}(\bot) \cdot \mathbf{0}$ (all tests can be gathered together in a single condition, without loss of generality)

  or else $P_i = \mathsf{c_{in}}(x_i') \cdot P_i'$

In the last case, our claim follows trivially. So, we discard this case in the following.

Let θ be $\theta_i \uplus \{x_i \mapsto \kappa_\eta^\tau(m_n)\}$. By hypothesis, for any variable x in the domain of θ, $\phi_{n-1} \vdash x\theta$. Since Γ is a computation sequence, $[\![\Phi]\!]^{\tau,\theta} = 1$.

If $M(s)$ is an atomic formula occurring in Φ and $\mathcal{M}, \theta \not\models M(s)$, then there is a subterm w of $s\theta\downarrow$ whose head symbol is a projection or a decryption symbol. Assume first that $w = \mathsf{dec}(v, k)$ is a minimal subterm of $s\theta\downarrow$ headed with a projection or a decryption. Since $[\![M(s)]\!]^{\tau,\theta} = 1$, $D([\![v]\!]_\eta^\tau, \tau(k)) \neq \bot$.

Moreover, v must be a subterm of some $x\theta$ by lemma 23, and we fall in case 3. The case $w = \pi_i(v)$ is not possible as we must have $\pi_i([\![v]\!]_\eta^\tau) \neq \bot$ (see the assumption on the computational interpretation of pairing, at the beginning of section 4).

Now, if $EQ(s, t)$ is an atomic formula occurring in Φ and $\mathcal{M}, \theta \not\models EQ(s, t)$, while $\mathcal{M}, \theta \models M(s) \wedge M(t)$, by lemma 22, either s or t only contains decryption symbols, projections and names and since $\mathcal{M}, \theta \models M(s) \wedge M(t)$, $s\theta\downarrow$ or $t\theta\downarrow$ is a subterm of some $x\theta$ by lemma 23. It follows that either $s\theta\downarrow$ or $t\theta\downarrow$ is a subterm of $\phi \cup \{\kappa_\eta^\tau(m)\}$.

Consider for instance that it is $s\theta\downarrow$. Moreover, $[\![s\theta\downarrow]\!]^\tau = [\![t\theta\downarrow]\!]^\tau$ by hypothesis and therefore, by (P): $s\theta\downarrow = \kappa_\eta^\tau([\![s\theta\downarrow]\!]_\eta^\tau) = \kappa_\eta^\tau([\![t\theta\downarrow]\!])$. Now, by lemma 22, $t = C[s_1, \dots, s_k]$ where C does

not contain the symbols $\mathsf{dec}, \pi_1, \pi_2$ and $s_1, \ldots, s_k$ only contain decryption, projections, names and variables.

We prove by induction on C that, if $[\![t\theta]\!]^\tau_\eta$ is equal to a some $[\![u]\!]^\tau_\eta$ where u is a subterm of some $x\theta$ and $t\theta \neq u$, then we fall in either case 3 or 4 or case 5 of the lemma.

If C is empty, then, by lemma 23, $t\theta \downarrow$ is a subterm of some $x\theta$, which is not possible, since, in that case, thanks to property (P), $t\theta \downarrow = \kappa^\tau_\eta([\![t\theta]\!]^\tau_\eta) = \kappa^\tau_\eta([\![u]\!]^\tau_\eta) = u$.

If t is a name k_1, then we fall in case 4 of the lemma.

If t is a pair, then u must be a pair and we use the induction hypothesis: one of the direct subterms of $t\theta$ is distinct from the corresponding subterm of u, while their interpretations are identical.

Now, if $t = \{t_1\}^{r_1}_{k_1}$, u cannot be a pair. If it is a name, then we fall in case 3 of the lemma. If it is a ciphertext $u = \{u_1\}^{r_2}_{k_2}$, either $k_1 \neq k_2$ and we fall again in case 3 of the lemma or else $k_1 = k_2$. In this latter case, either $t_1 \neq u_1$ and we apply the induction hypothesis, or else $t_1 = u_1$ and we fall in the case 5 of the lemma.

