Security proof with dishonest keys*

Hubert Comon-Lundh[1], Véronique Cortier[2], and Guillaume Scerri[1,2]

[1] LSV, ENS Cachan & CNRS & INRIA, France
[2] LORIA, CNRS, France

Abstract. Symbolic and computational models are the two families of models for rigorously analysing security protocols. Symbolic models are abstract but offer a high level of automation while computational models are more precise but security proof can be tedious. Since the seminal work of Abadi and Rogaway, a new direction of research aims at reconciling the two views and many *soundness results* establish that symbolic models are actually sound w.r.t. computational models.
This is however not true for the prominent case of encryption. Indeed, all existing soundness results assume that the adversary only uses honestly generated keys. While this assumption is acceptable in the case of asymmetric encryption, it is clearly unrealistic for symmetric encryption. In this paper, we provide with several examples of attacks that do not show-up in the classical Dolev-Yao model, and that do not break the IND-CPA nor INT-CTXT properties of the encryption scheme.
Our main contribution is to show the first soundness result for symmetric encryption and arbitrary adversaries. We consider arbitrary indistinguishability properties and an unbounded number of sessions.
This result relies on an extension of the symbolic model, while keeping standard security assumptions: IND-CPA and IND-CTXT for the encryption scheme.

1 Introduction

Formal proofs of security aim at increasing our confidence in the security protocols. The first formal proofs/attacks go back to the early eighties (for instance [DY81]). Such proofs require a formal model for the concurrent execution of processes in a hostile environment (for instance [AG99,AF01]). As a consequence, the security proof only proves something about a formal model. That is why we were faced to paradoxical situations, in which a protocol is proved to be secure and later an attack is found. This is the case for the Bull authentication protocol [RS98], or the Needham-Schroeder-Lowe protocol [War03,BHO09]. In such cases, the proof is simply carried in a model that differs from the model in which the attack is mounted. There are much more examples, since the security proofs always assume some restrictions on the attacker's capabilities, ruling out

* The research leading to these results has received funding from the European Research Council under the European Union's Seventh Framework Programme (FP7/2007-2013) / ERC grant agreement no 258865, project ProSecure.

some side-channel attacks. The examples show however that we need to specify as precisely as possible the scope of the proofs, i.e., the security assumptions.

This is one of the main goals of the work that started ten years ago on computational soundness [AR00,BPW03]: what is the scope of formal proofs in a Dolev-Yao style model ? This is important, because the automatic (or checkable) proofs are much easier in a Dolev-Yao, called *symbolic* hereafter, model, in which messages are abstract terms and the attacker is any formal process that can intercept and send new messages that can be forged from the available ones.

Numerous results have been obtained in this direction, but we will only focus on the case of symmetric encryption. If we assume only two primitives: symmetric encryption and pairing, to what extent is the symbolic model accounting for attacks performed by a probabilistic polynomial time attacker ? The first result [AR00] investigates the case of a passive attacker, who cannot send fake messages, but only observes the messages in transit. Its goal is to distinguish between two message sequences, finding a test that yields 1 on a sequence and yields 0 on the other sequence (for a significant subset of the sample space). The authors show for instance that, if the encryption scheme is IND-CPA, is "which key-preserving" (two encrypted messages with the same key are indistinguishable from two messages encrypted with different keys) and hides the length, then the symbolic indistinguishability implies the computational indistinguishability. In short, in that case, the symbolic model accounts for the probabilistic polynomial time attacks on the implementations of the messages.

To our knowledge, only two further works extend this result: first M. Backes *et al* in [BP04] and two of us in [CLC08a]. Both works try to consider an active attacker, thus allowing an interaction of the attacker with the protocol. Both works require additional assumptions: INT-CTXT for the encryption scheme, no dynamic corruption of keys, no key cycles,... The main difference between the two results lies in the security properties that are considered: while [BP04] considers trace properties, [CLC08a] considers equivalence properties. Therefore the proof methods are quite different.

We wish however to insist on another issue: in [CLC08a], the encryption keys are assumed to be authentic. In other words, if the attacker forges a key, then this key must be generated using the key generation algorithm. This is a strong assumption, that is hard to ensure in practice. For a public key cryptosystem, we can imagine that the public keys are certified by a key-issuing authority and that this authority is trusted. But in the case of symmetric encryption, there are many examples in which a participant generates himself a session key. This limitation and its consequences are discussed at length in [CC11].

Concerning [BP04], the case of dishonest keys is not mentioned explicitly, while the proof assumes that there is no such key: the paper implicitly assumes that all keys are generated using the key generation algorithm.

On the other hand, the problem of dishonest keys is important: the cryptographic assumptions, such as IND-CPA, INT-CTXT, IND-CCA,... rely on a sampling of the keys. This does not say anything on any particular key: there could be a key for which all the properties fail and such a key could be chosen

by the attacker. As we show in section 2, there are many situations in which we can mount an attack, even when the encryption scheme has all the desired properties.

The main source of examples of formal security proofs of protocols using symmetric key encryption (not assuming that keys are always honest) is CryptoVerif [Bla08]. These proofs show, as an intermediate step, that the keys used for encryption by a honest agent for a honest agent are honestly generated. In this way, the security properties of the encryption scheme are only applied to ciphertexts using a randomly generated key. This works for many protocols, but cannot work for a soundness result since there are protocols that are secure, while at some point a honest agent may use a fake key (sent by the attacker) for encrypting a message sent to a honest participant.

In this paper, we propose a solution to the dishonest keys problem, adding capabilities to the symbolic attacker. We try to capture the ability to forge a key, that has an arbitrary behavior (choosen by the attacker), *on messages that have been sent so far*. Roughly, the attacker may forge a particular key k, such that given any pair of known messages (m_1, m_2), the encryption (resp. decryption) of m_1 with k yields m_2. As we show in an example in section 2, the attacker must also get any encryption/decryption of a message that uses a fake key.

This model is formalized in section 3, building on the applied π-calculus of [AF01]. We then show in section 5 that this model is computationally sound, without assuming of course that keys are honestly generated. More precisely, we prove, in the case of simple processes, that, if two processes P, Q are observationally equivalent, then their implementations $[\![P]\!], [\![Q]\!]$ are computationally indistinguishable, provided that the encryption scheme is IND-CPA and INT-CTXT. In other words, as in [CLC08a] we (also) cover equivalence properties. This soundness proof is similar to the proof of [CLC08a]: we prove a tree soundness result and a trace mapping. There are some significant technical differences, that will be pointed out. Also, a gap in the final proof of [CLC08a] is fixed here, considering a new indistinguishability game.

Finally, we show that our soundness result does not give too much power to the symbolic attacker: we give a computationally secure process in our extended model, in which the attacker may send fake keys that are then used for decryption.

2 Motivation : some examples of insufficiency of current models

Standard cryptographic assumptions do not provide any guarantee for keys that are not generated using the key generation algorithm. In particular, the IND-CPA and IND-CTXT properties do not exclude the case where some keys have particular properties. We provide below several examples of protocols whose security is broken due to the behavior of dishonestly generated keys. For the sake of clarity, we provide with an informal specification of the protocols and we consider attacks that consist of some agent reaching an undesired **bad** state.

These examples could be easily turned into examples with confidentiality or authenticity properties. For simplicity, we also omit the randomness used for encryption.

A first fact about dishonest keys is that decrypting honest cyphertexts with dishonest keys does not necessary fail and may, on the contrary, result into plaintext that can be exploited by an attacker.

Example 1. Assume k_{AB} is a secret key shared between A and B.

$$A \to \langle c, \{c\}_{k_{AB}} \rangle \qquad \begin{array}{l} B \leftarrow \langle z, \{\{b\}_z\}_{k_{AB}} \rangle \\ B \to \textbf{bad} \end{array}$$

B waits for a key z and a message looking like $\{\{b\}_z\}_{k_{AB}}$ and goes in a bad state. For all usual formal models, B can not reach the bad sate. On the other hand, it is computationally feasible for an adversary to forge a key k such that $\mathcal{D}(c,k) = b$ ($\mathcal{D}(c,k)$ is the decryption of c with k), in that case B goes in the bad state receiving $\langle k, \{c\}_{k_{AB}} \rangle$.

This example can easily be generalized to the case where the decryption of several ciphertexts with some dishonest key yields exploitable results.

Example 2. Assume k_{AB} is a secret key shared between A and B.

$$A \to \langle \langle c, \{c\}_{k_{AB}} \rangle, \langle d, \{d\}_{k_{AB}} \rangle \rangle \qquad \begin{array}{l} B \leftarrow \langle k, \langle \{\{b\}_k\}_{k_{AB}}, \{\{b'\}_k\}_{k_{AB}} \rangle \rangle \\ B \to \textbf{bad} \end{array}$$

The standard cryptographic assumptions do not prevent the adversary from forging a key k such that $\mathcal{D}(c,k) = b$ and $\mathcal{D}(d,k) = b'$ simultaneously.

The two previous examples seem to rely on the fact that the adversary knows the (honest) cyphertexts that are decrypted. This is actually not needed to mount attacks.

Example 3. Assume k_{AB} is a secret key shared between A and B and s be a secret known only to A.

$$A \to \{s\}_{k_{AB}} \qquad \begin{array}{l} B \leftarrow \langle k, \{\{b\}_k\}_{k_{AB}} \rangle \\ B \to \textbf{bad} \end{array}$$

In the computational setting the adversary could forge a key k such that, if s is randomly chosen , $\mathcal{D}(s,k) = b$ with a non negligible probability. Receiving $k, \{s\}_{k_{AB}}$, B would reach the bad state with non negligible probability.

Another important behavior of dishonest keys is the fact that attempting to decrypt a message with a dishonest key may actually reveal the message.

Example 4. Consider the following protocol where s is secret.

$$A \to \{s\}_{k_{AB}} \qquad \begin{array}{l} B \leftarrow \langle k, \{\{z\}_k\}_{k_{AB}} \rangle \\ B \to \textbf{ok} \end{array}$$

The agent B simply tries to decrypt the message received under k_{AB} and outputs **ok** if he succeeds. In any usual formal model, s stays secret.

Let us consider a key k_i such that k_i decrypts s if and only if the i-th bit of s is with a 0. Sending $k_i, \{s\}_{k_{AB}}$ to a copy of B the adversary learns the $i-th$ bit of s and is then able to learn s entirely.

The previous examples exhibit problematic behaviors when decrypting with a dishonest key. Similar issues occur when encrypting with a dishonest key. The next example shows that the adversary may use dishonest keys to build equalities between cyphertexts.

Example 5. Assume k_{AB} is a secret key shared by A and B.

$$\begin{array}{ll} A \to \{a\}_{k_{AB}} & B \leftarrow \{x, x\}_{k_{AB}} \\ A \leftarrow k & B \to \textbf{bad} \\ A \to \{\{s\}_k, \{a\}_{k_{AB}}\}_{k_{AB}} & \end{array}$$

As previously, nothing prevents the adversary from building a key k such that for a random s, $\{s\}_k^r = \{a\}_{k_{AB}}$ with non negligible probability. Using that key, it is possible to drive B in the bad state.

More generally, $\{x\}_k$ may be an arbitrary function of (x, r) and the previous knowledge of the adversary.

Example 6. Assume k_{AB} is a secret key shared by A and B, s is a secret nonce known to A and s' is a nonce (not necessarily secret).

$$\begin{array}{ll} A \leftarrow k & B \leftarrow \langle k', \{\{\langle s, s\rangle\}_{k'}\}_{k_{AB}}\rangle \\ A \to \{\{\langle s, s'\rangle\}_k\}_{k_{AB}} & B \to \textbf{bad} \end{array}$$

The adversary could forge k, k' such that $\mathcal{D}(\{\langle x, y\rangle\}_k^r, k') = \langle x, x\rangle$ (when x and y are of equal length) which allows B to go in the bad state.

One could think that the collisions induced by a dishonest key k are determined by the message under encryption/decryption and the knowledge of the adversary *at the moment* he forged k. The last example shows that it is actually not the case.

Example 7. Assume that k_{AB} is a secret ket shared by A and B and that s is initially secret.

$$\begin{array}{ll} A \leftarrow \langle k_0, k_1, k_2\rangle & \\ A \to \langle\{k_0\}_{k_{AB}}, \{k_1\}_{k_{AB}}, \{k_2\}_{k_{AB}} & \\ A \to \{\langle A, A\rangle\}_{k_{AB}} & B \leftarrow \langle\{k\}_{k_{AB}}, \{t\}_{k_{AB}}\rangle \\ A \to s & B \to \{\{t\}_k\}_{k_{AB}} \\ A \leftarrow \{\langle x, s\rangle\}_{k_{AB}} & \\ A \to \textbf{bad} & \end{array}$$

Running B an arbitrary (polynomial) number of times yields a computational attack. Consider the three following keys :

– k_0 such that $\{\langle i, n\rangle\}_{k_0} = \langle i+1, n\rangle$
– k_1 such that $\{\langle i, n\rangle\}_{k_1} = \langle i-1, n\rangle$
– k_2 such that $\{\langle i, n\rangle\}_{k_2} = \langle i, n'\rangle$ where n' is the same bitstring as n apart from the i-th bit which is flipped.

With these keys, we can use role B to transform any cyphertext $\{\langle m_1, m_2\rangle\}_{k_{AB}}$ into $\{\langle x, s\rangle\}_{k_{AB}}$.

Let us summarize the lessons provided by these examples. Clearly, existing symbolic models are unsound. Examples 1, 2, and 3 show that we need to let the adversary adds equations when decrypting or encrypting with a dishonest key. Examples 5 and 6 show that these equations may depend on the knowledge of the attacker when he add them. Example 4 demonstrates that any message under decryption/encryption with a dishonest key should be added to the adversary knowledge. Example 7 shows that we have to delay the commitment on the properties of the dishonest key until the state, at which the encryption/decryption with that key is used.

Let us note that in some examples we try to decrypt a honest nonce which should be forbidden by tagging but it is easy to patch these examples by replacing the honest nonces by honest encryptions.

3 Model

Our model is an adaptation of the applied pi-calculus [AF01], enriched with a syntax that allows the attacker to create new equalities between terms, corresponding to equalities permitted by the IND-CCA and the IND-CTXT properties, as illustrated in the previous section.

3.1 Syntax and deduction

Messages are represented by terms build upon a set $\mathcal{V}$ of variables, a set $\mathcal{N}ames$ of names and the signature $\mathcal{F} = \{\{\_\}_\_, \langle\_, \_\rangle, \mathrm{dec}(\_, \_), \pi_1(\_), \pi_2(\_)\}$. As usual, the term $\{s\}_k^r$ represents the encryption of s with the key k and the randomness r, $\langle u, v\rangle$ represents the concatenation of u and v, while $\mathrm{dec}(\_, \_)$, $\pi_1(\_)$, $\pi_2(\_)$ represent respectively the decryption and the left and right projections of a concatenation. We may write $\langle x, y, z\rangle$ for $\langle\langle x, y\rangle, z\rangle$. The set of ground terms (i.e. terms without variables) is denoted by $T(\mathcal{F})$. We divide the set $\mathcal{N}ames$ into three (infinite) subsets: $\mathcal{K}_1$ for honest keys, $\mathcal{K}_2$ for dishonest keys, and $\mathcal{N}$ for the nonces. The set $\mathcal{V}$ is divided into $\mathcal{V}_1 = \{x_1, x_2, \cdots\}$ and $\mathcal{V}_2 = \{y_1, y_2, \cdots\}$ that will be respectively used to store terms and equations.

We assume given a length function $l : T(\mathcal{F}) \mapsto H$ from ground terms to a set H that measures the symbolic length of a term. An example of a length function will be given in the Section 4.2.

We write $\{x_1 \mapsto u_1, \ldots, x_n \mapsto u_n\}$ for the substitution that maps x_i to u_i. The substitution is *ground* when every term u_i is ground. The application of a substitution σ to a term u is denoted $u\sigma$. The signature $\mathcal{F}$ is equipped

with an equational theory, that is closed under application of function symbols, substitution of terms for variables. We write $M =_E N$ when the equation $M = N$ is in the theory E. We may omit the subscript E when it is clear from the context. In this paper, we will consider in particular the theory E_0 defined by the following (infinite, yet recursive) set of equations.

$$\mathrm{dec}(\{x\}_k^z, k) = x \quad \text{for } k \in \mathcal{K}_1 \cup \mathcal{K}_2 \qquad \pi_1(\langle x, y\rangle) = x \qquad \pi_2(\langle x, y\rangle) = y$$

E_0 will then be enriched by the equalities created by the adversary.

The current knowledge of an adversary is represented by a *frame* $\phi = \nu\bar{n} \cdot \sigma$ where σ is a ground substitution that represents the messages accessible to the adversary while $\bar{n}$ denotes the private names (that the adversary does not know initially). From its knowledge ϕ, an attacker can then deduce any term that it can build from the terms in σ and applying function symbols and public names.

