Computational Soundness of Observational Equivalence

Hubert Comon-Lundh
Research Center for Information Security and
ENS Cachan
AIST, Akihabara-Daibiru, Tokyo, Japan
h.comon-lundh@aist.go.jp

Véronique Cortier[*]
LORIA, CNRS & INRIA project CASSIS
Nancy, France
cortier@loria.fr

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.

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.

Categories and Subject Descriptors

D.2.4 [**Verification**]: Formal methods

General Terms

Verification

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 [14] 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 [4], 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, 12, 31] 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. [8, 18, 22, 30]). 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 [10, 32].
There are however several security properties, which cannot be defined (or cannot be naturally defined) as trace properties.

- 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 [23] also use an equivalence and cannot be expressed in linear temporal logic.

- Similarly, in the computational worlds, anonymity of group signatures [5] 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 [27].

- The "computational secrecy" states that the protocol does not leak any piece of the secret (this is sometimes

[*]This work has been partially supported by the ARA SSIA FormaCrypt and the ARA project AVOTÉ.

CCS'08 October 27–31, 2008, Alexandria, Virginia, USA.

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 [32, 9, 22, 26, 18, 21]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 [11, 29], 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* [17, 19] 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 [15, 24].

We prove our result for symmetric encryption and pairing, using a fragment of the applied pi-calculus [2] for specifying protocols and relying on standard cryptographic assumptions (IND-CPA and INT-CTXT) as well as hypotheses, which are similar to those of [8]. 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 [22, 26], 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. [4, 1, 12, 31]). 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 [28] 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 [32] and continued with e.g. [22, 26], 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.

There is a huge amount of work on simulatability/universal composability, especially the work of Backes *et. al.* and Canetti [17, 11, 10, 8, 9]. In the black-box simulatability approach of [11], which we refer to as BPW, the symbolic model is different than ours: there are essential constructions such as handles, which we do not have in our (more abstract) symbolic model, that is a standard process algebra. The BPW model also requires to construct a simulator, within the model, which we do not require. Therefore, we must be cautious with any comparison.

BPW-simulatability roughly states that $[\![P]\!] \approx P\|S$: the computational interpretation of the process P is indistinguishable from the simulated version of P. As shown in [7], this implies the trace mapping property, hence that symbolic trace properties transfer to the computational level. The BPW-simulatability should also imply the soundness of observational equivalence of the corresponding BPW-processes in a simple way (D. Unruh, private communication). The precise relationships with our work are worth being further investigated.

Conversely, if a simulated process $S\|P$ could be seen as the computational interpretation of a process Q, then the BPW-simulatability itself could be seen as an instance of our result.

Our work can also be seen as a generalization of soundness results for static equivalence [4, 3, 12] 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 [18] 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.* [33] 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.

To our knowledge, the only previous work formally connecting observational equivalence and computational indistinguishability is [6]. 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 [6] are useful in designing secure implementations of abstract functionalities, not for the verification of existing protocols.

Finally, the work on automation and decision of observational equivalence [25, 15, 24] shows that there exist systematic ways of deriving such equivalences in the symbolic world. This is also the advantage of using a standard process algebra as a symbolic model.

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. More details, including intermediate lemmas, the notions of computation trees, tree oracles, tree soundness are provided in section 6. We omit details and proofs in this short paper: they can be found in [20].

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 [11] for a discussion). The adversary is a special CTM with an additional *scheduling tape.* A *network* $\mathcal{F}\|A$ consists of an adversary A and a family of CTMs $\mathcal{F} = (\mathcal{M}_n)_{n\in\mathbb{N}}$. We also call $\mathcal{F}$ the *environment* of A. This model is a simplification of interactive Turing machines of [17], keeping only the essential features.

In brief, in the *initial configuration*, each machine of the network has the security parameter in unary on its input tape and possibly other data 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. The number of CTMs in the network is unbounded. Note that this setting does allow dynamically corrupted parties as in most results relating symbolic and computational models. Initially corrupted machines simply send their keys on the network.

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

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

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.

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.

Anonymity of group signatures [5] 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 $\mathcal{F}_0 \approx \mathcal{F}_1$.

3. THE APPLIED PI-CALCULUS

We use the applied π-calculus of [2] as a symbolic model. There are several reasons for this choice. First, there are verification tools relying on this model [15]. Next, Though only a small fragment of this process calculus is used in the present paper, we plan several extensions in various directions: considering more primitives (and equational theories), enriching the control structure, e.g. with conditionals and sequential composition,... The applied π-calculus is well suited for such extensions.