Now, to complete the proof of our preliminary statement, if we are not in case 3 or case 4, $\mathcal{M}, \theta \models \Phi$, since Φ is a conjunction of atomic formulas of the forms $M(s)$ or $EQ(s,t)$. This can be iterated if Q itself is a conditional statement (we omit the details here). Then

$$Q_{n-1} \xrightarrow{\mathsf{c_{in}}(\kappa^\tau_\eta(m_n))} \xrightarrow{\overline{\mathsf{c_{out}}}(s_1),\ldots,\overline{\mathsf{c_{out}}}(s_q)} Q_n$$

and, if we let $\phi_n = \nu\overline{k_n}.\,\mathcal{L}_n$, $\phi_{n-1} = \nu\overline{k_{n-1}}.\,\mathcal{L}_{n-1}$, $\mathcal{L}_n = \mathcal{L}_{n-1} \cdot (\kappa^\tau_\eta(m_n), s_1, \ldots, s_q)$, there is indeed a position $p' = m_1 \cdots m_n$ in $T_\mathcal{P}$ such that $T_\mathcal{P}(p') = (Q_n, \phi_n)$ with the desired properties.

Now, we have completed the proof of our preliminary statement. Then, either there is a path p such that $T_\mathcal{P}(p) = (Q, \phi)$ and there is a transition $\gamma_{n-1}, L_{n-1} \xrightarrow{m} \gamma_n, L_n$ in Γ such that $[\![\phi]\!]^\tau_\eta = L_{n-1}$ and $\phi \not\vdash \kappa^\tau_\eta(m)$, or else, from the preliminary statement, we can construct, by induction on the length of Γ, a path p which fully abstracts Γ.

In case $\kappa^\tau_\eta(m_n)$ is not deducible, by induction on its size, we construct m, constructable in polynomial time from m_n and such that m is not a pair or an encryption with a known key: in either of the latter cases, we can project or decrypt, getting a smaller term.

□

Concrete traces can be lifted to symbolic ones with overwhelming probability:

Lemma 24 *Assume that the encryption scheme is INT-CTXT and IND-CPA. Let $\mathcal{P}$ be a simple process that admits a key ordering and $T_\mathcal{P}$ be its process computation tree. Let A be a concrete attacker. The probability over all samples τ, that there is a path p in $T_\mathcal{P}$ fully abstracting the computation sequence of $A \| [\![\mathcal{P}]\!]^\tau$ is overwhelming.*

To prove this lemma, we first simplify the trees by applying the functions Ψ_k thanks to Lemmas 19 and 17. Then, using the lemma 21, in each case, in which a trace cannot be fully abstracted, we break either INT-CTXT or IND-CPA.

Proof: From an attacker A, we construct a machine B, which is a scheduler of the tree $T_\mathcal{P}$ (or, more precisely, the tree in $T_\mathcal{P}$ in which all symbolic traces that have no concrete instances, are removed). B behaves as A, except that he gets his answer from the oracle $\mathcal{O}_{T_\mathcal{P},\tau}$. Moreover, if there is no successor in $T_\mathcal{P}$ corresponding to A's request (the trace cannot be lifted any more), then we assume that the oracle $\mathcal{O}_{T_\mathcal{P},\tau}$ sends an error message which depends on the reason of the failure (ϕ is defined as in lemma 21):

- $(\texttt{guessed ciphertext}, m_1)$ if a component of the request m is a message m_1 such that $\kappa^\tau_\eta(m_1) = \{w\}^r_k$, and $\phi \not\vdash \{w\}^r_k$, $\phi \not\vdash k$ or $\kappa^\tau_\eta(m_1) = k$, and $\phi \not\vdash k$.
- $(\texttt{confusion}, [\![v]\!]^\tau_\eta)$ if there is a subterm v of $\phi \cup \{\kappa^\tau_\eta(m)\}$ such that $\mathsf{dec}(v,k)$ is in normal form and $D([\![v]\!]^\tau_\eta, \tau(k)) \neq \perp$.