Definition 1 (deductibility). *A ground term s is deducible from $\phi = \nu\bar{n} \cdot \sigma$ and an equation set E (we write $\phi \vdash_E s$) if there exists a public term (i.e. not containing names from $\bar{n}$) R such that $R\sigma =_E s$.*

Example 8. Let $\phi = \nu n_1, n_2, n_3, r_1, r_2, r_3 \cdot \sigma$ with $\sigma = \{x_1 \mapsto \{n_1\}_{k_1}^{r_1}, x_2 \mapsto \langle\{n_2\}_{n_1}^{r_2}, \{n_3\}_{n_2}^{r_3}\rangle\}$. Then $\phi \vdash_{E_0} n_3$. The corresponding public term is : $R = \mathrm{dec}(\pi_2(x_2), \mathrm{dec}(\pi_1(x_2), \mathrm{dec}(x_1, k_1)))$

As in [CLC08b], we first extend the applied pi-calculus with predicates that represent the tests that an adversary can perform. We consider four predicates: $\mathrm{M}(u)$ states whether a term u is valid (i.e. will have a computational interpretation); $\mathrm{EQ}(u, v)$ checks whether two terms are equal; $\mathrm{P}_{\mathrm{samekey}}(u, v)$ checks whether u and v are two cyphertexts using the same key; and $\mathrm{EL}(u, v)$ checks whether two terms have the same length. A formula, as defined in Figure 1, is a Boolean combination of these atomic formulas.

The processes are then defined as usual (in Figure 1) with the addition of two new constructors (*eq* and *neq*) that allow to generate new equalities or disequalities between terms. These constructions may appear in attackers processes, but not in the protocols.

The behaviour of a process depends on the equational theory. We therefore consider *localized process* E, X_w, X_c, P where E is a set of ground equations and disequations, that have already been added by the adversary, X_w and X_c are sets of variables and P is a process. The adversary will be allowed to add equations in E "on-the-fly" depending on what he learns. More precisely, when we need to evaluate a test, that involves dishonest keys, the attackers enters a "ADD" mode in which he has to commit to the (non)-validity of equalities containing such keys. In this mode, the frame of P records, using the variables in X_w, the equalities that need a commitment. It also records, using the variables X_c, the equalities on which he committed since he entered the "ADD" mode. When leaving the mode, committed (dis)-equalities have been flushed in E.

$\Phi_1, \Phi_2 ::=$	Formula
$\mathrm{EQ}(s,t), \mathrm{M}(s), \mathrm{P}_{\mathrm{samekey}}(s,t), \mathrm{EL}(s,t)$	predicate application
$\Phi_1 \wedge \Phi_2$	conjunction
$\Phi_1 \vee \Phi_2$	disjunction
$\neg\Phi_1$	negation
$P, Q, R ::=$	Processes
$c(x) \cdot P, \bar{c}(s) \cdot P$	input, output on channel c
$eq(s,t) \cdot P, neq(s,t) \cdot P$	equation, disequation
$\mathbf{0}$	null process
$P \| Q$	parallel composition
$!P$	replication
$(\nu\alpha)P$	restriction
$\texttt{if } \Phi \texttt{ then } P \texttt{ else } Q$	conditional
$A, B ::=$	Extended processes
P	process
$A \| B$	parallel composition
$(\nu\alpha)A$	restriction
$\{x \mapsto s\}$	substitution

For simplicity reasons we will often write $\texttt{if } \Phi \texttt{ then } P \texttt{ else } \overline{c_{out}}(\bot)$ as $[\Phi]P$.

Fig. 1. Syntax of Formula and Processes.

Example 9. Let us consider the protocols of the Examples 4 and 5. For the sake of conciseness, we do not describe the role of A. We instead directly enrich the initial frame with the message emitted by A. We also make use of a pattern-matching notation like in ProVerif [Bla05]. For example, $c(\langle a, \{y\}_{k_{ab}}\rangle).\bar{c}(y)$ denotes the process $c(x).[(\pi_1(x) = a) \wedge M(\mathrm{dec}(\pi_2(x), k_{ab}))].\bar{c}(\mathrm{dec}(\pi_2(x), k_{ab}))$.

The process modeling the protocol described in Example 4 is:

$$P_4 = (\nu k, r, k_{AB})\{x \mapsto \{s\}^r_{k_{AB}}\} \| !c_{in}(\langle z_1, \{z_2\}_{k_{AB}}\rangle).[\mathrm{M}(\mathrm{dec}(z_2, z_1))]\overline{c_{out}}(\mathbf{ok})$$

where $s = \{n\}^r_k$. Similarly, the process modeling Example 5 is:

$$P_5 = (\nu k, r, r_1, r_2, r_3, k_{AB})\{x_1 \mapsto \{a\}^{r_2}_{k_{AB}}, x_2 \mapsto k, x_3 \mapsto \{\{s\}^{r_1}_k, \{a\}^{r_3}_{k_{AB}}\}^{r_2}_{k_{AB}}\} \\ \| c_{in}(\{z_1, z_2\}_{k_{AB}}).[\mathrm{EQ}(z_1, z_2)]\overline{c_{out}}(\mathbf{bad})$$

where $s = \{n\}^r_k$.

3.2 Operational Semantics

Our operational semantics is inspired by the applied π-calculus. For localized processes of the form E, X_w, X_c, A, terms are interpreted in $\mathcal{T}/E \cup E_0$. $E \cup E_0$ is completed into a convergent rewriting system, that minimizes the number of destructors in a term. $t{\downarrow_E}$ will denote the normal form of the term t w.r.t. such

$$
\begin{aligned}
E, X_w, X_c, A\|\mathbf{0} &\equiv E, X_w, X_c, A \\
E, X_w, X_c, A\|B &\equiv E, X_w, X_c, B\|A \\
E, X_w, X_c, (A\|B)\|C &\equiv E, X_w, X_c, A\|(B\|C) \\
E, X_w, X_c, (\nu\alpha)(\nu\beta)A &\equiv E, X_w, X_c, (\nu\beta)(\nu\alpha)A \\
E, X_w, X_c, (\nu\alpha)(A\|B) &\equiv E, X_w, X_c, A\|(\nu\alpha)B \qquad \text{if } \alpha \notin \mathrm{fn}(A) \cup \mathrm{fv}(A) \\
E, X_w, X_c, (\nu x)\{x \mapsto s\} &\equiv E, X_w, X_c, \mathbf{0} \\
E, X_w, X_c, (\nu\alpha)\mathbf{0} &\equiv E, X_w, X_c, \mathbf{0} \\
E, X_w, X_c, !P &\equiv E, X_w, X_c, P\|!P \\
E, X_w, X_c, \{x \mapsto s\}\|A &\equiv E, X_w, X_c, \{x \mapsto s\}\|A\{x \mapsto s\} \\
E, X_w, X_c, \{x \mapsto s\} &\equiv E, X_w, X_c, \{x \mapsto t\} \qquad \text{if } s =_E t \\
E, \emptyset, X_c, P &\equiv E, \emptyset, \emptyset, P
\end{aligned}
$$

Fig. 2. Structural equivalence.

a rewrite system. More generally, in what follows, when we refer to E, we will implicitly assume $E \cup E_0$.

Structural equivalence is very similar to applied π-calculus and is defined in Figure 2.

We first define the semantics of the four predicates as follows.

- $E \vDash \mathrm{M}(s)$ if, for all subterms t of s, $t{\downarrow_E}$ does not contain destructors or variables and uses only keys in key position.
- $E \vDash \mathrm{EQ}(s,t)$ if $E \vDash \mathrm{M}(s) \wedge \mathrm{M}(t)$ and $s{\downarrow_E} = t{\downarrow_E}$.
- $E \vDash \mathrm{P}_{\mathrm{samekey}}(s,t)$ if $E \vDash \mathrm{M}(s) \wedge \mathrm{M}(t)$ and $\exists k, u, v, r, r'$ such that $E \vDash \mathrm{EQ}(s, \{u\}_k^r) \wedge \mathrm{EQ}(t, \{v\}_k^{r'})$
- $E \vDash \mathrm{EL}(s,t)$ if $E \vDash \mathrm{M}(s) \wedge \mathrm{M}(t)$ and $l(s) = l(t)$.

The semantics of formulas is then defined as expected.

We are now ready to define how an attacker can add new equalities between terms. A first condition is that equalities should be *well-formed* in the sense that they should not contradict previously added equalities and they should involve either dishonest encryption or dishonest decryption.

Definition 2. *Let s and t be two ground terms such that $l(s) = l(t)$ and t is without destructors. An equation $s = t$ is well formed with respect to an equation set E, a set of expected equations Y and a frame ϕ (written $\mathrm{wf}^E_{Y,\phi}(s = t)$) if*

- *$E \nvdash (s = t)$*
- *if $(v \neq w) \in E$, $E \cup \{s = t\} \nvdash v = w$*
- *$E \cup \{s = t\} \nvdash n = n'$ with n, n' names and $n \neq n'$*
- *$E \cup \{s = t\} \nvdash \{u\}_k^r = \{u'\}_{k'}^{r'}$ with $k, k' \in \mathcal{K}_1$ and $k \neq k'$ or $r \neq r'$*
- *$E \cup \{s = t\} \nvdash u = v$ when u is a pair and v is not a pair or when u is a ciphertext and v is a private name.*

and the equation satisfies of one of the two following sets of conditions:

1. *$s = \{u\}_k^r$ with $k \in \mathcal{K}_2$ and $\langle u, k, r, t, \mathrm{enc}\rangle \in Y$*
2. *$\phi, u \vdash t$*

3. u is in normal form for E and without destructors

i $s = \mathrm{dec}(u, k)$ *with* $k \in \mathcal{K}_2$ *and* $\langle u, k\,\mathrm{dec}\rangle \in Y$ *and* $(\mathrm{dec}(u, k) = *) \notin E$
ii $\phi, u \vdash t$ *or* $t = \bot$
iii u *is in normal form for* E*, without destructors, and* u *is either a public nonce or an encryption.*

Similarly, a disequation $(s \neq t)$ *is well formed, denoted* $\mathrm{wf}^E_{X,\phi}(s \neq t)$ *if* s *is also without destructors and*

- $s = \{u\}^r_k$ *with* $k \in \mathcal{K}_2$ *and* $\langle u, k, r, t, \mathrm{enc}\rangle \in X$
- $E \cup E_0 \not\models s = t$

We define $\mathrm{wf}^E_\phi(e)$ *to hold if there exists* X *such that* $\mathrm{wf}^E_{X,\phi}(e)$ *holds.*

Intuitively, an adversary can add an equation of the form $\{u\}^r_k = t$ or $\mathrm{dec}(u, k) = t$ only if t is deducible from ϕ, u since dishonest encryption and decryption must be function of the current knowledge ϕ and their input u.

After receiving a message, an agent typically checks the validity of some condition. This test may pass or fail, depending on the value of dishonest encryptions and decryptions performed during the test. As illustrated in Example 4, this may provide the adversary with an additional knowledge, which we define now:

Definition 3. *Let* E *be a set of ground equations,* φ *and* X *be two frames, and* Φ *be a formula. The* additional knowledge *induced by the condition* Φ *w.r.t.* E *and* X*, writen* $K^E_{X,\varphi}(\Phi)$ *is the union of the two following sets :*
the set of all $\langle s, k, \mathrm{dec}\rangle$ *s.t.*

- *There exists a literal* $M(u)$ *in* Φ *such that* $\mathrm{dec}(s, t) \in St(u)$ *with* $E \vDash \mathrm{M}(s)$, $t{\downarrow_E} = k$ *and* $k \in \mathcal{K}_2$.
- $E \not\models \mathrm{M}(\mathrm{dec}(s, k))$ *(to ensure that the condition is not trivially true, in which case the adversary does not learn anything)*
- $\forall y' \in \mathcal{V}_2, \forall s' =_E s, \{y' \mapsto \langle s', k, \mathrm{dec}\rangle\} \notin X$ *(avoiding redundancy)*

and the set of all $\langle s, k, r, v, \mathrm{enc}\rangle$ *s.t.*

- *there exists a literal* $\mathrm{EQ}(t, u)$ *in* Φ *such that* $E \vDash \mathrm{M}(t) \wedge \mathrm{M}(u)$ *and* $t{\downarrow_E} = C[t_1, \cdots, t_n], u{\downarrow_E} = C[u_1, \cdots, u_n]$
- *for all* $i \in \{1, \cdots, n\}$ *there exist* s_i *and* $k_i \in \mathcal{K}_2$ *such that*
  - *either* $t_i = \{s_i\}^{r_i}_{k_i}$ *and* $\mathrm{wf}^E_\varphi(t_i = u_i)$*. In that case we let* $v_i = u_i$.
  - *or* $u_i = \{s_i\}^{r_i}_{k_i}$ *and* $\mathrm{wf}^E_\varphi(u_i = t_i)$*. In that case we let* $v_i = t_i$.
- $\exists i \in \{1 \cdots n\}$ *such that* $s_i = s, k_i = k, r_i = r, v_i = v$ *(we chose a pair of terms)*
- $\forall y' \in \mathcal{V}_2, \{y' \mapsto \langle s, k, r, v, \mathrm{enc}\rangle\} \notin X$ *(to avoid redundancy).*

Example 10. Back to Example 4 , $K^\emptyset_{\emptyset,(\nu s,r,k_{AB})\{x\mapsto\{s\}^r_{k_{AB}}\}}(\mathrm{M}(\mathrm{dec}(s, k)))$. Indeed, the only literal in the condition is $\mathrm{M}(\mathrm{dec}(s, k))$, and the knowledge set is empty, therefore $K^\emptyset_{\emptyset,(\nu s,r,k_{AB})\{x\mapsto\{s\}^r_{k_{AB}}\}}(\mathrm{M}(\mathrm{dec}(s, k))) = \langle s, k, \mathrm{dec}\rangle$.

$$\frac{\tilde{y} \text{ are the next } \#K^E_{\phi(P)_{|X_w},\phi(P)\setminus(X_w\cup X_c)}(t_\Phi(P)) \text{ free variables in } \mathcal{V}_2}{E, X_w, X_c, P \xrightarrow{\varepsilon} E, X_w \cup \{\tilde{y}\}, X_c, P\|\{\tilde{y} \mapsto K^E_{X_w}(Pt_\Phi(P))\}} \text{ R-Add}$$

$$\frac{\mathrm{wf}^E_{\phi(z),\phi\setminus(X_w\cup X_c)}(s = t) \qquad z \in X_w}{E, X_w, X_c, eq(s,t).P\|\phi \xrightarrow{\tau} E \cup \{s = t\}, X_w\setminus z, X_c \cup \{z\}, P\|\phi} \text{ R-Eq}$$

$$\frac{\mathrm{wf}^E_{\phi(z),\phi\setminus(X_w\cup X_c)}(s \neq t) \qquad z \in X_w}{E, X_w, X_c, neq(s,t).P\|\phi \xrightarrow{\tau} E \cup \{s \neq t\}, X_w\setminus z, X_c \cup \{z\}, P\|\phi} \text{ R-Neq}$$

$$\frac{}{E, \emptyset, \emptyset, c(x).P\|\bar{c}(t).Q \xrightarrow{\tau} E, \emptyset, \emptyset, P\|Q\|\{x \mapsto t\}} \text{ R-Com}$$

$$\frac{E \cup E_0 \vDash \Phi}{E, \emptyset, \emptyset, \texttt{if } \Phi \texttt{ then } P \texttt{ else } Q \xrightarrow{\tau} E, \emptyset, \emptyset, P} \text{ R-Cond1}$$

$$\frac{E \cup E_0 \nvDash \Phi}{E, \emptyset, \emptyset, \texttt{if } \Phi \texttt{ then } P \texttt{ else } Q \xrightarrow{\tau} E, \emptyset, \emptyset, Q} \text{ R-Cond2}$$

$\phi(P)$ denotes the maximal frame which can be extracted from process P. If $X = \{x_1, \cdots, x_n\}$ is a set of terms (ordered), then $\{\tilde{y} \mapsto X\}$ denotes the frame $\{y_1 \mapsto x_1, \cdots, y_n \mapsto x_n\}$. $\phi\setminus X$ stands for $\phi_{|\mathcal{V}\setminus X}$. $t_\Phi(P)$ is set of conditions that occurs in head in P, that is $t_\Phi(P) = \{\Phi_1, \ldots, \Phi_n\}$ if $P \equiv \nu\bar{n}[\Phi_1]P_1\| \cdots \|[\Phi_n]P_n\|Q$ where n is maximal.

Fig. 3. Reduction semantics.

The reduction semantics is defined in Figure 3. The rules R-Com, R-Cond1, R-Cond2 are the standard communication and conditional rules. Note that these rules require the sets X_w and X_c to be empty. The validity of a condition Φ may depend on the behavior of dishonest encryption/decryption performed when evaluating the condition. The R-Add rule adds to the frame the knowledge induced by the conditions that are about to be evaluated, making it available to the attacker. Simultaneously, R-Add adds in X_w the variables referring to all the equations that need to be decided before evaluating the conditions. It is then necessary to apply the rules R-Eq and R-Neq until X_w is empty, in order to decide whether each possible equality involving a dishonest encryption/decryption should be set at true or false.