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*, the least set containing $\mathcal{N},\mathcal{X}$ and closed by application of a symbol in Σ. We assume that Σ contains a binary pairing function $< u, v >$, the corresponding projections functions π_1, π_2, and a length function l, which is a morphism from $T(\mathcal{N},\mathcal{X})$ to $\mathbb{N}$. We assume infinitely many names of any length. Terms represent messages and names stand for (randomly) generated data. $\alpha, \beta, \dots$ are meta-variables that range over names and variables. We confuse the name generation and the local variables using the same ν construction, as they obey the same scoping/renaming rules. $\overline{u}$ stands for a sequence $u_1, \dots, u_n$. $\sigma = \{x_1 \mapsto s_1, \dots, x_k \mapsto s_k\}$ 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 $\{x_1, \dots, x_k\}$.

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.*

The syntax of processes and extended processes is displayed in Figure 1. In what follows, we restrict ourselves to processes with public channels. $\mathcal{P}$ is a set of predicate symbols with an arity. A difference with [2] is that 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 may want (or not). In [2] the condition is always an equality. Encoding the predicate semantics with equalities is (only) possible when there is no negative condition: it suffices then to express when a predicate is true. We believe that predicates allow to better reflect the ability of the adversary, with less coding. As we will see in the section 4, the predicates will be interpreted as polynomially computable Boolean functions.

Note that we use unbounded (un-guarded) replication of processes. This does not prevent from getting both soundness and completeness w.r.t. the computational interpretation: we show that if there is a computational attack, then there is a symbolic one (soundness). This symbolic attack does not depend on the security parameter: in this respect, it is a constant size attack. Interpreting back the attack in the computational world, this means that there is an attack whose size is independent of the security parameter.

ϕ, ψ `::=`	conditions
$p(s_1, \dots, s_n)$	predicate application
$\phi \wedge \psi$	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

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

3.2 Operational semantics

We briefly recall the operational semantics of the applied pi-calculus (see [2] 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. Predicate symbols are interpreted as relations over $\mathcal{T}(\mathcal{N})/=_E$. This yields a structure $\mathcal{M}$.

Example 3.2. *The equations E_0 corresponding to Σ_0 are* $\mathsf{dec}(\{x\}^z_y, y) = x \qquad \pi_1(< x, y >) = x \qquad \pi_2(< x, y >) = y$

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.*
- *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.*
- $\mathsf{P_{samekey}}$ *is binary and holds on ciphertexts using the same encryption key: $\mathcal{M} \models \mathsf{P_{samekey}}(s,t)$ iff $\exists k, u, v, r, r'. EQ(s, \{u\}^r_k) \wedge EQ(t, \{v\}^{r'}_k)$.*
- *EL is binary and holds on s, t iff $M(s), M(t)$ and s, t have the same length. we assume that there is a length function, which is defined on terms as a homomorphism from terms to natural numbers.*

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

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

Figure 2: Structural equivalence

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:

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

$\xrightarrow{*}$ 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 2. *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 3.3 (Group signature). *Group signature as defined in [5] 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 $\mathtt{if}\ \phi\ \mathtt{then}\ \overline{\mathsf{c}_{\mathsf{out}}}(s) \cdot B\ \mathtt{else}\ \overline{\mathsf{c}_{\mathsf{out}}}(\perp) \cdot \mathbf{0} \quad \in \mathcal{B}(i, \overline{n}, \overline{x})$. Intuitively, if all tests are satisfied, 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
$$\mathsf{c}_{\mathsf{in}}(x).\ \mathtt{if}\ EQ(\pi_1(x), i)\ \mathtt{then}\ B\ \mathtt{else}\ \overline{\mathsf{c}_{\mathsf{out}}}(\perp) \cdot \mathbf{0}$$
$\in \mathcal{B}(i, \overline{n}, \overline{x})$. Intuitively, on input x, the basic process first checks that it is the expected recipient of the message, before processing it.