- $(\texttt{collision1}, [\![u]\!]^\tau_\eta)$ if u is a term in $\kappa^\tau_\eta(\mathsf{SMessages}(\mathcal{P}, \eta, \tau, A))$ and a name k occurring in $\mathcal{P}$ such that $\tau(k) = [\![u]\!]^\tau_\eta$ and $k \neq u$.
- $(\texttt{collision2}, m)$, if $\kappa^\tau_\eta(m) = \{v\}^r_k$ occurs as a subterm of $\kappa^\tau_\eta(\mathsf{SMessages}(\mathcal{P}, \eta, \tau, A))$ and there is a term $\{v\}^{r'}_k$, $r \neq r'$ such that $m = [\![\{v\}^{r'}_k]\!]^\tau_\eta$
- `other` otherwise

In further transformations of the machine and the tree, the above properties might no longer be satisfied. That is why, in addition, upon receiving any of these error messages, B checks the corresponding message: in case of $(\texttt{guessed ciphertext}, m_1)$, it checks that $m_1 \notin [\![\phi]\!]_\eta$. If it is indeed the case, it stops outputting 1. Otherwise, it outputs 0. In case of $(\texttt{confusion}, m)$, it checks that there is a subterm v of $\phi \cup \{\kappa^\tau_\eta(m)\}$ and a name k such that $[\![v]\!]^\tau_\eta = m$ and $D(m, \tau(k)) \neq \bot$. If it is the case, the machine outputs 1, otherwise it outputs 0. In case of $(\texttt{collision1}, m)$, again B checks the condition and outputs 1 if it is satisfied, otherwise it outputs 0. In case of `collision2`, there is no possible verification and B simply outputs 1. All these verifications can be performed in polynomial time.

Otherwise, on any fully abstract trace, B stops outputting 0.

$$\epsilon = \mathbf{Pr}\{\tau, r : B^{\mathcal{O}_{T_\mathcal{P}, \tau}}(0^\eta \mid r) = 1\}$$

is the probability that, given a sample τ, the corresponding concrete trace Γ cannot be fully abstracted.

From lemma 21, the above failure cases are the only possible ones. Therefore there is a name k (chosen from $P(\eta)$ possible names) such that for some $i \in \{1, 2, 3, 4\}$:

$$\mathbf{Pr}\{\tau, r : B^{\mathcal{O}_{T_\mathcal{P}, \tau}}_{i,k}(0^\eta \mid r) = 1\} > \frac{\epsilon}{4 \times P(\eta)}$$

where $B_{1,k}$ (resp. $B_{2,k}$, resp. $B_{3,k}$, resp. $B_{4,k}$) is the machine, which behaves as B, except that it outputs 1 only in case of `guessed ciphertext` (resp. only in case of `confusion`, resp. only in case of `collision1`, resp. only in case `collision2`).

Now, let T_0 be the tree obtained by applying $\Psi_{k_1} \cdots \Psi_{k_n}$ to $T_\mathcal{P}$, as in the proof of lemma 19. Note that the oracle may not reply the same error message when the adversary is interacting with T and with $\Psi_k(T)$: first it may be the case that on one tree there is an error message and not in the other. Next, the final verifications may succeed in one case and not in the other, as for instance m may no longer belong to $\Psi_k(\phi)$; this was the purpose of the verification phase.

As in the proof of lemma 19, either we break joint INT-CTXT and IND-CPA or else, for some $i \in \{1, 2, 3, 4\}$ and some key k,

$$\mathbf{Pr}\{\tau, r : B^{\mathcal{O}_{T_0, \tau}}_{i,k}(0^\eta \mid r) = 1\} > \frac{\epsilon}{4 \times P(\eta)^2}$$

Let us construct, in each case, a machine which breaks joint INT-CTXT and IND-CPA.

The machine $C_{i,k}$ behaves as $B_{i,k}$, but first draws all names except k, and, instead of querying the oracle $\mathcal{O}_{T_0^k, \tau}$, it performs several computations (as in the proof of lemma 19: first parse the request of $B_{i,k}$ possibly stopping if INT-CTXT is broken. Then retrieve from the process component what is the (symbolic) expected reply $s_1, \ldots, s_r$. Then compute $t_i = \Psi_{k_1} \ldots \Psi_{k_m}(s_i)$. These terms do not contain any occurrence of k, except in expressions $\{0^\alpha\}^r_k$. Then compute $[\![t_1, \ldots, t_r]\!]^\tau$, by computing all messages which do not contain k, using the sampled keys and querying the encryption oracle for each encryption with k, which the attacker does not already have.