The R-Add rule should be applied before evaluating a condition (i.e. before applying R-Cond1, R-Cond2). Therefore, we define $\rightarrow^*$ as the smallest transitive relation containing $\equiv$, $(\xrightarrow{\tau}\xrightarrow{\varepsilon})$ and closed by application of contexts.

We will write, if $t\#\tilde{n} : P \xrightarrow{c(t)} Q$ if $P \rightarrow^* E, X_w, X_c, (\nu\tilde{n})c(x).P'\|Q'$, and $E, X_w, X_c, (\nu\tilde{n})P'\|Q'\|\{x \mapsto t\} \xrightarrow{\varepsilon}\rightarrow^* Q$

We also write, if $t\#\tilde{n} : P \xrightarrow{\bar{c}(t)} Q$ if $P \rightarrow^* E, X_w, X_c, (\nu\tilde{n})\bar{c}(t).P'\|Q'$, and $E, X_w, X_c, (\nu\tilde{n})P'\|Q'\|\{x \mapsto t\} \xrightarrow{\varepsilon}\rightarrow^* Q$

We also write $E, X_w, X_c, P \xrightarrow{(n)eq(s,t)} E \cup \{s = t\}, X'_w, X'_c, Q$ if we have $E, X_w, X_c, P \| (n)eq(s,t) \rightarrow^* E \cup \{s(\neq) = t\}, X'_w, X'_c, Q$

3.3 Examples

We show how the computational attacks described in Section 2 are now reflected in our symbolic model.

For attacking the process P_4, we consider the following (symbolic) adversary:

$$A_4 = \overline{c_{in}}(\langle k, x\rangle).eq(\mathrm{dec}(\pi_2(y_1)), n).\overline{c}(\pi_2(y_1))$$

With rule R-Com and some structural congruences, $\emptyset, \emptyset, \emptyset, A_4 \| P_4$ reduces to $\emptyset, \emptyset, \emptyset, Q_1$ where Q_1 is:

$$(\nu s, r, k_{AB})\{x \mapsto \{s\}^r_{k_{AB}}\} \| [\mathrm{M}(\mathrm{dec}(s,k))]\overline{c_{out}}(\mathbf{ok}) \\ \| eq(\mathrm{dec}(\pi_2(y_1)), n).\overline{c}(\pi_2(y_1)) \| \{z \mapsto \langle k, \{s\}^r_{k_{AB}}\rangle\}$$

As explained in Example 10, $K^{\emptyset}_{\emptyset,(\nu s,r,k_{AB})\{x\mapsto\{s\}^r_{k_{AB}}\}}(\mathrm{M}(\mathrm{dec}(s,k))) = \langle s, k, \mathrm{dec}\rangle$. Applying R-Add we get that $\emptyset, \emptyset, \emptyset, Q_1$ reduces to $\emptyset, \{y_1\}, \emptyset, Q_2$ where Q_2 is:

$$(\nu s, r, k_{AB})\{x \mapsto \{s\}^r_{k_{AB}}\} \| [\mathrm{M}(\mathrm{dec}(s,k))]\overline{c_{out}}(\mathbf{ok}) \\ \| eq(\mathrm{dec}(\pi_2(y_1)), n).\overline{c}(\pi_2(y_1)) \| \{z \mapsto \langle k, \{s\}^r_{k_{AB}}\rangle\} \| \{y_1 \mapsto \langle \mathrm{dec}, s, k\rangle\}$$

With rule R-Eq and some structural congruences, as $\mathrm{dec}(s,k) = n$ is well formed, we obtain $\{\mathrm{dec}(s,k) = n\}, \emptyset, \{y_1\}, Q_3$ where Q_3 is:

$$(\nu s, r, k_{AB})\{x \mapsto \{s\}^r_{k_{AB}}\} \| [\mathrm{M}(\mathrm{dec}(s,k))]\overline{c_{out}}(\mathbf{ok}) \\ \| \overline{c}(s) \| \{z \mapsto \langle k, \{s\}^r_{k_{AB}}\rangle\} \| \{y_1 \mapsto \langle \mathrm{dec}, s, k\rangle\}$$

With rule R-Cond1 and some structural equivalence, we have $\{\mathrm{dec}(s,k) = n\}, \emptyset, \emptyset, Q_4$ where Q_4 is :

$$(\nu s, r, k_{AB})\{x \mapsto \{s\}^r_{k_{AB}}\} \| \{z \mapsto \langle k, \{s\}^r_{k_{AB}}\rangle\} \| \{y_1 \mapsto \langle \mathrm{dec}, s, k\rangle\} \\ \| \overline{c_{out}}(\mathbf{ok}) \| \overline{c}(s)$$

The adversary is now able to emit s on channel c and, even if it was not necessary to learn s, the process P_4 has progressed to his last state, which would not have been possible with another symbolic model.

Let now show how we also capture the computational attack described for Example 5. The adversary is as follows :

$$A_5 = \overline{c_{in}}(x_3).eq(\{\pi_1(y_1)\}^{\pi_3(y_1)}_{\pi_2(y_1)}, x_1)$$

The localized process $\emptyset, \emptyset, \emptyset, P_5 \| A_5$ reduces in some steps to $\emptyset, \{y_1\}, \emptyset, Q_1$ where Q_1 is

$$(\nu s, r, r_1, r_2, k_{AB})\{x_1 \mapsto \{a\}^r_{k_{AB}}, x_2 \mapsto k, x_3 \mapsto \{\{s\}^{r_1}_k, \{a\}^r_{k_{AB}}\}^{r_2}_{k_{AB}}\} \\ \| \{z \mapsto \{\{s\}^{r_1}_k, \{a\}^r_{k_{AB}}\}^{r_2}_{k_{AB}}\} \| \{y_1 \mapsto \langle s, k, r_1, \{a\}^r_{k_{AB}}, \mathrm{enc}\rangle\} \\ \| [\mathrm{EQ}(\{s\}^{r_1}_k, \{a\}^r_{k_{AB}}]\overline{c_{out}}(\mathbf{bad}) \| eq(\{s\}^{r_1}_k, \{a\}^r_{k_{AB}})$$

$\emptyset, \{y_1\}, \emptyset, Q_1 \xrightarrow{eq(\{s\}_k^{r_1}, \{a\}_{k_{AB}}^r, \varepsilon, \tau, \varepsilon} \{\{s\}_k^{r_1} = \{a\}_{k_{AB}}^r\}, \emptyset, \emptyset, Q_2$ where Q_2 is :

$$(\nu s, r, r_1, r_2, k_{AB})\{x_1 \mapsto \{a\}_{k_{AB}}^r, x_2 \mapsto k, x_3 \mapsto \{\{s\}_k^{r_1}, \{a\}_{k_{AB}}^r\}_{k_{AB}}^{r_2}\}$$
$$\|\{z \mapsto \{\{s\}_k^{r_1}, \{a\}_{k_{AB}}^r\}_{k_{AB}}^{r_2}\}\|\{y_1 \mapsto \langle s, k, r_1, \{a\}_{k_{AB}}^r, \text{enc}\rangle\}$$
$$\|\overline{c_{out}}(\textbf{bad})$$

where P_5 is in the bad state we wanted to avoid.

3.4 Observational equivalence

We recall the classical definition of observational equivalence, stating that there is no context (or adversary) yielding an emission on a channel c in one experiment, and no emission on c in the other experiment:

Definition 4 (observational equivalence). *An* evaluation context *is a process $C = (\nu\bar{\alpha})([\cdot]\|P)$ where P is a process. We write $C[Q]$ for $(\nu\bar{\alpha})(Q\|P)$. A context (resp. process) is called* closed *if* $\text{fv}(C) \cap \mathcal{V}_1 = \emptyset$. *Let us note that we do not forbid free names.*

The observational equivalence relation $\sim_o$ is the largest equivalence relation on completed processes such that $A \sim_o B$ implies :

- *If, for some evaluation 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', $B \xrightarrow{*} C'[\overline{c}(s') \cdot B']$*
- *If $A \xrightarrow{*} A'$, then for some B', $B \xrightarrow{*} B'$ and $A' \sim_o B'$*
- *For any closed evaluation context C, $C[A] \sim_o C[B]$*

In the proof, we also rely on static equivalence, in order to model the indistinguishability of two sequences of term for the adversary. Two frames ϕ, ϕ' are statically equivalent if, for any public term sequence $s_1, \ldots, s_k$ and any predicate p, $E \models p(s_1, \ldots, s_k)\phi$ iff $E \models p(s_1, \ldots, s_k)\phi'$.

4 Computational interpretation

We only need a small fragment of our calculus in order to describe the vast majority of protocols. These are called *simple processes* and are built as described in Section 4.1. We then provide their computational interpretation in Section 4.2.

4.1 Simple processes

Definition 5. *A* simple condition *with respect to a set of terms S is a conjunction of atomic formulas of one of the following forms :*

- $\text{M}(s)$ *where s contains only destructors, names and variables*
- $\text{EQ}(s_1, s_2)$ *where each s_i is of one of the two following forms :*
  - *s_i contains only destructors, names and variables*

- s_i *is a subterm of the a term in* S.

We also exclude the case in which s_1 *and* s_2 *are two subterms of the frame.*

Let $\bar{x}$ be a sequence of variable in $\mathcal{V}_1$, i a name called *pid* (the process identifier), and $\bar{n}$ a sequence of names, S be a set of terms such that $\mathrm{fv}(S) \subseteq \bar{x}$ and $\mathrm{fn}(S) \subseteq \bar{n}$. We define recursively *basic processes* $\mathcal{B}(i, \bar{n}, \bar{x}, S)$ as follows.

- $\mathbf{0} \in \mathcal{B}(i, \bar{n}, \bar{x}, S)$
- If $B \in \mathcal{B}(i, \bar{n}, \bar{x}, S \cup \{s\})$, $s \in \mathcal{T}(\bar{n}, \bar{x})$, Φ is a simple condition with respect to S such that $\mathrm{fn}(\Phi) \subseteq \bar{n}$ and $\mathrm{fv}(\Phi) \subseteq \bar{x}$, then :

$$[\Phi \wedge \mathrm{M}(s)]\overline{c_{out}}(s).B \qquad \in \mathcal{B}(i, \bar{n}, \bar{x}, S)$$

  If Φ is true, the the process checks if s is well formed and sends it out.
- If $B \in \mathcal{B}(i, \bar{n}, \bar{x}, x, S)$ and $x \notin \bar{x}$ then

$$c_{in}(x) \cdot [\mathrm{EQ}(\pi_1(x), i)]B \qquad \in \mathcal{B}(i, \bar{n}, \bar{x}, S)$$

  The process checks that it was the intended recipient of the message and processes it.

Basic processes are sequences of inputs and tests followed by an output. Else branches must be trivial. Basic processes are used to build *simple processes*.

Definition 6. *A* simple process *is obtained by composing and replicating basic processes, hiding some names and variables. Formally it is a process of the following form :*

$$(\nu \bar{n})[(\nu \bar{x}_1, \bar{n}_1 B_1 \| \sigma_1) \| \cdots \| (\nu \bar{x}_k, \bar{n}_k B_k \| \sigma_k) \| \\ !(\nu \bar{z}_1, l_1, \bar{m}_1 \overline{c_{out}}() B'_1) \| \cdots \| !(\nu \bar{z}_n, l_n, \bar{m}_n \overline{c_{out}}() B'_n)]$$

with $B_j \in \mathcal{B}(i_j, \bar{n} \uplus \bar{n}_j, \bar{x}_j, \emptyset), \mathrm{dom}(\sigma_j) \subseteq \bar{x}_j, B'_j \in \mathcal{B}(l_j, \bar{n} \uplus \bar{m}_j, \bar{z}_j, \{l_j\})$. *Let us note that each replicated process outputs its pid in order to let the adversary communicate with it.*

We also assume that for every subterm $\{t\}_k^v$ *occurring in a process,* v *is a name which occur only in this term and is restricted. We allow several occurrences of the term* $\{t\}_k^v$. *This ensures that the randomness of an encryption is re-used somewhere else.*

In what follows, we also assume that no key cycle is generated by the process. This can be ensured by defining a key hierarchy.

4.2 Computational model

As in [CLC08b] each simple process is interpreted as a Communicating Turing Machine (), and we can relate each state of a process to a state of the corresponding Turing Machine. We only give the implementation hypotheses here.

The implementation of the symmetric encryption is a joint IND-CPA and INT-CTXT symmetric encryption scheme. Let $\mathcal{K}$ be the key generation algorithm, $\mathcal{E}$ the encryption algorithm an $\mathcal{D}$ the decryption algorithm. All honest keys are drawn using $\mathcal{K}$ and, for any key k, message m, and randomness r, $\mathcal{D}(\mathcal{E}(m,k,r),k) = m$.

We assume that pairing is non ambiguous and that there are four different tags, one for the pairs, one for the encryptions, one for the keys and one for the honest nonces; every bitstring starts with the tag corresponding to the last constructor used to build it. Dishonest messages need not to be properly tagged. We assume that the symbolic length function l is such that two terms have the same length if and only if the corresponding bitstrings have the same length. It is easy to build such a function for example if the length of the nonces are proportional to the security parameter η, the computational length of pair is $|v\|w| = |v|+|w|+a.\eta$ for some a and the length of the encryption is $|\mathcal{E}(m,k,r)| = |m| + b.\eta$ for some b.

We also need to give the implementation of the predicates used by the simple processes : $[\![M]\!]$ is the set of bitstring which are different from $\perp$, and $[\![EQ]\!]$ is the set pairs of non $\perp$ identical bitstrings.

Let us note that for simple processes, the computation of the network answer to a request is always in polynomial time. This ensures that if the attacker is a PPT (with an oracle for the process), then running it with the process as oracle is still in polynomial time.

5 Main result

We show two main results. First, any computational trace is now reflected by a symbolic one, even in the presence of an attacker that dishonestly generates its keys. This allows to transfer all trace-based properties. Second, we show that we can also transfer equivalence-based properties, showing that observational equivalence implies computational indistinguishability.

5.1 Results

Let us start by defining the sequence of the messages exchanged between P and $\mathcal{A}$, and what it means for such a trace to be fully abstracted in the process P. Note that, given τ and η, the behaviour of $\mathcal{A}$ interacting with P, denoted by $[\![P]\!]^\tau_\eta \| \mathcal{A}_\tau$, is deterministic.

Definition 7.

Let us now define *symbolic traces* of a process P :

Definition 8. *$s = \alpha_1.\cdots.\alpha_n$ is a* trace of *P if $\emptyset,\emptyset,\emptyset, P \xrightarrow{\alpha_1} E^1, X^1_w, X^1_c, P_1 \xrightarrow{\alpha_2} \cdots \xrightarrow{\alpha_n} E^n, X^n_w, X^n_c, P_n$ and for all $i \leq n$ if , then $P_{i-1} = P' \| \phi$ with ϕ a frame and $\phi \vdash_{E_{i-1}}$.*

Definition 9 (Full abstraction). *Let* $\gamma_1, L_1 \xrightarrow{m_1} \gamma_2, L_2 \cdots \xrightarrow{m_n} \gamma_n, L_n$ *be an execution,* s *be a trace of* P*. Let us write* $s = \alpha_1.\cdots.\alpha_m$*. Let* $\alpha_{n_1} \cdots \alpha_{n_k}$ *be the subsequence of* s *which are inputs.*

s fully abstracts $\gamma_1, L_1 \xrightarrow{m_1} \gamma_2, L_2 \cdots \xrightarrow{m_n} \gamma_n, L_n$ *if* $= n$ *and* $\forall j \leq n$

–

– *If* $P \xrightarrow{\alpha_1.\cdots.\alpha_{n_{j+1}-1}} E^j, X_c^j, X_w^j, Q^j \| \phi^j$ *with*
  - $[\![Q^j]\!]_\eta^\tau = \gamma_j$
  - $[\![\phi_j \cap \{x \mapsto t | t \in \mathcal{T}, x \in \mathcal{V}_1\}]\!]_\eta^\tau = L_j$
  - $\forall (s = t) \in E^j, [\![s]\!]_\eta^\tau = [\![t]\!]_\eta^\tau$
  - $\forall (s \neq t) \in E^j, [\![s]\!]_\eta^\tau \neq [\![t]\!]_\eta^\tau$

A computational trace Γ *is* fully abstracted *if there exists a trace* s *of the process* P *which fully abstracts* Γ*.*

We are now able to give our full abstraction theorem :

Theorem 1. *Let* P *be a simple process without key cycles. For every PPT* $\mathcal{A}$*, for every security parameter* η*, the sequence* $\text{Messages}(P, \eta, \tau)$ *is fully abstracted with overwhelming probability (over* τ*).*

This result ensures that it is sufficient to prove trace properties in our model for them to hold in the computational model. Our second result is the computational soundness of observational equivalence.