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

Example 3.4. *The Wide Mouth Frog [16] is a simple protocol where a server transmits a session key from an agent A to an agent B.*

$$\begin{array}{rcl}
A \rightarrow S & : & A, \{N_a, B, K_{ab}\}_{K_{as}} \\
S \rightarrow B & : & \{N_b, A, K_{ab}\}_{K_{bs}}
\end{array}$$

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

$$A(a,b) = \mathit{if}\ \mathsf{true}\ \mathit{then}\ \overline{\mathsf{c}_{\mathsf{out}}}(< a, \{< n_a, < b, k_{ab} >>\}^r_{k_{as}} >) \cdot \mathit{0} \quad \mathit{else}\ \overline{\mathsf{c}_{\mathsf{out}}}(\perp) \cdot \mathit{0}$$

Similarly a session of role S played for agents a, b and whose id is l, can be modeled by

$$\begin{array}{l}
S(a,b,l) = \mathsf{c}_{\mathsf{in}}(x).\ \mathit{if}\ EQ(\pi_1(x), l)\ \mathit{then} \\
\mathit{if}\ \pi_1(\pi_2(x)) = a\ \wedge\ \pi_1(\pi_2(\mathsf{dec}_{k_{as}}(\pi_2(\pi_2(x))))) = b\ \mathit{then} \\
\overline{\mathsf{c}_{\mathsf{out}}}(\{< n_b, < a, \pi_2(\pi_2(\mathsf{dec}_{k_{as}}(\pi_2(\pi_2(x))))) >>\}^r_{k_{bs}}) \cdot \mathit{0} \\
\qquad \mathit{else}\ \overline{\mathsf{c}_{\mathsf{out}}}(\perp) \cdot \mathit{0}\ \mathit{else}\ \overline{\mathsf{c}_{\mathsf{out}}}(\perp) \cdot \mathit{0}
\end{array}$$

A *simple process* combines any number of instances of the protocol roles, hiding names that are meant to be (possibly shared) secrets:

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

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 $!((\nu\overline{y_i}, l_i)\overline{\mathsf{c}_{\mathsf{out}}}(l_i).B'_i)$ 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 [4]).

Example 3.5. *Continuing Example 3.4, 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*

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

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

3.4 Deduction and static equivalence

As in the applied pi calculus [2], 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. $\overline{n}$ stands for fresh names that are *a priori* unknown to the attacker.

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 following rules:

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

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

Example 3.6. *Consider the signature and equational theory of example 3.2. Let $\phi = \nu n_1, n_2, n_3, r_1, r_2, r_3.\sigma$ with $\sigma = \{x_1 \mapsto \{n_1\}_{k_1}^{r_1}, x_2 \mapsto < \{n_2\}_{n_1}^{r_2}, \{n_3\}_{n_2}^{r_3} >\}$. Then $\nu n_1, n_2, n_3, r_1, r_2, r_3.(\sigma \vdash n_3)$.*

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 3 (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 $\mathcal{M} \models p(\underline{s_1}, \ldots, s_k)\sigma$. 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*

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

Example 3.7. *Consider (again for the theory of Example 3.2) 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*

4. COMPUTATIONAL INTERPRETATION

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 an interpretation $\mathcal{M}$ of the predicates symbols in the symbolic model we assume that $[\![p]\!]$ is an *implementation* of this interpretation $p \subseteq (T(\mathcal{N}))^n$, that is

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

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

Example 4.1. *Consider the predicate symbols of Example 3.2. 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 [31]).*
- *$[\![EQ]\!]$ 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.*

Given a random tape τ and a security parameter η, a simple process P is implemented as expected. In particular, we assume that shared names are distributed to the expected machines in an initialization phase and random number are computed according to the random tape. The implementation of P is denoted by $[\![P]\!]_\eta^\tau$.

5. MAIN RESULT

5.1 Assumptions and result

Encryption scheme.

We assume that it is IND-CPA (more precisely "type 3"-secure of [4]) and INT-CTXT, as defined in [13]. Moreover, we assume that each time the adversary needs a new key, it requests it to the protocol (e.g. using a corrupted party). The parties are supposed to check that the keys they are using have been properly generated.

Key hierarchy.

A term u which occurs at least once in t at another position than a key or a random number (third argument in encryption) is called a *plaintext subterm* of t. E.g. k_1 and k_3 occur in plaintext in $< k_1, \{\{k_3\}_{k_2}^{r_2}\}_{k_1}^{r_1} >$ but not k_2. We say that k *encrypts* k' in a set of terms S if S contains a subterm $\{u\}_k^r$ such that k' is a plaintext subterm of u. We assume a *key hierarchy*, i.e. an ordering on private keys such that, for any execution of the protocol no key encrypts a greater key. If there is a key hierarchy, no key cycle can be created. Note that, when comparing two processes, the two key hierarchies do not need to be identical.