Finally, we distinguish between the three cases:

1. $i = 1$: if $B_{1,k}$ outputs 1, then there is two cases.

- Either it got at the last stage a message m from which it can extract m_1 such that $m_1 = [\![\{w\}_k^r]\!]_\eta^\tau$ and $\{w\}_k^r$ is not deducible from $\Psi_{k_1} \cdots \Psi_{k_m}(\phi)$. In that case, since we applied all the functions Ψ_{k_i}, $\{w\}_k^r$ cannot occur as a subterm in $\Psi_{k_1} \cdots \Psi_{k_m}(\phi)$ thus $[\![w]\!]_\eta^\tau$ has never been sent as a query to the encryption oracle. So, $C_{1,k}$ simply submits m_1 to the INT-CTXT oracle.
- Or it got at the last stage a message m from which it can extract m_1 such that $m_1 = [\![k]\!]_\eta^\tau$ and k is not deducible from $\Psi_{k_1} \cdots \Psi_{k_m}(\phi)$. Then $C_{1,k}$ builds an encrypted message $m_2 = [\![\{0^\alpha\}_k^r]\!]_\eta^\tau$ using the key $m_1 = [\![k]\!]_\eta^\tau$ and submits m_2 to the INT-CTXT oracle.

2. $i = 2$: if $B_{2,k}$ outputs 1, we break "confusion freeness", hence INT-CTXT (see [35]).
3. $i = 3$: Either the term u can be deduced from $\kappa_\eta^\tau(\mathsf{SMessages}(\mathcal{P}, \eta, \tau, A))$, in which case the attacker can guess $\tau(k)$ and break INT-CTXT, or else u is encrypted with k. But in that case $[\![u]\!]_\eta^\tau = \tau(k)$ must be 0^α, a case which cannot occur with probability $\frac{\epsilon}{4 \times P(\eta)^2}$.
4. $i = 4$: the machine $C_{4,k}$ queries the IND-CPA encryption oracle with 1^α, checks the equality with the term at hand $[\![\{0^\alpha\}_k^r]\!]_\eta^\tau$. In case of equality, he deduces that he must be talking with the oracle encrypting always 0^α. The guess is correct if $[\![\{0^\alpha\}_k^r]\!] = [\![\{0^\alpha\}_k^{r'}]\!]$, and only in this case, since we cannot have $[\![\{1^\alpha\}_k^r]\!] = [\![\{0^\alpha\}_k^{r'}]\!]$ because of the determinacy of decryption. This occurs with probability at least $\frac{\epsilon}{4 \times P(\eta)^2}$, when r, r' are chosen randomly: we break IND-CPA.

□

7.4 Concluding the proof of the main theorem

In the last lemma, we simply apply the previous result. Since traces can be lifted, an attacker on the concrete processes is actually a scheduler of the computation trees, hence distinguishing the concrete processes amounts to distinguish the corresponding computation trees.

Lemma 25 *Let P_1 and P_2 be two simple processes admitting a key hierarchy. Let T_{P_i} be the process computation tree associated to P_i. If the encryption scheme is joint IND-CPA and INT-CTXT, then $T_{P_1} \approx T_{P_2}$ implies that $[\![P_1]\!] \approx [\![P_2]\!]$.*

Proof: If A is a CTM, which distinguishes between $[\![P_1]\!]_\eta$ and $[\![P_2]\!]_\eta$, let

$$\epsilon = |\mathbf{Pr}\{\tau, r : A \| [\![P_1]\!]_\eta^\tau(0^\eta \mid r) = 1\} - \mathbf{Pr}\{\tau, r : A \| [\![P_2]\!]_\eta^\tau(0^\eta \mid r) = 1\}|$$

From lemma 24, for every $i \in \{1, 2\}$, the probability (over τ, r) that there is a path p in T_{P_i} fully abstracting the computation sequence Γ_τ^i of $A \| [\![P_i]\!]_\eta^\tau$ is overwhelming. By definition, if p fully abstracts Γ_τ^i, then A behaves as a scheduler of T_{P_i} on this sample input:

$$\mathbf{Pr}\{\tau, r : A^{\mathcal{O}_{T_{P_i}, \tau}}(0^\eta \mid r) = 1\} - \mathbf{Pr}\{\tau, r : A \| [\![P_i]\!]_\eta^\tau(0^\eta \mid r) = 1\}$$

is negligible.