Theorem 2. *Let* P *and* Q *be two simple processes without key cycles, such that* $P \sim_o Q$*. Then* $[\![P]\!] \approx [\![Q]\!]$*.*

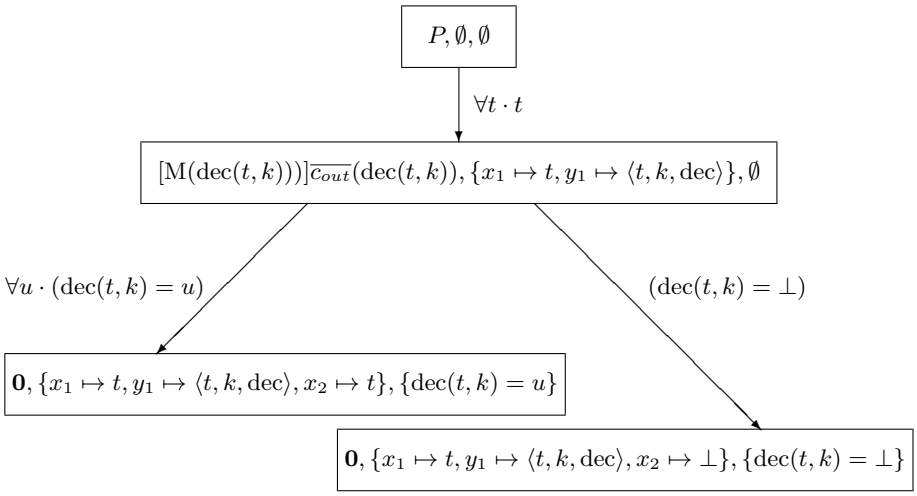
This second result allows us to prove any indistinguishability property in our model instead of proving it in a computational setting.

5.2 Sketch of proof

The main tool of the proof of Theorem 2 is the use of execution trees. The execution tree of a process is the set of traces of a process organized in a tree. An example is provided in Figure 4. An execution tree is not necessarily the execution tree of a process. This generalization allows to consider transformations on process trees, which would not have being possible directly on processes.

The proof then proceeds in the following steps.

1. We show that the observational equivalence of processes transfers to equivalence of process execution trees.
2. We then replace all honest encryptions in a process tree by encryption of zeros, showing that the trees are symbolically equivalent.
3. If two trees are equivalent, then their computational interpretation are indistinguishable.



The edge labelled by $\forall t \cdot t$ is in fact a multiple edge representing the edges t for all t. As well the edge $\forall u \cdot (\mathrm{dec}(t,k) = u)$ is a multiple edge representing the edge for all u.

Fig. 4. Process execution tree corresponding to the process $P = c_{in}(x) \cdot$ `if` $\mathrm{M}(\mathrm{dec}(x,k))$ `then` $\overline{c_{out}}(x)$ `else` $\overline{c_{out}}(\bot)$ with k dishonest.

4. The only result left to prove is that a process is computationally indistinguishable from its process tree. As all the traces are listed in the tree, this amounts in proving Theorem 1. For this, we need to classify the cases in which the full abstraction fails, and we then add these failure cases in the execution trees (this is the originality of our approach with respect to [CLC08b]) and prove that these cases can not be found with a non negligible probability by the adversary given the computational hypotheses.

6 Application

We show how our framework can be used to show the computational security for protocols, even in the case where the attacker can create interesting equalities between ciphertexts using dishonest keys. This section also demonstrates that our symbolic model does not over-approximate too much the behavior of computational attackers: our model is fine enough to complete security proof.

6.1 Specification of the protocol

We describe below a protocol that is designed in such a way that honest participants may indeed use dishonest keys, making the security proof more challenging. The protocols aims at securely transmitting s_{AB} from B to A. We will use N as a shortcut for $\{N'\}_{k_N}^{r_N}$ where k_N is a secret key known only to $\mathbf{A}$. We actually simply need N to be of type cyphertext to make our example more interesting.

$\mathbf{A} \longrightarrow \mathbf{B}\ k_1, \{\{\{k_2\}_{k_3}\}_{k_1}, N\}_{k_{AB}}$
$\mathbf{B} \longrightarrow \mathbf{A}\ \{\{k_2\}_{k_3}, N\}_{k_{AB}}$
$\mathbf{A} \longrightarrow \mathbf{B}\ \{N, k_3\}_{k_{AB}}$
$\mathbf{B} \longrightarrow \mathbf{A}\ \{s_{AB}\}_{k_2}$

k_{AB} is a long term shared key between A and B, the keys k_1, k_2, and k_3 are fresh and N is a fresh "session" nonce.

We first specify this protocol in the applied π-calculus, using a syntax with pattern matching for simplicity reasons. The interpretation is that the protocol checks with M and EQ all the constraints given by the pattern matching. For the sake of clarity, we omit the verifications of process session identifiers. But it would be easy to transform our process into a simple process. Instead of restricting all the names used in N, we simply write (νN) as a shortcut for $(\nu N', k_N, r_N)$.

$$\begin{aligned} P_A = (\nu k_1, k_2, k_3, N, r_1, r_2, r_3, r_4)\overline{c_{out}}(\langle k_1, \{\{\{k_2\}^{r_1}_{k_3}\}^{r_2}_{k_1}, N\}^{r_3}_{k_{AB}}\rangle).\\ c_{in}(\{\{k_2\}^{r_1}_{k_3}, N\}^{-}_{k_{AB}}).\overline{c_{out}}(\{N, k_3\}^{r_4}_{k_{AB}})\\ P_B = (\nu r_5, r_6)c_{in}(\langle x_1, \{\{x\}^{-}_{x_1}, x_N\}^{-}_{k_{AB}}\rangle).\overline{c_{out}}(\{x, x_N\}^{r_5}_{k_{AB}}).\\ c_{in}(\{x_N, x_3\}^{-}_{k_{AB}}).[\mathrm{M}(\mathrm{dec}(x, x_3))]\overline{c_{out}}(\{s_{AB}\}^{r_6}_{\mathrm{dec}(x,x_3)}) \end{aligned}$$

The process we consider is $(\nu k_{AB}, s_{AB})!P_B \| !P_A$ and the security property we want to prove is as follows. For every $\mathcal{A}$ PPT with an oracle :

$$\Pr(\mathcal{A} \| [(\nu k_{AB}, s_{AB})!P_B \| !P_A] \to s_{AB}) = \mathrm{negl}(\eta)$$

where for a PPT M, $M \leftarrow m$ stands for M accepts writing m on its output tape.

This *a priori* not straightforward since introducing dishonest keys allows the attacker to tamper with the normal behavior of the protocol. For example, the adversary can learn any instance of N and can obtain as output $\{u, k_3^l\}_{k_{AB}}$ where u is any deducible term, with k_3^l an instance of k_3. Indeed, once the attacker knows $\{N, k_3\}_{k_{AB}}$, it can forward it to B together with a dishonest key, that is sending $k_1^*, \{N, k_3\}_{k_{AB}}$. As explained in Example 4, attempting to decrypt with the dishonest key k_1^* potentially reveals N to the attacker. Then for any deducible message u, the attacker can forge a dishonest key k_2^* such that $\mathtt{dec}(N, k_2^*) = u$, which allows the attacker to obtain $\{u, k_3^l\}_{k_{AB}}$ from B.

6.2 Security

Despite the behaviors described in the preceding section, the protocol is secure and we are able to prove it. Applying Theorem 1, it is sufficient to prove weak secrecy in our symbolic model, that is, it is sufficient to prove the following proposition.

Proposition 1. *The process* $(\nu k_{AB}, s_{AB})!P_B \| !P_A \| c_{in}(x).[x = s_{AB}].\overline{c_{error}}$ *never emits on channel* c_{error}.

The process $c_{in}(x).[x = s_{AB}].\overline{c_{error}}$ serves as a witness, checking whether the intruder is able to emit s_{AB}. The idea of the proof is that the second component of the pair is only transmitted,so dishonnest keys will not help learning something about it, and the k_3 stays secret, which ensures that k_2 also stays secret. This is formally proved by computing on over-approximation of the messages learned by the attacker.

7 Conclusion

We designed a symbolic model, for which the observational equivalence is sound, even when the attacker may forge his own keys. We believe that it is the first result on computational soundness considering dishonest keys.

We assumed in this work that the processes do not contain non-trivial conditional branching and no nested replications, but it should not be very hard to extend our results to these cases.

Another issue, that might be the subject of future work, concerns the automatisation of the proofs of observational equivalence, in this new model. It is likely that deducibility constraint solving can be extended to ground equational theories, which is what we need.

References

[AF01] M. Abadi and C. Fournet. Mobile values, new names, and secure communication. In *Principles of Programming Languages (POPL'01)*, pages 104–115, 2001.

[AG99] M. Abadi and A.D. Gordon. A calculus for cryptographic protocols: the spi calculus. *Information and Computation*, 148(1), 1999.

[AR00] 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*, 2000.

[BHO09] G. Bana, K. Hasebe, and M. Okada. Computational semantics for first-order logical analysis of cryptographic protocols. In *LNCS*, volume 5458 of *Lecture Notes in Computer Science*, pages 33–56. Springer, 2009.

[Bla05] Bruno Blanchet. An automatic security protocol verifier based on resolution theorem proving (invited tutorial). In *20th International Conference on Automated Deduction (CADE-20)*, Tallinn, Estonia, July 2005.

[Bla08] Bruno Blanchet. A computationally sound mechanized prover for security protocols. *IEEE Trans. on Dependable and Secure Computing*, 5(4):193–207, 2008. Special issue IEEE Symposium on Security and Privacy 2006.

[BP04] M. Backes and B. Pfitzmann. Symmetric encryption in a simulatable dolev-yao style cryptographic library. In *Proc. IEEE Computer Security Foundations workshop*, 2004.

[BPW03] M. Backes, B. Pfitzmann, and M. Waidner. A composable cryptographic library with nested operations. In *Proc. 10th ACM Concerence on Computer and Communications Security (CCS'03)*, 2003.

[CC11] H. Comon-Lundh and V. Cortier. How to prove security of communication protocols? A discussion on the soundness of formal models w.r.t. computational ones. In *28th Annual Symposium on Theoretical Aspects of Computer Science (STACS'11)*, volume 9 of *LIPIcs*, pages 29–44, 2011.

[CLC08a] H. Comon-Lundh and V. Cortier. Computational soundness of observational equivalence. In *ACM Conf. Computer and Communication Security (CCS'08)*, 2008.

[CLC08b] H. Comon Lundh and V. Cortier. Computational soundness of observational equivalence. Research Report RR-6508, INRIA, 2008.

[DY81] D. Dolev and A.C. Yao. On the security of public key protocols. In *Proc. IEEE Symp. on Foundations of Computer Science*, pages 350–357, 1981.

[KT09] R. Küsters and M. Tuengerthal. Computational Soundness for Key Exchange Protocols with Symmetric Encryption. In *16th ACM Conference on Computer and Communications Security (CCS 2009)*, pages 91–100, 2009.

[MW04] D. Micciancio and B. Warinschi. Completeness theorems for the Abadi-Rogaway language of encrypted expressions. *Journal of Computer Security*, 2004.

[RS98] P. Ryan and S. Schneider. An attack on a recursive authentication protocol: a cautionary tale. In *Information Processing Letters*, volume 65, pages 7–10. 1998.

[War03] B. Warinschi. A computational analysis of the needham-schroeder(-lowe) protocol. In *16th Computer Science Foundation Workshop (CSFW'03)*, pages 248–262, 2003.

A Rewriting Systems

We assumed that $E \cup E_0$ can be completed into a convergent rewriting system, that minimizes the number of destructors. Let us precise here this statement and show that the completion can indeed been performed (actually in polynomial time), when E is a finite set of ground equations (which is what we need: E is always a finite set of ground equations in localized processes).

Lemma 1. *Let E be a finite set of ground equations over the finite alphabet $A \cup \mathcal{N}$ where $A = \{\{\_\}^{\_}_{\_}, \mathrm{dec}(\_,\_), \langle\_,\_\rangle, \pi_1(\_), \pi_2(\_)\}$. We can compute in polynomial time (w.r.t. $E, \mathcal{N}$) a canonical term rewriting system $\mathcal{R}_E$ such that $=_{\mathcal{R}}$ is the smallest congruence on ground terms $\mathcal{T}(A \cup \mathcal{N})$ that contains $=_{E \cup E_0}$.*

Proof sketch: In a first step, we flatten the equations in E, introducing a linear number of additional constants C_0: we may assume w.l.o.g. that equations in E are of one of the following forms: $f(a_1, a_2) = a, g(a_1) = a, a_1 = a$ where $f, g, a_1, a_2, a \in A \cup C_0$.

We choose then an arbitrary linear ordering $\geq$ on constants and run a Knuth-Bendix completion procedure w.r.t. a lexicographic path ordering extending the precedence

$$\mathrm{dec} > \pi_1 > \pi_2 > \{\_\}^{\_}_{\_} > \langle\_,\_\rangle > C_0 > \mathcal{N}$$

Every rule that is eventually generated by this procedure is of one of the forms $f(a_1, a_2) \to a, g(a_1) \to a, a_1 \to a$ and $\mathrm{dec}(\{x\}^z_{a_1}, a_2) \to x$, $\pi_1(\langle x, y\rangle) \to x$, $\pi_2(\langle x, y\rangle) \to y$, where a_1, a_2, a are constants. The only non-trivial overlap is between a rule $\mathrm{dec}(\{x\}^z_a, b) \to x$ and a rule $\{c\}^r_a \to d$, which yields $\mathrm{dec}(d, b) \to c$, that has the desired form.

Since there are at most a cubic number of such rules, the procedure halts in cubic time.

In addition, if the input set of ground equations E only contains equations of one of the following forms:

- $\mathrm{dec}(u, k) = v$ where u, v contain only constructors and names and $k \in \mathcal{K}_1 \cup \mathcal{K}_2$
- $\{u\}^r_k = v$ where u, v contain only constructors and names and $k \in \mathcal{K}_1 \cup \mathcal{K}_2$

then, for any u that does not contain any constant in C_0, $u{\downarrow_E} \in C_0$ iff there is a t containing only constructors and names (and such that key positions are held with elements in $\mathcal{K}_1 \cup \mathcal{K}_2$) such that $t{\downarrow_E} = u{\downarrow_E}$. This is simply an invariant in the completion that was sketched in the previous lemma.

In other words: given the type of equations that are added in E, $E \models \mathrm{M}(u)$ iff there is a term v, built with constructors and names only, using only key names in key positions and such that $u =_{E \cup E_0} v$. This is used in a subcase of a proof in the appendix G.

B Key hierarchy

We say that u is a plaintext subterm of t if u appears as a subterm of t in a position which is not a key position or a randomness position and u is a name, a constant or a variable. Let us define this more formally :

Definition 10. *Let us write* $\mathrm{pst}(t)$ *for the set of plaintext subterms of* t*, we have :* $\mathrm{pst}(n) = \{n\}$ *if* n *is a name, a constant or a variable,* $\mathrm{pst}(\langle t_1, t_2\rangle) = \mathrm{pst}(t_1) \cup \mathrm{pst}(t_2)$ *et* $\mathrm{pst}\{u\}^r_v = \mathrm{pst}(u)$.

Example 1 $\mathrm{pst}(\langle k_1, \{\{a\}^{r_2}_{k_2}\}^{r_1}_{k_1}\rangle) = \{a, k_1\}$

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 $k' \in \mathrm{pst}(u)$. We say that there is a key cycle in S if there exists $k_1, \cdots, k_n$ such that k_{i+1} encrypts k_i for $1 \leq i \leq n-1$ and k_1 encrypts k_n.

The usual way to avoid key cycles is to define a key hierarchy. In order to define a key hierarchy we need to explain how renaming is done in processes. This appears mostly in the replication of a process : $!((\nu n)P) \to (\nu n)P \| !((\nu n)P)$. We will assume that the name n is successively renamed in $n_1, n_2, n_3, \cdots$

Definition 11. *We say that a simple process* P *admits a key hierarchy if there exists a strict order* $<$ *on the honest key of* P *such that for every honest key* k, k' *in the image of* σ*, for every evaluation context* C*, for every process* Q *such that* $C[P] \xrightarrow{*} Q$ *and such that* $\nu\bar{n} \cdot \sigma$ *is the frame of* Q*, then if* k *encrypts* k' *in* σ *then* $k' < k$.

We note that if there exists a key hierarchy, then no key cycle can be created. We also note that if there exists a key hierarchy, no honest key can be learned by the adversary.