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.

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

THEOREM 4. *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 3.3. 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'.

5.2 Overview of the proof

The rest of the paper is devoted to the proof sketch of Theorem 4.

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.

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 the input they get from the environment: T_P is a labeled transition system, whose initial state is P. 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 7) 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 11) 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.

As a (very simple) example, consider the trees T_P and T_Q whose edges are labeled with any possible pair of symbolic messages. The path labeled $< u_1, v_1 >, \ldots, < u_n, v_n >$ yields a node labeled with $\{u_1\}_k^{r_1}, \ldots, \{u_n\}_k^{r_n}$ in T_P and yields a node labeled with $\{v_1\}_k^{r_1}, \ldots, \{v_n\}_k^{r_n}$. This corresponds to a Left-Right oracle of an IND-CPA game. The tree soundness states in this case that the two trees are indistinguishable: even if the attacker adaptatively chooses his requests (i.e. a path in the tree), he cannot make a difference between the two experiments. IND-CCA2 could be also expressed in this way. Here we consider more general experiments, specified by the two processes P, Q.

$T_P \approx T_Q \Rightarrow [\![P]\!] \approx [\![Q]\!]$ (Lemma 13) It uses trace lifting: we need to prove that a computational trace is, with an overwhelming probability, an instance of a symbolic trace.

For instance in the above IND-CPA game, we cheated a little bit since the requests of the attacker were instances of symbolic requests, while in a true IND-CPA game they can be arbitrary bitstrings. This last step shows that it is actually not cheating: it does not make a significant difference (actually it does not make any difference at all in an IND-CPA game).

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

We first define a general notion of trees that could serve to design oracles: the main purpose is to lift static equivalence (of frames) to trees, i.e. in an adaptative setting. Trees defined by the protocols (processes) are special cases, as we will see next. But we use the general definition in further transformations of the oracles.

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. For $i = 1, 2$, let $\phi_i = \nu\overline{n}\sigma_i$ be two frames. We write $t \in \phi_1$ if $t = x\sigma_1$ for some $x \in \mathsf{dom}(\sigma_1)$ and $\phi_1 \subseteq \phi_2$ if $x\sigma_1 = x\sigma_2$ for all $x \in \mathsf{dom}(\sigma_1)$.

Definition 5. *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 ϕ 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.*

Such trees will be used in the next section to represent all possible behaviors of the processes, in a structured way. Since we have unbounded replication, the trees need not to be bounded in depth: there may be infinite paths. They may also be infinitely branching, as, at any time, the possible attacker's actions are unbounded

In this definition, 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 6. *$\sim$ 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

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

where each Q_j^p is either $\mathbf{0}$ or $\mathsf{c_{in}}(x_j^p).P_j^p$. 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

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

in which case

$$T_\mathcal{P}(p \cdot \alpha) = (\nu\overline{n}\nu\overline{x}.Q_1^q\|\sigma_1^q\| \cdots \|Q_N^q\|\sigma_N^q,\\ \nu\overline{n}.\sigma \cup \{x_\alpha \mapsto \alpha\} \cup \textstyle\bigcup_{i=1}^m \{x_{\alpha_i} \mapsto \alpha_i\})$$

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 $\mathsf{c_{in}}(x_i^q).P(x_i^q)$ 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 $S_j =!(\nu\overline{y}\nu l\ \overline{\mathsf{c_{out}}}(l).B)$. Then

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

where z is a fresh variable and assuming by renaming that the names of $\overline{y}$ do not appear free in $Q_1^p\|\sigma_1^p\| \cdots \|Q_N^p\|\sigma_N^p$. Note that by construction, $B \in \mathcal{B}(l, \overline{n_l}, \overline{x_l})$ 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 7. *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 [2] (see the proof in [20]).

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.

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. When T is a process computation tree, 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 $1 \leq j \leq n$.
- 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 8. *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, $|\boldsymbol{Pr}\{\tau : A^{\mathcal{O}_{T_1,\tau}}(0^\eta) = 1\} - \boldsymbol{Pr}\{\tau : A^{\mathcal{O}_{T_2,\tau}}(0^\eta) = 1\}|$ 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 3.2 and in the following examples of Section 3.2.