It follows that

$$|\mathbf{Pr}\{\tau, r : A^{\mathcal{O}_{T_{P_1}, \tau}}(0^\eta \mid r) = 1\} - \mathbf{Pr}\{\tau, r : A^{\mathcal{O}_{T_{P_2}, \tau}}(0^\eta \mid r) = 1\}| + \nu_1 + \nu_2 \geq \epsilon$$

where ν_1, ν_2 are negligible.

Then, if $T_{P_1} \approx T_{P_2}$, ϵ is also negligible. □

Theorem 12 is now a straightforward consequence of Lemma 15, 19 and 25.

8 Future work

First, there are several possible extensions. Following our proof scheme, we believe that our results can be extended to other security primitives, e.g. public-key encryption or signatures.

There are harder extensions. For instance, can we drop the requirement that private keys are not dynamically disclosed? As explained in [10], there is a commitment problem if we rely on a simulator. We could also extend our results to a wider class of equational theories by extending in particular Lemma 19.

Another possible extension is to allow the attacker to generate encryption keys, which are then not necessarily obtained through the key generation algorithm. Currently, we assumed that the adversary gets keys, upon requesting corrupted users. As explained in section 5.1, this is a non trivial question, which may require an extension of the symbolic model.

Finally, our results could be extended to a wider class of processes, for instance allowing a non-trivial else branch in the conditionals. Such extensions may be not very hard.

Besides extensions of the result, we also wish to apply it to actual protocols, for instance voting protocols, getting computational security guarantees.

We also wish to investigate proofs of simulatability/universal composability, relying on our abstraction of indistinguishability.

Acknowledgments

We thank Steve Kremer and Bogdan Warinschi for many useful comments they gave on earlier drafts of this paper.

References

[1] M. Abadi, M. Baudet, and B. Warinschi. Guessing attacks and the computational soundness of static equivalence. In *Foundations of Software Science and Computation Structure (FoSSaCS'06)*, volume 3921 of *LNCS*, pages 398–412, 2006.

[2] M. Abadi and V. Cortier. Deciding knowledge in security protocols under (many more) equational theories. In *Computer Security Foundations Workshop (CSFW'05)*, pages 62–76, 2005.

[3] M. Abadi and C. Fournet. Mobile values, new names, and secure communication. In *Principles of Program- ming Languages (POPL'01)*, pages 104–115, 2001.

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

[5] M. Abadi and P. Rogaway. Reconciling two views of cryptography: the computational soundness of formal encryption. In *Int. Conf. on Theoretical Computer Science*, volume 1872 of *LNCS*, Sendai, Japan, 2000.

[6] M. Abdalla and B. Warinschi. On the minimal assumptions of group signature schemes. In *6th International Conference on Information and Communication Security*, pages 1–13, 2004.

[7] P. Adão and C. Fournet. Cryptographically sound implementations for communicating processes. In *International Colloquium on Algorithms, Languages and Programming (ICALP'06)*, 2006.

[8] A. Armando and et al. The AVISPA Tool for the automated validation of internet security protocols and applications. In *Computer Aided Verification, (CAV'05)*, volume 3576 of *LNCS*, pages 281–285, 2005.

[9] M. Backes, M. Dürmuth, and R. Küsters. On simulatability soundness and mapping soundness of symbolic cryptography. In *Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, 2007.

[10] M. Backes and B. Pfitzmann. Symmetric encryption in a simulatable Dolev-Yao style cryptographic library. In *Computer Security Foundations Workshop (CSFW'04)*, 2004.

[11] M. Backes and B. Pfitzmann. Relating cryptographic und symbolic key secrecy. In *Symp. on Security and Privacy (SSP'05)*, pages 171–182, 2005.