C computational hypotheses

Definition 12. *A cryptographic scheme is said IND-CPA if for every non zero polynomial* P*, for every PPT* $\mathcal{A}$ *with a oracle, there exists* N *such that for every* $\eta > N$*,*

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

With $\bar{r} = r_1, r_2, \cdots$ *the sequence of randomness used by,* $\mathcal{O}_k$ *the oracle which on a request* x *sends out* $\mathcal{E}(x, k, r_i)$ *and* $\mathcal{O}'_k$ *the oracle which on a request* x *sends out* $\mathcal{E}(0^{l(x)}, k, r_i)$.

Definition 13. *A cryptographic scheme is said INT-CXT if for every non zero polynomial P, for every PPT $\mathcal{A}$ with a oracle, there exists N such that for every $\eta > N$,*

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

Where $S = \{\{x_1\}_k^{r_1}, \{x_2\}_k^{r_2}, \cdots\}$ is the sequence of answers of the oracle $\mathcal{O}_k(x_i) = \{x_i\}_k^{r_i}$

Let us note that these two hypotheses do not ensure anything on dishonest keys.

D Trees

D.1 Static Equivalence

Definition 14 (Static equivalence). *Let $\phi = \nu\bar{n}\sigma$ be a frame, E an equation set. Let p be a predicate, $s_1, \cdots, s_k$ terms. We write $\phi \vDash_E p(s_1, \cdots, s_k)$ if $s_1, \cdots, s_k$ are public and $E \vDash p(s_1\phi, \cdots, s_k\phi)$. We say $\phi_1 = \nu\bar{n}_1 \cdot \sigma_1, E_1$ and $\phi_2 = \nu\bar{n}_2 \cdot \sigma_2, E_2$ are statically equivalent (written $\phi_1, E_1 \sim \phi_2, E_2$) if*

1. $\mathrm{dom}(\phi_1) = \mathrm{dom}(\phi_2)$
2. $\forall s_1, \cdots, s_k \in \mathcal{T}, \forall p \in \mathcal{P}.\ \phi_1 \vDash_{E_1} p(s_1, \cdots, s_k) \Leftrightarrow \phi_2 \vDash_{E_2} p(s_1, \cdots, s_k)$

D.2 Execution trees

Let us define execution trees. Let S be the set of pairs $\langle i, u\rangle$ of a *pid* and a term (for the request u to the process i) and equations $(\{s\}_k^r = v), (\{s\}_k^r \neq v), (\mathrm{dec}(s, k) = t)$ (for the equalities added by the adversary)

For simplicity reasons, we will write $t \in \phi$ for a term t and a frame ϕ if $t \in \phi(\mathcal{V})$.

Definition 15. *An execution tree T is a function from a prefix-closed set $Pos(T)$ of S^* (called positions of T) to the triples (P, ϕ, E) with P a process $\phi = \nu\bar{n} \cdot \mathcal{L}$ and E an equation set. If $p \in Pos(T)$, and $T(p) = (P, \phi, E)$ we write $\phi = \phi(T, p)$ and $E = E(T, p)$. T should satisfy the following constraints :*

- $\forall p, q \in Pos(T)$ *if* $p > q$ *then* $\phi(T, q) \subseteq \phi(T, p)$, $E(T, q) \subseteq E(T, p)$
- $\forall p \cdot t \in Pos(T)$, $t \in \phi(T, p \cdot t)$ *or* $t \in E(T, p \cdot t)$
- $\forall p \cdot t \in Pos(T)$, $\phi(T, p) \vdash_{E(T,p)} t$
- $\forall p \in Pos(T)$ $t \in E(T, p)$ *if and only if* $t \in p$
- $\forall p \cdot t \in Pos(T)$ *if t is an equation or a disequation, then* $\mathrm{wf}_{E(T,p)}^{\phi(T,p)}(t)$
- $\forall p \cdot t, p \cdot u \in Pos(T)$ *if* $t =_{E(T,p)} u$ *then* $t = u$ *(this condition simply ensures that all branches are really different in order to have a deterministic tree)*
- $\forall p \in Pos(T)$ *there is a finite number of s, k such that* $\exists t, p.(\mathrm{dec}(s, k) = t) \in Pos(T)$.

– $\forall p \in Pos(t)$ *there is a finite number of* s, k, r, v *such that* $p.(\{s\}^r_k = v) \in Pos(T)$ *or* $p.(\{s\}^r_k \neq v) \in Pos(T)$
– *There is no infinite suffix containing only equations*
– $\forall p \in Pos(T)$ *if there exists a successor* $p.t$ *of* p *such that* t *is an equation, then for all* $s \in S$ *such that* $p.s \in Pos(T)$, s *is an equation*

Let us note that process trees are not necessarily finite. They can contain infinite branching and infinite paths. This will not lead to a problem as the number of messages exchanged on the computational side is polynomially bounded.

As we want to prove soundness for observational equivalence, we need to define an observational equivalence on trees.

Definition 16. $\sim$ *is the largest equivalence relation on execution trees such that if* $T_1 \sim T_2$ *then*

– $\phi(T_1, \epsilon) \sim \phi(T_2, \epsilon)$
– *there exists a bijection* β *from* S *into itself such that for every size 1 position of* T_1 *we have* $T_1|_t \sim T_2|_{\beta(t)}$

D.3 Process trees

T_P is the ordered set of all execution of P. Each node of T_P is labeled by (Q, ϕ, E) where Q is the current state of the process, E the equation set and ϕ the sequence of messages sent or received by P on the network plus the additional knowledge given to the adversary.

Definition 17. *Let us define* T_P *by induction.*

$T_P(\epsilon) = (P, \emptyset, \emptyset)$. *Let* $p \in Pos(T_P)$, *we write* $T_P(p) = ([E, X_w, X_c, P'], \bar{\nu}\bar{n}' \cdot \mathcal{L}, E)$

– *Let us consider the case* $X_w = \emptyset$, *in that case* $X_c = \emptyset$. *The only reductions of* E, X_w, X_c, P' *are as follows. A message* $u = \langle l_i, u' \rangle$ *is sent to a process of pid* l_i *:*

$$P' = \nu\bar{n}, \nu\bar{x} \cdot Q^p_1 \| \sigma^p_1 \| \cdots \| Q^p_N \| \sigma^p_N \| S$$

Where Q^p_j *is either* $\mathbf{0}$ *or* $c(x^p_j) \cdot P^p_j$. *Then* $q = p \cdot \alpha \in Pos(T_P)$ *if* $\alpha = \langle l_i, u \rangle$, $Q^p_i \neq \mathbf{0}$, *and* $\nu n' \cdot \mathcal{L} \vdash_E \alpha$. *Let then* $P^p_i = [\Phi]Q$. *We have*

$$E, \emptyset, \emptyset, P^p_i \| \{x^p_j \mapsto \alpha\} \xrightarrow{\varepsilon} E, Y_w, \emptyset, \varphi \| P^p_i \| \{x^p_j \mapsto \alpha\}$$

for some φ. *In that case,*
- *If* $Y_w \neq \emptyset$ *the adversary should commit on some equations.*

$$T_P(p \cdot \alpha) = ([E, Y_w, \emptyset, \nu\bar{n}, \nu\bar{x} \cdot Q^p_1 \| \sigma^p_1 \| \cdots \| P^p_i \| \{x^p_j \mapsto \alpha\} \| \cdots \| Q^p_N \| \sigma^p_N \| S], \nu\overline{n'} \cdot \mathcal{L} \cdot \alpha \cdot \varphi, E)$$

where $\mathcal{L}.\alpha.\varphi$ *is* $\mathcal{L}.\alpha.\bar{\gamma}$ *where* $\bar{\gamma}$ *is the sequence* $\varphi(y_1), \cdots, \varphi(y_n)$ *where* $\{y_1, \cdots y_n\}$ *is the domain of* φ *ordered according to the increasing order on* $\mathcal{V}_2$

- *If $Y_w = \emptyset$ then there are no equations to add and*

$$E, \emptyset, \emptyset, Q_i^p \| \sigma_i^p \xrightarrow{c_{in}(\alpha)} \xrightarrow{\overline{c_{out}}(\alpha_1), \cdots, \overline{c_{out}}(\alpha_n)} E, \emptyset, \emptyset, Q_i^q \| \sigma_i^p \| \{x_i^p \mapsto \alpha\}$$

Let us now define $Q_j^q = Q_j^p$ and $\sigma_j^q = \sigma_j^p$ for $j \neq i$.

$$T_P(p \cdot \alpha) = ([E, \emptyset, \emptyset, \nu\bar{n}, \nu\bar{x} \cdot Q_1^q \| \sigma_1^q \| \cdots \| Q_N^q \| \sigma_N^q \| S], \\ \nu\overline{n'} \cdot \mathcal{L} \cdot (\alpha, \alpha_1, \cdots, \alpha_n), E)$$

– *Let us now consider the case where X_w in non empty. The adversary should commit on an equation :*

$$P' = \nu\bar{n}, \nu\bar{x}. \cdot Q_1^p \| \sigma_1^p \| \cdots \| P_i^p \| \cdots \| Q_N^p \| \sigma_N^p \| S$$

With exactly one $P_i^p = [\Phi]Q$. Let us write φ for the maximal frame of P'. In that case $p.\alpha \in Pos(T_P)$ if $\mathrm{wf}^E_{X_w, \varphi_{\mathcal{V} \setminus X_c}}(\alpha)$ (note that α can be an equation or a disequation). We write $q = p.\alpha$.
Let us write $E, X_w, X_c, P_i^p \| \varphi \xrightarrow{eq(\alpha)} \xrightarrow{\varepsilon} E \cup \{\alpha\}, Y_w, Y_c, \phi, P_i^p \| \varphi$
- *If $Y_w \neq \emptyset$. Defining $Q_j^q = Q_j^p$ for $i \neq j$ as previously and $Q_i^q = P_i^p$.*

$$T_P(p \cdot \alpha) = ([E \cup \{\alpha\}, Y_w, Y_c, \nu\bar{n}, \nu\bar{x} \cdot Q_1^q \| \sigma_1^q \| \cdots \| Q_N^q \| \sigma_N^q \| S], \\ \nu\overline{n'} \cdot \mathcal{L} \cdot \alpha \cdot \phi, E \cup \{\alpha\})$$

- *If $Y_w = \emptyset$. Let n be maximal such that*

$$E \cup \{\alpha\}, \emptyset, \emptyset, P_i^p \xrightarrow{\tau} \xrightarrow{\overline{c_{out}}(\alpha_1), \cdots, \overline{c_{out}}(\alpha_n)} E \cup \{\alpha\}, \emptyset, \emptyset, Q_i^q$$

In that case :

$$T_P(p \cdot \alpha) = ([E \cup \{\alpha\}, \emptyset, \emptyset, \nu\bar{n}, \nu\bar{x} \cdot Q_1^q \| \sigma_1^q \| \cdots \| Q_N^q \| \sigma_N^q \| S], \\ \nu\overline{n'} \cdot \mathcal{L} \cdot \alpha \cdot \phi \cdot (\alpha_1, \cdots, \alpha_n), E \cup \{\alpha\})$$

– *The last case is the case in which we activate a copy of a replicated process in $S = S_1 \| \cdots \| S_k$. In that case $q = p.new_j \in Pos(T_P)$ with $1 \leq j \leq k$ if $X_w = \emptyset$.*
Let us write $S_j = !(\nu\bar{y}, l, \bar{n}_j \overline{c_{out}}(l) \cdot B)$. We have

$$T_P(p.new_j) = (E, \emptyset, \emptyset, \nu\bar{n}\nu\bar{x}\nu\bar{y}\nu l \nu\bar{n}_j \cdot Q_1^p \| \sigma_1 \| \cdots \| Q_N^p \| \sigma_n \| B \| S, \\ \nu\overline{n'}\nu\overline{n_j} \cdot \mathcal{L} \cdot l, E)$$

In order to use this definition, we need the following lemma :

Lemma 2. *Let P and Q be simple processes. If $P \sim_o Q$ then $T_P \sim T_Q$.*

Proof. Assume that $T_P \not\sim T_Q$, and let us build a context which distinguishes between P and Q. First of all, we will construct, by induction on the length of $p \in \mathrm{Pos}(T_P)$ a family of one to one functions β_p such that :

– if β_p is defined, then, for any $q < p, \beta_q$ is defined. We define inductively B by $B(p.\alpha) = B(p).\beta_p(\alpha)$.
– if β_p is defined, then, for any $q \leq p, \phi(T_P, q) \sim \phi(T_Q, B(q))$
– either $\phi(T_P, p) \not\sim \phi(T_Q, B(p))$, then β_q is undefined for $q \geq p$
– or β_q is defined for all $q < p$ 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, $T_P|_{p.\alpha} \not\sim T_Q|_{B(p.\alpha)}$.

Let us now build β_p.

– If β is defined on any strict prefix of p and $T_P|_p \sim T_Q|_{B(p)}$, we use the definition of $\sim$: there is a one to one function β_p such that, for every $p.\alpha \in \text{Pos}(T_P)$, $T_P|_{p.\alpha} \sim T_Q|_{B(p).\beta_p(\alpha)}$
– If β is defined on any strict prefix of p and $T_P|_p \not\sim T_Q|_{B(p)}$ and $\phi(T_P, p) \sim \phi(T_Q, B(p))$, we define β_p as follows : let $\phi(T_P, p) = \nu\bar{n}_1.\sigma_1^p, E(T_P, p) = E_1$ and $\phi(T_Q, B(p)) = \nu\bar{n}_2.\sigma_2^p, E(T_Q, B(p)) = E_2$
  • if for all α such that $p.\alpha \in \text{Pos}(T_P)$ we have α is a term, then, by definition, $\sigma_1^p \vdash_{E_1} \alpha$, then there is a term t_α such that $\text{fn}(t_\alpha) \cap \bar{n}_1 = \emptyset$ and $E_1 \vDash t_\alpha\sigma_1^p = \alpha$. We let $\beta_p(\alpha) = t_\alpha\sigma_2^p$. Let us now show that β_p is one to one from $\mathcal{T}(\mathcal{N})/=_{E_1}$ to $\mathcal{T}(\mathcal{N})/=_{E_2}$. If $\beta_p(\alpha) =_{E_2} \beta_p(\alpha')$ then there are two term t and u, such that $E_2 \vDash t\sigma_2^p = u\sigma_2^p$ and $E_1 \vDash t\sigma_1^p = \alpha \wedge u\sigma_2^p = \alpha'$. Since $\phi(T_P, p), E_1 \sim \phi(T_Q, B(p)), E_2$ and equality is in our set of predicates we have $E_1 \vDash \alpha = \alpha'$.
  • if for all α such that $p.\alpha \in \text{Pos}(T_P)$ we have $\alpha = (\alpha_1 = \alpha_2)$ is an equation, then, by definition, $\sigma_1^p \vdash_{E_1} \alpha_1, \alpha_2$. There are two terms t_α, u_α such that $\text{fn}(t_\alpha, u_\alpha) \cap \bar{n}_1 = \emptyset$ and $E_1 \vDash t_\alpha\sigma_1^p = \alpha_1 \wedge u_\alpha\sigma_1^p = \alpha_2$. We let $\beta_p(\alpha) = t_\alpha\sigma_2^p = u_\alpha\sigma_2^p$. As previously, β_p is one to one.

This concludes the construction of the family β_p.

As $T_P \not\sim T_Q$ there is a minimal position p where β_p is undefined. Let $p = p_1.\alpha, p_2 = B(p_1)$. α can not be a constant $\texttt{new}_j$ since, in that case, $\nu\bar{n}_1.\mathcal{L}_1, E_1 = \phi(T_P, p_1), E(T_P, p_1) \sim \phi(T_P, p_2), E(T_P, p_2) = \nu\bar{n}_2.\mathcal{L}_2, E_2$ implies $\nu\bar{n}_1'.\mathcal{L}_1.l_1, E_1 \sim \nu\bar{n}_2'.\mathcal{L}_2.l_2, E_2$ and then rewriting the previous relation, we have :

$$\phi(T_P, p_1.\texttt{new}_j), E(T_P, p_1.\texttt{new}_j) \sim \phi(T_P, p_2.\texttt{new}_j), E(T_P, p_2.\texttt{new}_j)$$

A direct consequence of the definition of the process execution trees is that at each position p, the equation set of the process Q labelling the tree at position p is equal to the equation set $E(T_P, p)$ and the set of restricted names of $\phi(T_P, p)$ is equal to the set of restricted names of Q. It also follows that at each position the tree accepts term if and only if the process accepts terms.

Let us also note that if two sequences $l_1.\cdots.l_n$ and $m_1.\cdots.m_n$ are statically equivalent, if l_i stores knowledge obtained from tests (results from the rule $(R-Add)$) the m_i also does as the constants *enc* and *dec* exist only at this type of positions.