7.1 Getting rid of encryption

The function Ψ_k on trees replaces terms under encryption by constants 0^l of the same length and is used later for stepwise simplifications:

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

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 is unable to make a difference:

LEMMA 9. *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 $T \sim \Psi_k(T)$.*

The lemma is proved by choosing Ψ_k for the one-to-one function β.

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

LEMMA 10. *Let ϕ_1 and ϕ_2 be two frames such that for any subterm of ϕ_1 or ϕ_2 of the form $\{u\}_k^r$, 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.*

7.2 Soundness of static equivalence on trees

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

LEMMA 11. *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 9. 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 $\{u\}_k^r$ are such that $u = 0^l$ for some l. In this case, we show that equivalence of such frames coincides with equality using Lemma 10.

7.3 Relating concrete and symbolic traces

We need here to show that concrete traces are, with overwhelming probability, interpretations of symbolic ones. We first define formally what it means.

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

Its computation can be represented as $m_1 \cdots m_n$, the sequence of messages sent by the adversary.

If T is a process computation tree and $p \in Pos(T)$, p *fully abstracts* the computation sequence $m_1 \cdots m_n$ if $p = u_1 \cdots u_n$ and, for every $j \leq n$, $[\![u_j]\!]^\tau = m_j$. In other words, p fully abstracts a computation sequence if it defines a symbolic trace whose interpretation is that computation sequence.

LEMMA 12. *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 any computational sequence of $A \| [\![\mathcal{P}]\!]^\tau$ is fully abstracted by some path p in $T_{\mathcal{P}}$, is overwhelming.*

To prove this lemma, we first simplify the trees by applying the functions Ψ_k thanks to Lemmas 11 and 9. Then, we investigate in which cases the adversary may produce traces that cannot be lifted and, in each situation, we break either INT-CTXT or IND-CPA.

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 13. *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 $[\![P_1]\!] \approx [\![P_2]\!]$.*

Theorem 4 is now a straightforward consequence of Lemma 7, 11 and 13.

8. EXTENSIONS

Following our proof scheme, we believe that our results can be extended to other security primitives, e.g. public-key encryption or signatures. We also wish to extend the simple process, for instance addind conditionals and sequential composition.

There are harder extensions. For instance, can we drop the requirement that private keys are not dynamically disclosed? As explained in [8], 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 11.

Acknowledgements

We thank Michael Backes, Steve Kremer, Dominique Unruh and Bogdan Warinschi for their comments on this paper, as well as the anonymous referees.

9. 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 C. Fournet. Mobile values, new names, and secure communication. In *Principles of Programming Languages (POPL'01)*, pages 104–115, 2001.

[3] M. Abadi and J. Jürgens. Formal eavesdropping and its computational interpretation. In *Theoretical Aspects of Computer Software, LNCS 2215*, 2001.

[4] M. Abadi and P. Rogaway. Reconciling two views of cryptography: the computational soundness of formal encryption. *J. Cryptology*, 2007.

[5] 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.

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

[7] M. Backes, M. Dürmuth, and R. Küsters. On simulatability soundness and mapping soundness of symbolic cryptography. In *Proc. of 27th FSTTCS*, volume 4855 of *LNCS*, 2007.

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

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

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

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

[12] M. Baudet, V. Cortier, and S. Kremer. Computationally sound implementations of equational theories against passive adversaries. In *Proc. ICALP'05*, volume 3580 of *LNCS*, 2005.

[13] 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.

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

[15] 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.

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

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

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

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

[20] H. Comon-Lundh and V. Cortier. Computational soundness of observational equivalence. Research Report 6508, INRIA, `https://hal.inria.fr/inria-00274158`, Apr. 2008.

[21] V. Cortier, S. Kremer, R. Küsters, and B. Warinschi. Computationally sound symbolic secrecy in the presence of hash functions. In *Proc. FSTTCS*, volume 4337 of *LNCS*, pages 176–187, 2006.

[22] 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.

[23] S. Delaune, S. Kremer, and M. D. Ryan. Coercion-resistance and receipt-freeness in electronic voting. In *Computer Security Foundations Workshop (CSFW'06)*, pages 28–39, 2006.

[24] 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.

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

[26] 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.

[27] 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.

[28] 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.

[29] 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.

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

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

[32] 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.

[33] 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.