[12] M. Backes, B. Pfitzmann, and M. Waidner. A com- posable cryptographic library with nested operations. In *10th ACM Concerence on Computer and Communications Security (CCS'03)*, 2003.

[13] M. Backes, B. Pfitzmann, and M. Waidner. The reactive simulatability (RSIM) framework for asynchronous systems. *Information and Computation*, 205(12):1685–1720, 2007.

[14] G. Bana. Soundness and completeness of formal logics of symmetric encryption. Cryptology ePrint Archive, Report 2005/101, 2005. `http://eprint.iacr.org/`.

[15] M. Baudet, V. Cortier, and S. Kremer. Computationally sound implementations of equational theories against passive adversaries. In *Int. Colloquium on Automata, Languages and Programming (ICALP'05)*, volume 3580 of *LNCS*, 2005.

[16] 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 *LNCS*, pages 531–545, 2000.

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

[18] 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, 2008.

[19] M. Burrows, M. Abadi, and R. Needham. A logic of authentication. Technical Report 39, Digital Systems Research Center, February 1989.

[20] R. Canetti. Universal composable security: a new paradigm for cryptographic protocols. In *Symposium on Foundations of Computer Science*, 2001.

[21] R. Canetti and J. Herzog. Universally composable symbolic analysis of cryptographic protocols. In *Theory of Cryptography Conference (TCC'06)*, 2006.

[22] R. Canetti and T. Rabin. Universal composition with joint state. Cryptology ePrint Archive, report 2002/47, Nov. 2003.

[23] V. Cortier, S. Kremer, R. Küsters, and B. Warinschi. Computationally sound symbolic secrecy in the presence of hash functions. In *Fundations of Software Technology and Theoretical Computer Science (FSTTCS'06)*, volume 4337 of *LNCS*, pages 176–187, 2006.

[24] V. Cortier and B. Warinschi. Computationally sound, automated proofs for security protocols. In *European Symposium on Programming (ESOP'05)*, volume 3444 of *LNCS*, pages 157–171, 2005.

[25] S. Delaune, S. Kremer, and M. D. Ryan. Coercion-resistance and receipt-freeness in electronic voting. In *Proceedings of the 19th Computer Security Foundations Workshop (CSFW'06)*, pages 28–39, Venice, Italy, July 2006. IEEE Computer Society Press.

[26] S. Delaune, S. Kremer, and M. D. Ryan. Symbolic bisimulation for the applied pi-calculus. In *Foundations of Software Technology and Theoretical Computer Science (FSTTCS'07)*, volume 4855 of *LNCS*, pages 133–145, 2007.

[27] H. Hüttel. Deciding framed bisimilarity. In *Proc. INFINITY'02*, 2002.

[28] R. Janvier, Y. Lakhnech, and L. Mazaré. Completing the picture: Soundness of formal encryption in the presence of active adversaries. In *European Symposium on Programming (ESOP'05)*, volume 3444 of *LNCS*, pages 172–185. Springer, 2005.

[29] R. Janvier, Y. Lakhnech, and L. Mazare. (de)compositions of cryptographic schemes and their applications to protocols. Cryptology ePrint Archive, Report 2005/020, 2005.

[30] R. Janvier, Y. Lakhnech, and L. Mazaré. Computational soundness of symbolic analysis for protocols using hash functions. In *Workshop on Information and Computer Security (ICS'06)*, ENTCS, 2006.

[31] A. Juels, M. Luby, and R. Ostrovsky. Security of blind digital signatures. In *advances in cryptology, (CRYPTO-97)*, volume 1294 of *LNCS*, pages 150–164, 1997.

[32] S. Kremer and L. Mazaré. Adaptive soundness of static equivalence. In *European Symposium on Research in Computer Security (ESORICS'07)*, volume 4734 of *LNCS*, pages 610–625, 2007.

[33] R. Küsters and M. Tuengerthal. Joint state theorems for public-key encryption and digital signature functionalities with local computations. In *Computer Security Foundations (CSF'08)*, 2008.