Let us define

– $T_P(p_1) = (E_1, X_{w,1}, X_{c,1}, P_1), \nu\bar{n}_1.\mathcal{L}_1, E_1$
– $T_Q(p_2) = (E_2, X_{w,2}, X_{c,2}, P_2), \nu\bar{n}_2.\mathcal{L}_2, E_2$

– $\phi_1, E_1 = \nu\bar{n}_1.\mathcal{L}_1, E_1 \sim \nu\bar{n}_2.\mathcal{L}_1, E_2 = \phi_2, E_2$

and, if $X_{w,1} = X_{c,1} = \emptyset$:

– $P_1 = (\nu\bar{n}'_1\nu\bar{x}_1)Q_1\|\theta^1_1\|\cdots\|Q_N\|\theta^1_N\|S_1$ where $Q_i \equiv c_{in}(x_i).Q'_i$
– $E_1, \emptyset, \emptyset, Q_i \xrightarrow{c_{in}(\alpha)} \xrightarrow{\bar{c_{out}}(\alpha_1),\cdots,\bar{c_{out}}(\alpha_m)} E_1, Y_w, \emptyset, Q''_i\|\theta^1_i\|\{x_i \mapsto \alpha\}\|\{Y_w \mapsto \bar{\gamma}\}$
where m is maximal.
– $\phi(T_P, p_1.\alpha) = \nu\bar{n}'_1.\mathcal{L}_1 \cdot (\alpha, \bar{\gamma}, \alpha_1, \cdots, \alpha_m)$
– $P_2 = (\nu\bar{n}_2\nu\bar{x}_2)R_1\|\theta^2_1\|\cdots\|R_N\|\theta^2_N\|S_2$ where $R_j \equiv c_{in}(z_j).R'_j$
– $E_2, \emptyset, \emptyset, R_j \xrightarrow{c_{in}(\beta_{p_1}(\alpha))} \xrightarrow{\bar{c_{out}}(\alpha'_1),\cdots,\bar{c_{out}}(\alpha'_{m'})} E_2, Y'_w, \emptyset, R''_j\|\theta^2_j\|\{z_j \mapsto \beta_{p_1}(\alpha)\}\|\{Y'_w \mapsto \bar{\gamma}'\}$
where m' is maximal.

or, if $X_{w,1} \neq \emptyset$:

– $P_1 = (\nu\bar{n}_1\nu\bar{x}_1)Q_1\|\theta^1_1\|\cdots\|Q_N\|\theta^1_N\|S_1$ where $Q_i \equiv [\Phi_1]Q'_i$
– $E_1, X_{w,1}, X_{c,1}, Q_i \xrightarrow{eq(\alpha)} \xrightarrow{\bar{c_{out}}(\alpha_1),\cdots,\bar{c_{out}}(\alpha_m)} E_1 \cup \{\alpha\}, Z_w, Z_c, Q''_i\|\theta^1_i\|\{Y_w\tilde{\gamma}\}$
where m is maximal.
– $\phi(T_P, p_1.\alpha) = \nu\bar{n}'_1.\mathcal{L}_1 \cdot (\alpha, \bar{\gamma}, \alpha_1, \cdots, \alpha_m)$
– $P_2 = (\nu\bar{n}_2\nu\bar{x}_2)R_1\|\theta^2_1\|\cdots\|R_N\|\theta^2_N\|S_2$ where $R_j \equiv [\Phi_2]R'_j$
– $E_2, X_{w,2}, X_{c,2}, R_j \xrightarrow{eq(\beta_{p_1}(\alpha))} \xrightarrow{\bar{c_{out}}(\alpha'_1),\cdots,\bar{c_{out}}(\alpha'_{m'})} E_2 \cup \{\beta_{p_1}(\alpha)\}, Z'_w, Z'_c, Q''_i\|\theta^2_j\|\{Y'_w\tilde{\gamma}'\}$
where m' is maximal.

We first build a process I_{p_1} with free variables $\tilde{x} = (z_1, \cdots, z_M)$ where $(z_1, \cdots, z_M)$ is the range of the canonical subsitution associated with the list $\mathcal{L}_1$ (and $\mathcal{L}_2$), such that

$$E_1, \emptyset, \emptyset, I_{p_1}\|(\nu\bar{n}'_1\mathcal{L}_1)\|P_1 \xrightarrow{*} \bar{c}(a).I_1$$

while for any I_2

$$E_2, , \emptyset, \emptyset, I_{p_1}\|(\nu\bar{n}'_2\mathcal{L}_2)\|P_2 \not\xrightarrow{*} \bar{c}(a).I_2$$

or its converse.

Let us write $Y_w = \{y_1, \cdots, y_q\}$ and $Y'_w = \{y_1, \cdots, y_{q'}\}$

Since $\Phi(T_P, p_1.\alpha), E(T_P, p_1.\alpha) \not\sim \Phi(T_P, p_2.\beta_{p_1}(\alpha)), E(T_P, p_2.\beta_{p_1}(\alpha))$, either $m \neq m'$, or $q \neq q'$ or there is a test distinguishing the two frames.

Let us assume that $m > m'$ (possibly exchanging the processes)

$$I_{p_1} = \begin{cases} \nu\bar{x}.\bar{c_{in}}(t_\alpha).c_{out}(x_1).\cdots.c_{out}(x_m).\bar{c}(a) \text{ if } \alpha \text{ is a term} \\ \nu\bar{x}.eq(t_\alpha = u_\alpha).c_{out}(x_1).\cdots.c_{out}(x_m).\bar{c}(a) \text{ if } \alpha \text{ is an equation} \end{cases}$$

such that $\mathrm{fn}(I_{p_1}) \cap \bar{n}'_1 = \emptyset$ following from $\mathrm{fn}(t_\alpha, u_\alpha) \cap \bar{n}'_1 = \emptyset$.

Let us now assume $q' > q$ (possibly exchanging the processes)

$$I_{p_1} = \begin{cases} \nu\bar{x}.\bar{c_{in}}(t_\alpha).[\mathrm{M}(x_1) \wedge \cdots \wedge \mathrm{M}(x_m)]\bar{c}(a) \text{ if } \alpha \text{ is a term} \\ \nu\bar{x}.eq(t_\alpha = u_\alpha).[\mathrm{M}(x_1) \wedge \cdots \wedge \mathrm{M}(x_m)]\bar{c}(a) \text{ if } \alpha \text{ is an equation} \end{cases}$$

such that $\text{fn}(I_{p_1}) \cap \bar{n}'_1 = \emptyset$ following from $\text{fn}(t_\alpha, u_\alpha) \cap \bar{n}'_1 = \emptyset$.

If $m = m'$ and $q = q'$, there are term $u_1, \cdots, u_n$ such that $\text{fn}(u_i) \cap (\bar{n}'_1 \cup \bar{n}'_2) = \emptyset$ and a predicate symbol p such that $\sigma_1, E_1 \vDash p(u_1, \cdots, u_n)$ while $\sigma_2, E_2 \nvDash p(u_1, \cdots, u_n)$ (possibly exchanging the processes), if σ_1, σ_2 are the canonical substitutions associated with lists $\mathcal{L}_1.(\alpha, \alpha_1, \cdots, \alpha_m)$ and $\mathcal{L}_2.(\beta_{p_1}(\alpha), \alpha'_1, \cdots, \alpha'_m)$. We build I_{p_1} as follows (we will only give the case where α is a term, the equation case is similar), using p to distinguish P_1 and P_2

$$I_{p_1} = (\nu \bar{x}) c_{in}(t_\alpha).c_{out}(z_{M+q+1}).\cdots.c_{out}(z_{M+q+m}).[p(u_1, \cdots, u_n)]\bar{c}(a)$$

We have distinguished the processes at positions p_1 and p_2, we now need to move up along the paths p_1, p_2 in order to distinguish the processes P and Q. For every prefix q of p_1, let

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

We then build, by backward induction on q a context I_q that distinguishes between

$$E_{1,q}, P_{1,q} \| \nu\overline{n'_{1,q}}.\mathcal{L}_{1,q} \text{ and } E_{2,q}, P_{2,q} \| \nu\overline{n'_{2,q}}.\mathcal{L}_{2,q}$$

Let $q.\alpha$ be a length $n+1$ prefix of p. We will only show the case where α is an equation $(\alpha_1 = \alpha_2)$ as the case where α is a term is very similar. Assume $\mathcal{L}_{1,q.\alpha} = \mathcal{L}_{1,q}.(\alpha, \alpha_1, \cdots, \alpha_m)$ and $\mathcal{L}_{1,q.\alpha} = \mathcal{L}_{1,q}.(\beta_q(\alpha), \alpha'_1, \cdots, \alpha'_m)$ and let σ_1^q, σ_2^q be the two substitutions associated with $\mathcal{L}_{1,q}, \mathcal{L}_{2,q}$. Let k be the length of $\mathcal{L}_{1,q}$ (which is the length of $\mathcal{L}_{2,q}$). There are two terms u_α and t_α such that $\text{fn}(t_\alpha, u_\alpha) \cap (\overline{n'_{1,q}} \cup \overline{n'_{2,q}}) = \emptyset$ and $E_{1,q} \vDash t_\alpha \sigma_1^q = \alpha_1 \wedge u_\alpha \sigma_1^q = \alpha_2$ and $E_{2,q} \vDash t_\alpha \sigma_2^q = \beta_q(\alpha_1) \wedge u_\alpha \sigma_2^q = \beta_q(\alpha_2)$. Let $(x_1, \cdots, x_l)$ be the subsequence of variables in $\mathcal{V}_1$ of $(z_{k+1}, \cdots, z_{k+m})$. Let us now define I_q

$$I_q = eq(t_\alpha = u_\alpha).c_{out}(x_{k+1}).\cdots.c_{out}(x_{k+m}).I_{q.\alpha}$$

We then have :

$$E_{1,q}, I_q \| P_{1,q} \| \nu(\overline{n'_{1,q}})\sigma_q^1 \xrightarrow{*} E_{1,q.\alpha}, I_q \| P_{1,q.\alpha} \| (\nu\overline{n'_{1,q.\alpha}})\sigma_{q.\alpha}^1$$

$$E_{2,B(q)}, I_q \| P_{2,B(q)} \| \nu(\overline{n'_{2,B(q)}})\sigma_{B(q)}^2 \xrightarrow{*} E_{2,B(q.\alpha)}, I_q \| P_{2,B(q.\alpha)} \| (\nu\overline{n'_{2,B(q.\alpha)}})\sigma_{B(q.\alpha)}^1$$

We conclude by applying the induction hypothesis.

Applying this to $q = \epsilon$, we get a process I which distinguishes between P and Q.

D.4 Computational trees

We also need to define a computational counterpart to the trees. We first need to define a parsing function as follows :

Definition 18. *Let us define a (stateful) parsing function $\kappa_\eta^\tau : \{0,1\}^\eta \to \mathcal{T}$ as follows : take $w \in \{0,1\}^\eta$, let K_1 be the set of honestly generated keys, and H be the set of already known bitstring-term associations. We apply the following rules with in order.*

- *If $H(w)$ is defined, $\kappa_\eta^\tau(w) = H(w)$*
- *If w is the computational counterpart of new_j then $\kappa_\eta^\tau(w) = new_j$*
- *If w is the pair $w_1 \| w_2$ then $\kappa_\eta^\tau(w) = \langle \kappa_\eta^\tau(w_1), \kappa_\eta^\tau(w_2) \rangle$*
- *If there exists a $k \in K_1$ such that $\mathcal{D}(w, [\![k]\!]_\eta^\tau) = w_1$ with $w_1 \neq \bot$ (we consider the smallest such key for the key order) we create a fresh name r and $\kappa_\eta^\tau(w) = \{\kappa_\eta^\tau(w_1)\}_k^r$*
- *If w is a key, we take a fresh name $k \in \mathcal{K}_2$ and $\kappa_\eta^\tau(w) = k$*
- *If no previous rule apply we take a fresh name u and $\kappa_\eta^\tau(w) = u$*

Each time we do an association between a term and its implementation, by building the computational counterpart of a term or by parsing a bitstring, we record it in H.

Let us note that our parsing function verifies the following property : if u is a subterm of $\kappa_\eta^\tau(m)$ for some bitstring m, then $\kappa_\eta^\tau([\![u]\!]_\eta^\tau) = u$.

We now have to define how the tree oracle $\mathcal{O}_T$ answers to a request w of a computational attacker when the current position is p.

1. First of all we compute $t = \kappa_\eta^\tau(w)$, we go down in the tree at position $p.t$, if there is no such position $\mathcal{O}_T$ outputs 0.
2. If T is waiting for equations, we chose one, we test its computational validity and go down in T a the position corresponding to the result of this test (as we also record disequations). We loop at point 2 as long as T waits for equations.
3. When T is not waiting for equations, let q be the position of T at this point. Let us write $\phi_1 = \phi(T, p)$, $\phi_2 = \phi(T, q)$. $\mathcal{O}_T$ outputs $[\![x\phi_2]\!]_\eta^\tau$ for all $x \in (\mathrm{dom}(\phi_2) \setminus \mathrm{dom}(\phi_1)) \cap \mathcal{V}_1$

An important remark is that the order in which equations are added is not important as we will have to go through the whole set of equations before outputting anything.

E Suppressing encryption

We start be replacing all honest cyphertexts by encryptions of zeros. We will follow the key hierarchy, starting from a maximal key. The function Ψ_k replaces the plaintexts under a k encryption by a constant O^l of same length.

$$\begin{aligned}
\psi_k(n) &= n \text{ if } n \text{ is a name or a constant} \\
\psi_k(\langle t_1, t_2 \rangle) &= \langle \psi_k(t_1), \psi_k(t_2) \rangle \\
\psi_k(\{t\}_k^r) &= \{0^{l(t)}\}_k^r \\
\psi_k(\{t\}_{k'}^r) &= \{\psi_k(t)\}_{k'}^r \text{ if } k \neq k' \\
\psi_k(\pi_i(t)) &= \pi_i(\psi_k(t)) \\
\psi_k(\mathrm{dec}(t, u)) &= \mathrm{dec}(\psi_k(t), \psi_k(u)) \text{ if } u \neq k \text{ and undefined otherwise}
\end{aligned}$$

As the randoms from encryptions of two different term are different, ψ_k is injective. We extend ψ_k to trees by applying it at each term in the positions, frames and equation sets.

We now need to show that if ϕ is a frame and E a correct equation set for ϕ, such that $\phi \not\vdash_{E\cup E_0} k$, and u is such that $\phi \vdash_{E\cup E_0} u$, then $\psi_k(u{\downarrow}_{E_0\cup E}) = \psi_k(u){\downarrow}_{E_0\cup\psi_k(E)}$ in order to prove the results we need. This result is proven in the next subsection.

E.1 Commutation lemma

In the following part, we will always assume that the frames are built of ground terms without destructors with respect to the equational theory.

Let k be a secret key and u a deducible term. We want to show that $\psi_k(u{\downarrow}_E) = \psi_k(u){\downarrow}_{\psi_k(E)}$ for some oublicly generated set of equations E.

Lemma 3. *Let k, φ such that $\varphi \not\vdash k$, we assume that φ is in normal form. Let R be a public context.*

$$\psi_k(R\varphi{\downarrow}_{E_0}) = R\psi_k(\varphi){\downarrow}_{E_0}$$

Proof. This result is proven in [CLC08b]

Lemma 4. *Let k, φ be such that $\varphi \not\vdash k$. Let E be a well-formed equation set. If t is a subterm of ϕ, then t is in normal form and $\psi_k(t)$ is in normal form with respect to $\psi_k(E)$*

Proof. The first part is true by the fact that the frame is supposed to be ground with respect to E.

Let us prove the second part by induction on the term. If t is a name the result is trivial. If $t = \langle t_1, t_2\rangle$ as, by induction hypothesis, $\psi_k(t_1)$ and $\psi_k(t_2)$ are ground the form of the equations ensures that t is ground. For the same reason $\psi_k(\{t_1\}_k)$ with k honest is ground. Let us now consider the case $\{t_1\}_k$ with k dishonest. As $\psi_k(t_1)$ is ground by induction hypothesis, if $\psi_k(t)$ is not ground, then there exists $\psi_k(e_1) = \psi_k(e_2)$ in $\psi_k(E)$ such that $\psi_k(t) = \psi_k(e_i)$ for some $i \in \{1, 2\}$. As ψ_k is bijective then $t = e_i$ which contradicts the fact that t is ground.

Lemma 5. *Let ϕ be a frame in normal form with respect to E a publicly generated equation set. If u is a deducible term, t is a subterm of u, t is a subterm of ϕ or t is deducible. If t is directly under a destructor it is deducible.*

Proof. Let us prove the result by induction on the size of E. If E is of size 0, the trivially holds by a simple induction on the length of the E_0 reduction, as E_0 is subterm convergent. If E is of size n, let $e_1 = e_2$ be the last equation of E, as e_1 and e_2 are deducible with $E\backslash\{e_1 = e_2\}$ (the last equation is deducible with the previous equation set, and as the equations are well formed), the result holds on e_1 and e_2. We will write $E = \{e_1 = e_2\} \cup E'$.