[34] P. Laud. Symmetric encryption in automatic analyses for confidentiality against active adversaries. In *Symp. on Security and Privacy (SSP'04)*, pages 71–85, 2004.

[35] D. Micciancio and B. Warinschi. Completeness theorems for the Abadi-Rogaway language of encrypted expressions. *Journal of Computer Security*, 2004.

[36] D. Micciancio and B. Warinschi. Soundness of formal encryption in presence of an active attacker. In *Theory of Cryptography Conference (TCC'04)*, volume 2951 of *LNCS*, 2004.

[37] J. Mitchell, A. Ramanathan, and V. Teague. A probabilistic polynomial-time process calculus for the analysis of cryptographic protocols. *Theoretical Comput. Sci.*, 353:118–164, 2006.

Rules for M:

$$\begin{array}{rcll}
M(<u,v>) & \rightarrow & M(u) \wedge M(v) & \\
M(\{u\}^r_k) & \rightarrow & M(u) \wedge M(k) \wedge M(r) & \\
M(s) & \rightarrow & \top & \text{if } s \in \mathcal{N} \\
M(s[\pi_i(\{u\}^r_w)]) & \rightarrow & \bot & \\
M(s[\pi_i(k)]) & \rightarrow & \bot & \text{if } k \in \mathcal{N} \\
M(s[\mathsf{dec}(<u,v>,w)]) & \rightarrow & \bot & \\
M(s[\mathsf{dec}(u,<v,w>)]) & \rightarrow & \bot & \\
M(s[\mathsf{dec}(u,\{v\}^r_w)]) & \rightarrow & \bot & \\
M(s[\mathsf{dec}(k,u)]) & \rightarrow & \bot & \text{if } k \text{ is a name} \\
M(s[\mathsf{dec}(\{u\}^r_v,w)]) & \rightarrow & \bot & \text{if } v,w \in \mathcal{N} \text{ and } v \neq w \\
M(s[\mathsf{dec}(\{u\}^y_x,z)]) & \rightarrow & M(s[u]) \wedge M(x) \wedge M(y) \wedge M(z) \wedge EQ(x,z) &
\end{array}$$

Rules for EQ:

$$\begin{array}{rcll}
EQ(<u_1,v_1>,<u_2,v_2>) & \rightarrow & EQ(u_1,u_2) \wedge EQ(v_1,v_2) & \\
EQ(<u_1,v_1>,\{u_2\}^r_{v_2}) & \rightarrow & \bot & \\
EQ(<u_1,v_1>,k) & \rightarrow & \bot & \text{if } k \in \mathcal{N} \\
EQ(\{u_1\}^{r_1}_{v_1},\{u_2\}^{r_2}_{v_2}) & \rightarrow & EQ(u_1,u_2) \wedge EQ(v_1,v_2) \wedge EQ(r_1,r_2) & \\
EQ(\{u_1\}^{r_1}_{v_1},k) & \rightarrow & \bot & \text{if } k \in \mathcal{N} \\
EQ(x,x) & \rightarrow & M(x) & \\
EQ(s,t) & \rightarrow & \bot & \text{if } M(s) \stackrel{*}{\rightarrow} \bot \text{ or } M(t) \stackrel{*}{\rightarrow} \bot \\
EQ(s[\mathsf{dec}(\{u\}^y_x,z)],t) & \rightarrow & EQ(s[u],t) \wedge EQ(x,z) & \\
EQ(t,s[\mathsf{dec}(\{u\}^y_x,z)]) & \rightarrow & EQ(t,s[u]) \wedge EQ(x,z) &
\end{array}$$

Figure 5: Simplification of atomic formulas

A Simplifying the atomic formulas

A.1 Proof of lemma 22

Lemma 22. *Every condition Φ is (effectively) logically equivalent to a conjunction of atomic formulas*

- *$M(s_1), \ldots M(s_n)$ such that, for every i, s_i does not contain any pairing or encryption symbol and does not contain any subterm $\mathsf{dec}(k,u)$ where k is a name*
- *$EQ(t_1,u_1), \ldots EQ(t_m,u_m)$ such that, for every $j = 1,...n$ there is a term C not containing any decryption or projection symbol and there are indexes $i, i_1, \ldots, i_k$ such that $\{t_j,u_j\} = \{s_i, C[s_{i_1}, \ldots, s_{i_k}]\}$.*

Proof: First, we may assume w.l.o.g. that for all atomic formulas $M(s)$ or $EQ(s,t)$, s,t are in normal form (replacing a term with its normal form does not change the models). Next, for every formula $EQ(s,t)$ we add the atomic formulas $M(s)$ and $M(t)$. We use then the transformation rules displayed in figure 5. The rules terminate (the multiset of sizes of arguments of predicate symbols is strictly decreasing at each step). They are correct: $\mathcal{M}$ satisfies a left side iff it satisfies a right side. Finally, the normal forms satisfy the conclusion of the lemma:

- Each time a destructor (a symbol from $\mathsf{dec}, \pi_1, \pi_2$) is applied to a non-variable term, which is not headed itself by a destructor, there is at least one applicable rule. This is explicitly stated by pattern matching for the M predicate and covered by the 3 last rules for the EQ predicate symbol.

- M cannot be applied to a term headed with a constructor (thanks to the first 3 rules)
- EQ cannot be applied to two terms headed with constructors (thanks to the first rules of EQ)

□

In what follows, we assume this simplified version of conditions.

A.2 A useful lemma

Lemma 23. *If s is a term in normal form and $M(s)$ is a simplified condition (as in lemma 22) and θ is a substitution such that, for every variable x, $\mathcal{M} \models M(x\theta)$, then, for every subterm v of $s\theta\downarrow$, if one of the following holds:*

- *$v = \mathsf{dec}(w, k)$ for some w, k, and $\mathcal{M} \models M(w)$,*
- *$v = \pi_i(w)$ for some w and $\mathcal{M} \models M(w)$,*

then there is a variable x such that w is a subterm of $x\theta$.

Proof: We rely on normal forms of conditions (lemma 22). We prove the property for any term t such that $s\theta \xrightarrow{*} t$, by induction on the length of the reduction sequence.

If there is no reduction step, let $v = \mathsf{dec}(w, k)$ (resp. $v = \pi_i(w)$) be a subterm of $s\theta$. Since $\mathcal{M} \models M(w)$ and by lemma 22, w must be a subterm of some $x\theta$.

Now, assume the property true for t and that $t \to t'$. Any subterm $t'|_p = \mathsf{dec}(w, k)$ (resp. $\pi_i(w)$) of t' is either a subterm of t, or else we are in one of the following cases:

- $t|_p = \mathsf{dec}(\mathsf{dec}(\{w\}^r_{k'}, k'), k)$
- $t|_p = \mathsf{dec}(w, \mathsf{dec}(\{k\}^r_{k'}, k'))$
- $t|_p = \mathsf{dec}(\pi_1(< w, w' >), k)$
- $t|_p = \mathsf{dec}(\pi_2(< w', w >), k)$
- $t|_p = \mathsf{dec}(w, \pi_1(< k, w' >))$
- $t|_p = \mathsf{dec}(w, \pi_2(< w', k >))$

In all cases, we can apply the induction hypothesis, for instance consider the two first cases: In the first case, by induction hypothesis $\{w\}^k_{k'}$ is a subterm of some $x\theta$, hence w is a subterm of $x\theta$. In the second case, w is a subterm of $x\theta$ by induction hypothesis.

□

Centre de recherche INRIA Nancy – Grand Est
LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)

Centre de recherche INRIA Bordeaux – Sud Ouest : Domaine Universitaire - 351, cours de la Libération - 33405 Talence Cedex
Centre de recherche INRIA Grenoble – Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier
Centre de recherche INRIA Lille – Nord Europe : Parc Scientifi que de la Haute Borne - 40, avenue Halley - 59650 Villeneuve d'Ascq
Centre de recherche INRIA Paris – Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex
Centre de recherche INRIA Rennes – Bretagne Atlantique : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex
Centre de recherche INRIA Saclay – Île-de-France : Parc Orsay Université - ZAC des Vignes : 4, rue Jacques Monod - 91893 Orsay Cedex
Centre de recherche INRIA Sophia Antipolis – Méditerranée : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399