Let u be a deducible term, let R be a public term such that $R \rightarrow^*_{E\cup E_0} u$, let us prove our result by induction on the length of the reduction. If the reduction

is of size 0, the result trivially holds. Let us assume the result for any such reduction of size at most m. Let $R \rightarrow^{m}_{E \cup E_0} u' \rightarrow_{E \cup E_0} u$. The result holds on u' by induction hypothesis. Let $e'_1 = e'_2$ be the equation such that $u' = C[e'_1]$ and $u = C[e'_2]$, let t be a subterm of u. If t is a subterm of e'_2 and $e'_1 = e'_2$ is in E the result holds by induction hypothesis if it is a strict subterm of e'_2 it cannot be under a destructor as the second member of equations cannot contain destructors, and if it is e'_2, it is deducible. If t is a subterm of e'_2 and $e'_1 = e'_2$ is in E_0 it is also a subterm of u' and the result holds by induction hypothesis (let us note that if it is directly under a destructor, it already was in u' as E_0 only deletes destructors). The last case is $t = C'[e'_2]$. As the frame is in normal form, $t' = C'[e'_1]$ is not a subterm of the frame. As $C'[e_1]$ is a subterm of u', it is deducible, and then t is also deducible.

Lemma 6. *Let k, φ be such that $\varphi \not\vdash k$. Let E be a well-formed equation set. Let u be a deducible term. We have*

$$u \rightarrow_E v \implies \psi_k(u) \xrightarrow{=}_{\psi_k(E)} \psi_k(v)$$

Proof. We prove the result by induction on the length of u. If u is a name of a variable, the result trivially holds. Let now $u \rightarrow_E u'$ with u of size more than 1.

- If $u = \langle u_1, u_2 \rangle$ then the reduction is a reduction of u_1 or u_2 and as u_1 and u_2 are deducible, the induction hypothesis gives the result.
- If $u = \{u_1\}^r_{u_2}$ where u_2 is not a key. Let us assume that the reduction takes place in u_i, then as the frame is in normal form, u_i is not a subterm of the frame, it is then deducible (previous lemma) and we apply the induction hypothesis.
- If $u = \{u_1\}^r_{k'}$ where k' is honest, with $k \neq k'$, the reduction takes place in u_1. As the frame is in normal form, u_1 is not a subterm of the frame, it is then deducible (previous lemma) and we apply the induction hypothesis. The redex can not be the entire term because of the orientation of our equations.
- If $u = \{u_1\}^r_{k'}$ where k' is dishonest if the redex is u, then $\psi_k(u)$ is reducible by $\psi_k(E)$, otherwise we apply the induction hypothesis as previously.
- If $u = \{u_1\}^r_k$, the reduction takes place in u_1.As the equations preserve length we have $\psi_k(u') = \{0^{l(u_1)}\}^r_k = \psi_k(u)$.
- If $u = \pi_i(u_1)$ if $u_1 = \langle t_1, t_2 \rangle$, if the reduction is the E_0 reduction at top level, then it can be matched, otherwise the reduction takes place in u_1 and as previously the result holds as u_1 is deducible by the previous lemma.
- If $u = \mathrm{dec}(u_1, k)$, k is dishonest since directly under a destructor (previous lemma), if the reduction does not take place at top level, we apply the induction hypothesis as previously. If it takes place at top level the result holds.

Corollary 1. *Let ϕ be a frame in normal form, E be a publicly generated equation set under $\psi_k(\phi)$ and k a honest key such that $\phi \not\vdash_E k$. If $\phi \vdash_E u$ then $\psi_k(\phi) \vdash_{\psi_k(E)} \psi_k(u)$*

Proof. Let R be a public term such that $R\phi \rightarrow^*_E u$ with the previous lemma we have

$$\psi_k(R\phi) \rightarrow^*_{\psi_k(E)} \psi_k(u)$$

and as R is public and thus does not contain k

$$R\psi_k(\phi) \rightarrow^*_{\psi_k(E)} \psi_k(u)$$

Lemma 7. *If u is deducible and in normal form with respect to E (publicly generated equation set) and ϕ a frame in normal form, then $\psi_k(u)$ is in normal form with respect to $\psi_k(E)$*

Proof. We prove the result by induction on the length of u. If u is a name or a variable the result trivially holds.

- If $u = \langle u_1, u_2 \rangle$. Then u_1 and u_2 are in normal form and so is $\psi_k(u) = \langle \psi_k(u_1), \psi_k(u_2) \rangle$ by induction hypothesis.
- If $u = \{u_1\}^r_{u_2}$ where u_2 is not a key.The same argument holds.
- If $u = \{u_1\}^r_{k'}$ with $k \neq k'$, u_1 is either a subterm of the frame or deducible. In the first case we apply the lemma 4, in the second we use the induction hypothesis.
- If $u = \{u_1\}^r_k$, we have $\psi_k(u) = \{0^{l(u_1)}\}^r_k$ which is in normal form.
- If $u = \pi_i(u_1)$ as u_1 is deducible by lemma 5, we can apply our induction hypothesis.
- If $u = \mathrm{dec}(u_1, u_2)$, u_1 and u_2 are deducible (lemma 5), and we apply the induction hypothesis.

We then have as a corollary

Corollary 2. *Let k, φ such that $\varphi \not\vdash k$. Let E be well-formed wrt ϕ Let u be a deducible term.*

$$\psi_k(u{\downarrow_E}) = \psi_k(u){\downarrow_{\psi_k(E)}}$$

E.2 Equivalence under ψ_k

Lemma 8. *For every execution tree T, and for every key k not deductible from a frame of T, $T \sim \psi_k(T)$*

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 ϕ such that $\phi \not\vdash_E k$, we have $\phi \sim \psi_k(\phi)$. For any deducible term u, $\psi_k(u{\downarrow_E}) = \psi_k(u){\downarrow_{\psi_k(E)}}$ by corollary 2. Moreover, the predicates M, EQ, EL, $\mathrm{P}_{\mathrm{samekey}}$ are stable by replacement of zeros under encryption: $E \vDash P(t_1, t_2)[\{u\}^r_k \rightarrow \{0^{l(u)}\}^r_k] \Leftrightarrow P(t_1, t_2)$ This yields the desired property.

We now need to prove that once all plaintexts under honest encryption have been replaced by zeros, equivalence is exactly equality up to renaming.

Lemma 9. *Let ϕ_1, ϕ_2 be two frames, such that for every 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 $\mathrm{P}_{\mathrm{samekey}}$ and EL are in the predicate set, then $\phi_1 \sim \phi_2$ iff ϕ_1 and ϕ_2 are equal up to renaming.*

This lemma is already proven in [CLC08b], and their proof holds in our model.

F Computational soundness of tree equivalence

Let us define what it means for two trees to be computationally indistinguishable.

Definition 19. *We say that two symbolic trees T_1, T_2 are computationally indistinguishable (written $T_1 \approx T_2$) if, for all PPT $\mathcal{A}$:*

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

The computational soundness result we want is as follows :

Lemma 10. *If $T_{P_1} \sim T_{P_2}$ and for $i = 1, 2$, T_{P_i} admits a key hierarchy, then, if the cryptographic scheme is jointly IND-CPA and INT-CTXT, we have $T_{P_1} \approx T_{P_2}$*

Proof. We will prove the result using a hybrid argument. Let us write $T_1 := T_{P_1}$ and $T_2 := T_{P_2}$. Assume that there is a PPT $\mathcal{A}$ distinguishing T_1 and T_2 with a non negligible probability. $\mathcal{A}$ does only a finite number of requests new_j, and thus forces the generation of a polynomial number of honest keys. More precisely, when $\mathcal{A}$ interacts with T_i, it does request until a depth of at most $p(\eta)$ where p is a polynomial. Let us write $T/_n$ the execution tree obtained from T by removing all nodes at depth more than n. Let s_i be the number of replicated processes in P_i, $m_i^1, \cdots, m_i^{s_i}$ be the numbers of secret keys generated in $S_i^1, \cdots, S_i^{s_i}$ and m_i^0 be the number of secret keys in P_i without the replicated processes. We now know that $T_i/_{p(\eta)}$ contains at most $n_i := p(\eta).(\sum_{j=1}^{s_i} m_i^j) + m_i^0$ secret keys. This allows us to transform our tree key after key, showing each step is distinguishable with negligible probability, and concluding that the first and last steps are distinguishable with only negligible probability.

Let $<_i$ be an order on the keys of $T_i/_{p(\eta)}$. Let $k_i^1, \cdots, k_i^{n_i}$ be the honest keys of $T_i/_{p(\eta)}$ ordered such that $k_i^1 <_i k_i^2 <_i \cdots <_i k_i^{n_i}$. Let us write $T_i^1 = T_i/_{p(\eta)}$ and $T_i^{j+1} = \psi_{k_i^j}(T_j)$ for $j = 1..n_i$. The lemma 8 gives us $T_i^{j+1} \sim T_i^j$. Transitivity gives $T_1^{n_1} \sim T_2^{n_2}$. With these results : either $\mathcal{A}$ distinguishes T_i^j and T_i^{j+1} with non negligible probability for some i, j, or $\mathcal{A}$ distinguishes $T_1^{n_1}$ et $T_2^{n_2}$.

Let us start with the case in which $\mathcal{A}$ distinguishes T_i^j and T_i^{j+1} with non negligible probability. We show that in that case we break the IND-CPA or INT-CTXT game on key k_i^j. Without loss of generality, we will assume $i = 1$. Let us build a PPT B simulating $\mathcal{O}_{T_1^j}$ or $\mathcal{O}_{T_1^{j+1}}$ in polynomial time, using the IND-CPA and INT-CTXT oracles for the implementation of the key k_1^j.

B behaves exactly as $\mathcal{O}_{T_1^j}$ apart from the fact that it does not pick k_i^j but uses the IND-CPA oracle to encrypt and the INT-CTXT oracle to decrypt with this key. It outputs b if $\mathcal{A}$ stop and outputs b. $B^{\mathcal{O}_{k_1^j}}$ behaves as $\mathcal{A}^{\mathcal{O}_{T_1^j}}$ and $B^{\mathcal{O}'_{k_1^j}}$ behaves as $\mathcal{A}^{\mathcal{O}_{T_1^{j+1}}}$ apart from the cases in which we break INT-CTXT (the cases in which we succeed at decrypting a non already encrypted term). This means that B breaks joint IND-CPA, INT-CTXT when $\mathcal{A}$ breaks $T_1^j \approx T_1^{j+1}$.

Also B runs in polynomial time as simulating the protocol is polynomial, parsing is polynomial, and there is only a polynomial number of equations to test.

The case left is $\mathcal{A}$ distinguishes $T_1^{n_1}$ and $T_2^{n_2}$ with non negligible probability. Let β_1 be the bijection from $T_1^{n_1} \to T_2^{n_2}$ and β_2 be the bijection from $T_1^{n_2} \to T_2^{n_1}$. For every position p in $T_1^{n_1}$ the frame $\phi(T_1^{n_1}, p)$ is equivalent to $\phi(T_1^{n_2}, \beta(p))$. By lemma 9, the two frames are equals up to renaming. As the two frames contain the requests from the adversary at the same places, β_1 and β_2 are the identical functions (up to alpha renaming). As $T_1^{n_1} =_\alpha T_2^{n_2}$ are equals, they are indistinguishable by $\mathcal{A}$.

G Trace mapping

We now know from lemmas 2 and 10 that if $P \sim_o Q$ then $T_P \approx T_Q$. We now need to show that if $T_P \approx T_Q$ then $P \approx Q$. We will do this in two steps, the first one being the classification of the traces of P which are not abstracted in T_P, and the second one being the fact that under our cryptographic hypotheses, these cases can not occur with non negligible probability.

Let us start by defining the sequence of the messages exchanged between P and $\mathcal{A}$, and what it means for such a trace to be fully abstracted in the tree T_P. In order to do that note that given τ and η, the behaviour of $[\![P]\!]_\eta^\tau \| \mathcal{A}_\tau$ is deterministic.

Definition 20. Messages$(P, \eta, \tau, \mathcal{A})$ *is the sequence* a_n *defined as follows. Let* $\gamma_n^{\mathcal{A}}$ *be the sequence of configurations of* $\mathcal{A}$*, along its deterministic computation, on the random* τ*, interacting with* $[\![P]\!]_\eta^\tau$*. Let* $\delta_n^{\mathcal{A}}$ *be the subsequence of configurations following a new, send, or receive action of* $\gamma_n^{\mathcal{A}}$*.*

- *If* $\delta_n^{\mathcal{A}}$ *follows a new action, let* j *be the content of the control tape in* $\delta_n^{\mathcal{A}}$ *and* l *be the content of the receive tape in the same configuration. Then* $a_n = s(new_j).r(l)$*.*
- *If* $\delta_n^{\mathcal{A}}$ *follows a send action, we let* $a_n = s(\langle i, m \rangle)$ *where* i *(resp.* m*) is the content of the control (resp. send) tape in* $\delta_n^{\mathcal{A}}$*.*
- *If* $\delta_n^{\mathcal{A}}$ *follows a receive action, then* $a_n = r(m)$ *where* m *is the content of the receive tape in* $\delta_n^{\mathcal{A}}$

Lemma 11. *We can assume without loss of generality that* Messages$(P, \eta, \tau, \mathcal{A})$ *is a sequence of the form* $s(m_1).R_1.s(m_2).R_2 \cdots s(m_n), R_n$ *where each* R_i *is a sequence* $r(m'_1).\cdots.r(m'_{n_i})$*.*

In that case we can represent more elegantly Messages$(P, \eta, \tau, \mathcal{A})$ 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 : $L_{i+1} = L_i.R_i.m_i$. We then write SMessages$(P, \eta, \tau, \mathcal{A}) = (m_1, \cdots, m_n)$ the subsequence of messages sent by $\mathcal{A}$.

In order to bind more closely the computational processes to symbolic processes let us write γ_i the computational configuration of the network before the

i^{th} action *send* of $\mathcal{A}$, this allows us to describe the execution of $\mathcal{A}_\tau \| [\![P]\!]^\tau_\eta$ as follows :

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

Definition 21 (Full abstraction). *Let $\gamma_1, L_1 \xrightarrow{m_1} \gamma_2, L_2 \cdots \xrightarrow{m_n} \gamma_n, L_n$ be an execution, p be a position of the tree T. Let us write $p = \alpha_1.\cdots.\alpha_m$. Let $\alpha_{n_1} \cdots \alpha_{n_s}$ be the subsequence of p which are not equations.*

We say that p fully abstracts $\gamma_1, L_1 \xrightarrow{m_1} \gamma_2, L_2 \cdots \xrightarrow{m_n} \gamma_n, L_n$ if $s = n$ and $\forall j \leq n$

- $[\![\alpha_j]\!]^\tau_\eta = m_j$
- *If* $T(\alpha_1 \cdots \alpha_{n_j}) = (Q_j, \phi_j, E_j)$ *then*
  - $[\![Q_j]\!]^\tau_\eta = \gamma_j$
  - $[\![\phi_j \cap \{x \mapsto t | t \in \mathcal{T}, x \in \mathcal{V}_1\}]\!]^\tau_\eta = L_j$
  - $\forall (s = t) \in E_j, [\![s]\!]^\tau_\eta = [\![t]\!]^\tau_\eta$
  - $\forall (s \neq t) \in E_j, [\![s]\!]^\tau_\eta \neq [\![t]\!]^\tau_\eta$

We now want to prove that it is impossible for the attacker to produce a trace which is not fully abstracted with non negligible probability. First of all we need to classify the traces that can not be fully abstracted.

Lemma 12. *Let P be a simple process, T_P its execution tree. Given a security parameter η, a random τ and an attacker $\mathcal{A}$, let Γ be the execution of $\mathcal{A}_\tau \| [\![P]\!]^\tau_\eta$ defined as previously. There is a path p in $\mathrm{Pos}(T_P)$ such that one of the following properties holds :*

1. *p fully abstracts Γ*
2. *$T_P(p) = (Q, \phi, E)$ and there is a transition $\gamma_n, L_n \xrightarrow{m} \gamma_{n+1}, L_{n+1}$ in Γ such that p fully abstracts the prefix of Γ ending at γ_n, L_n and one of the following conditions holds.*
3. *$\phi \not\vdash \kappa^\tau_\eta(m)$.*
4. *$m = \langle l_i, m' \rangle$, there exists a process P_i with pid l_i in Q and $P_i = c_{in}(x) \cdot [R]$ and*
   - *$\exists \mathrm{M}(u) \in \Phi$ such that $\mathrm{dec}(s,t) \in St(u)$ with $t{\downarrow_E} = k$ and $k \in \mathcal{K}_2$*
   - *$E \vDash \mathrm{M}(s) \wedge \neg \mathrm{M}(\mathrm{dec}(s,k))$*
   - *$\mathcal{D}([\![s]\!]^\tau_\eta, [\![k]\!]^\tau_\eta) = m''$ with $\phi, s \not\vdash \kappa^\tau_\eta(m'')$*
5. *$m = \langle l_i, m' \rangle$, there exists a process P_i with pid l_i in Q such that $P_i = c_{in}(x) \cdot [\Phi] R$ and*
   - *$\exists \mathrm{EQ}(u,v) \in \Phi$ such that $u{\downarrow_E} = C[u_1, \cdots, u_n], v{\downarrow_E} = C[v_1, \cdots, v_n]$*
   - *$[\![u]\!]^\tau_\eta = [\![v]\!]^\tau_\eta$*
   - *for every $i \in \{1, \cdots, n\}$ there exists s_i et $k_i \in \mathcal{K}$ such that*
     - *either $u_i = \{s_i\}^{r_i}_{k_i}$. We let $t_i := v_i$.*
     - *either $v_i = \{s_i\}^{r_i}_{k_i}$. We let $t_i := u_i$.*
   - *for every $i \in \{1, \cdots, n\}$, $E \cup E_0 \vDash \mathrm{M}(s_i) \wedge \neg \mathrm{EQ}(u_i, v_i)$.*
   - *$\exists i \in \{1 \cdots n\}$ such that $\phi, s_i \not\vdash t_i$ and $k_i \in \mathcal{K}_2$.*
6. *$m = \langle l_i, m' \rangle$, there exists a process P_i with pid l_i in Q such that $P_i = c_{in}(x) \cdot [\Phi] R$ and*

- $\exists \mathrm{M}(u) \in \Phi$ *such that* $\mathrm{dec}(s,t) \in St(u)$ *with* $t{\downarrow_E}$ *et* $k \in \mathcal{K}_1$.
- $E \vDash \mathrm{M}(s)$
- $\mathrm{dec}(s{\downarrow_E}, k)$ *is in normal form.*
- $\mathcal{D}([\![s]\!]^\tau_\eta, [\![k]\!]^\tau_\eta) \neq \bot$ *and* k *is honest.*

7. $m = \langle l_i, m' \rangle$, *there exists a process* P_i *with pid* l_i *in* Q *such that* $P_i = c_{in}(x) \cdot [\Phi]R$ *and*
   - $\exists \mathrm{EQ}(u,v) \in \Phi$ *such that* $u{\downarrow_E} = C[u_1, \cdots, u_n], v{\downarrow_E} = C[v_1, \cdots, v_n]$
   - *for every* $i \in \{1, \cdots, n\}$ *there exists* s_i *and* $k_i \in \mathcal{K}$ *such that*
     - *either* $u_i = \{s_i\}^{r_i}_{k_i}$. *We let* $t_i := v_i$.
     - *either* $v_i = \{s_i\}^{r_i}_{k_i}$. *We let* $t_i := u_i$.
   - *for each* $i \in \{1, \cdots, n\}$, $E \vDash \mathrm{M}(s_i) \wedge \mathrm{M}(t_i)$
   - $\exists i \in \{1 \cdots n\}$ *such that* $\mathrm{dec}(t_i, k)$ *is in normal form and* $[\![\{s_i\}^{r_i}_{k_i}]\!]^\tau_\eta = [\![t_i]\!]^\tau_\eta$ *et* $k_i \in \mathcal{K}_1$ *et* $t_i = \{\cdot\}^{\cdot}_k$ *with* $k \neq k_i$ *et* $k \in \mathcal{K}_1$.
8. $m = \langle l_i, m' \rangle$, *there exists a process* P_i *with pid* l_i *in* Q *such that* $P_i = c_{in}(x) \cdot [\Phi]R$ *and*
   - $\exists \mathrm{EQ}(u,v) \in \Phi$ *such that* $u{\downarrow_E} = C[u_1, \cdots, u_n], v{\downarrow_E} = C[v_1, \cdots, v_n]$
   - *for each* $i \in \{1, \cdots, n\}$ *there exists* s_i *and* $k_i \in \mathcal{K}$ *such that*
     - *either* $u_i = \{s_i\}^{r_i}_{k_i}$. *We let* $t_i := v_i$.
     - *either* $v_i = \{s_i\}^{r_i}_{k_i}$. *We let* $t_i := u_i$.
   - *for all* $i \in \{1, \cdots, n\}$, $E \vDash \mathrm{M}(s_i) \wedge \mathrm{M}(t_i)$
   - $\exists i \in \{1 \cdots n\}$ *such that* $t_i = \{s_i\}^{r}_{k_i}$, *and* $r \neq r_i$ *with* $k_i \in \mathcal{K}_1$.
9. *There exists two honest names* s *and* t, *honestly generated such that* $[\![s]\!]^\tau_\eta = [\![t]\!]^\tau_\eta$
10. *There exists* s *honest name and* t *dishonest name such that* t *was in* H *before the generation of* s *and* $[\![s]\!]^\tau_\eta = [\![t]\!]^\tau_\eta$.
11. *There exists* s *honest encryption and* t *dishonest name such that* t *was in* H *before the generation of* s *and* $[\![s]\!]^\tau_\eta = [\![t]\!]^\tau_\eta$.

Proof. It is clear that there are only two cases in which a trace can not can not be completely abstracted :

- The case in which the tree receives a bitstring m such that $\kappa^\tau_\eta(m)$ would be accepted by a process but does not exist in the tree. In that case $\kappa^\tau_\eta(m)$ is non deducible and we fall in case 3.
- The case in which the term is accepted on both sides but the tree does not behave as the process. This means that there is a condition Φ in a subprocess such that Φ is computationally true but symbolically false and no well formed equations would make it symbolically true. The reminder of the proof deals with this case.

Let us start with the M predicate. Assume $E \vDash \neg \mathrm{M}(s[\bar{x}]\sigma)$ and $[\![\mathrm{M}(s[\bar{x}]\sigma)]\!]^\tau_\eta = 1$. Let us write $u = s[\bar{x}]\sigma$. The evaluation of $[\![\mathrm{M}(u)]\!]^\tau_\eta$ is the evaluation of $[\![u]\!]^\tau_\eta$. We know that $[\![u]\!]^\tau_\eta \neq \bot$. Let Po be the set of position p in u such that $E \vDash \neg \mathrm{M}(u_{|_p})$. $Po \neq \emptyset$, let p be maximal in Po. We are in one of the following cases :$u_{|_p} = \pi_i(t)$ or $u_{|_p} = \mathrm{dec}(t,v)$ with $E \vDash \mathrm{M}(t) \wedge \mathrm{M}(v)$ as $x \in \bar{x}$, $x\sigma$ and all its subterms are wellformed (they are without destructors), and p is maximal.

- If $u_{|_p}{\downarrow_E} = \pi_i(t)$, let us consider the following cases :

- If t is a pair, $E \vDash \mathrm{M}(\pi_i(t))$, this can not occur.
- If t is an encryption, the tagging hypotheses ensure that $[\![\pi_i(t)]\!]^\tau_\eta = \bot$ which can not be the case as $[\![u]\!]^\tau_\eta \neq \bot$.
- If t is a honest name, as previously $[\![\pi_i(t)]\!]^\tau_\eta = \bot$ which is impossible due to tagging hypotheses.
- If t is a dishonest name, and $[\![\pi_i(t)]\!]^\tau_\eta \neq \bot$, then $[\![t]\!]^\tau_\eta$ is a pair, and parsing ensures that this case is impossible.

– If $u_{|_p} = \mathrm{dec}(t, v)$, as $E \vDash \mathrm{M}(v)$ then $v{\downarrow_E} = k$ with $k \in \mathcal{K}$, otherwise, as previously we get a contradiction as a decryption succeeds only if it is done with a proper key and tagging ensures that only the elements of $\mathcal{K}$ are interpreted as keys.
  - If k is honest, then for every well formed u we have $E \nvDash \mathrm{dec}(t, k) = u$. As $\mathcal{D}([\![t]\!], [\![k]\!]) \neq \bot$, this case falls in case 6
  - If k is dishonest $\mathrm{dec}(t, k) = \bot$ and $\kappa^\tau_\eta(\mathcal{D}([\![t]\!], [\![k]\!])) = u$ and $\mathrm{dec}(t, k) = u$ is not a well formed equation (otherwise it would appear in the tree and thus it would be possible to abstract the equation),
    * The equation creates an equality of two name and we fall in case 9 or 10.
    * The equation creates an equality $\{t\}^r_k = \{t'\}^{r'}_{k'}$ with $k \neq k'$ or $r \neq r'$ and k and k' honest keys. We fall in case 7 or 8.
    * The equation create an equality $\{t\}^r_k = n$ with n dishonest name. The computational hypotheses ensures that n is dishonest and the parsing hypotheses ensure that n was created before $\{t\}^r_k$, we fall in case 11.
    * The second member of the equation is not deducible and we fall in case 4.

We easily check that the converse : $E \vDash \mathrm{M}(u)$ and $[\![u]\!]^\tau_\eta = \bot$ is impossible as all the equations added are computationally true (and thanks to the properties explained in the appendix A).

We will now deal with the EQ predicate. Take $\mathrm{EQ}(s_1[\bar{x}_1]\sigma, s_2[\bar{x}_2]\sigma)$, we assume w.l.o.g. that $\mathrm{M}(s_1[\bar{x}_1]\sigma)$ and $\mathrm{M}(s_2[\bar{x}_2]\sigma)$ are are in the condition and thus have already been dealt with, we can assume that $E \vDash \mathrm{M}(s_1[\bar{x}_1]\sigma) \wedge \mathrm{M}(s_2[\bar{x}_2]\sigma)$. In that case $s_1[\bar{x}_1]{\downarrow_E} := u_1$ and $s_2[\bar{x}_2]{\downarrow_E} := u_2$ are without destructors. We have $\mathrm{EQ}(s_1[\bar{x}_1]\sigma, s_2[\bar{x}_2]\sigma)$ is logically equivalent to $\mathrm{EQ}(u_1, u_2)$. Assume $E \nvDash \mathrm{EQ}(u_1, u_2)$ and $[\![u_1]\!]^\tau_\eta = [\![u_2]\!]^\tau_\eta$. Let us reason inductively on the first function symbol in u_1 and u_2.

– If u_1 is a honest name, then tagging hypotheses gives us that u_2 is a name. If u_2 is a honest name we fall in case 9. If u_2 is a dishonest name, the parsing hypotheses ensure that u_2 was in H before the creation of u_1 and we fall in case 10 with $C = [\cdot]$. The case where u_2 is honest and u_1 dishonest is treated exactly in the same way.
– If u_1 and u_2 are dishonest names, parsing ensure that $u_1 = u_2$. The cases where u_2 is not a name will be treated later.

– If u_1 is a pair, the tagging hypotheses ensures that u_2 is also a pair. Let us write $u_1 = \langle u_{11}, u_{12}\rangle$ and $u_2 = \langle u_{21}, u_{22}\rangle$. The computational hypotheses ensures that $[\![u_{11}]\!]^\tau_\eta = [\![u_{21}]\!]^\tau_\eta$ and $[\![u_{12}]\!]^\tau_\eta = [\![u_{22}]\!]^\tau_\eta$. We have $E \not\vdash \mathrm{EQ}(u_{11}, u_{21})$ or $E \not\vdash \mathrm{EQ}(u_{12}, u_{22})$, let us assume, w.l.o.g that we are in the first case. The induction hypothesis gives us a context C_1 corresponding to $\mathrm{EQ}(u_{11}, u_{21})$, if $\mathrm{EQ}(u_{12}, u_{22}) = \bot$ let C_2 be the corresponding context, otherwise we let $C_2 = u_{12}$. We let $C = \langle C_1, C_2\rangle$ and the failure case is the one of the induction on $\mathrm{EQ}(u_{11}, u_{21})$
– If u_1 is a dishonest encryption, and u_2 is not an encryption by the same dishonest key with the same random. In that case we have an invalid equation, otherwise it would appear in the tree an we could abstract the trace. The equation is not well formed and as the only cases in the definition of a well formed equation that would not contradict directly the computational hypotheses are the following cases :
  • The equation creates an equality of two name and we fall in case 9 or 10
  • The equation creates an equality $\{t\}^r_k = \{t'\}^{r'}_{k'}$ with $k \neq k'$ or $r \neq r'$ and k and k' honest keys. We fall in case 7 or 8.
  • The equation create an equality $\{t\}^r_k = n$ with n dishonest name. The computational hypotheses ensures that n is dishonest and the parsing hypotheses ensure that n was created before $\{t\}^r_k$, we fall in case 11.
  • The second member of the equation is not deducible and we fall in case 5

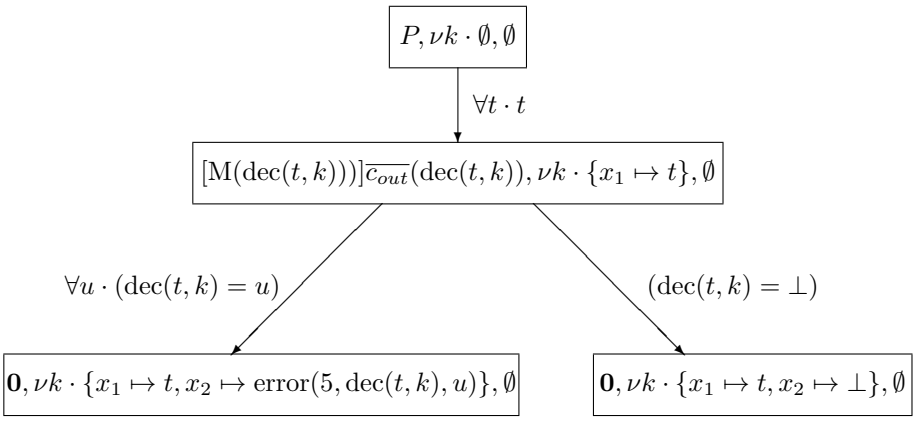
  In each of the previous cases, we pick $C = [\cdot]$.
– If u_1 is an encryption with a honest key k, and u_2 is not an encryption by k, if it is a dishonest we can apply the same arguments as in the previous case. If u_2 is a dishonest name the parsing hypotheses ensure that it was present in H before the encryption and we fall in case 11. The only case left is the case where u_2 is a honest encryption and we fall in case 7. We pick $C = [\cdot]$.
– If u_1 is a honest encryption by k and u_2 is an encryption by k with a different random we fall in case 8. In that case we pick $C = [\cdot]$
– If $u_1 = \{u'_1\}^r_k$ and $u_2 = \{u'_2\}^r_k$, we apply the induction hypothesis to $\mathrm{EQ}(u'_1, u'_2)$, let C' be the corresponding context. We take $C = \{C'\}^r_k$ and the failure case is the one of the induction.

We see that the cases in which the abstraction fails, apart from the case where the adversary gives a non deducible term, are very similar to the equations that can be "legally" added. We add these transitions in the tree in the same way as normal equations in order to build a new tree $\tilde{T}_P$. Let us explain a bit more what we mean by adding these equations in the tree. When we find a invalid symbolic equation, the tree answers the failure case and the involved terms. this allows us to treat invalid equations as we have treated valid equations, and then to use the tools we have on execution trees. This is the point which allows us to patch the proof of [CLC08b].

Example 11. Let us consider the following process :

$$P = (\nu k)c_{in}(x) \cdot \texttt{if } \mathrm{M}(\mathrm{dec}(x, k)) \texttt{ then } \overline{c_{out}}(x) \texttt{ else } \overline{c_{out}}(\bot)$$

its tree is as follows :



where the edge labelled by $\forall t \cdot t$ is in fact a multiple edge representing the edges t for all t. As well the edge $\forall u \cdot (\mathrm{dec}(t,k) = u)$ is a multiple edge representing the edge for all u.

Let us not that the error message gives out all interesting information on the error.

We apply ψ_k to these trees as previously, apart for the right member of the equalities 4 and we prove in the same way that if $\tilde{T}$ is such a tree, $\psi_k(\tilde{T}) \approx \tilde{T}$.

Lemma 13. *Let P be a process without key cycles, we have $P \approx \mathcal{O}_{T_P}$*

Proof. Let η be a security parameter, τ a random sample, $\mathcal{A}$ a computational adversary. The lemma 12 proves that $\mathcal{A}$ can distinguish P and T_P only if $\mathcal{A}^{\mathcal{O}_{\tilde{T}_P}}$ gives a failure case, otherwise $\mathcal{A}^P$ and $\mathcal{A}^{\mathcal{O}_{T_P}}$ behave exactly in the same way.

As $\tilde{T}_P \approx \tilde{T}_P^{n_1}$, we can replace $\tilde{T}_P$ by $\tilde{T}_P^{n_1}$. The problem is now to find a failure case in $\tilde{T}_P^{n_1}$. Let us use a joint IND-CPA, INT-CTXT oracle on all honest keys that are used. Let us define a simulator B, which will play against the security game, and which behaves as $\mathcal{O}_{\tilde{T}_P^{n_1}}$ appart from the fact that it uses the oracle to encrypt and decrypt. The oracles can be used for encrypting as we use only zeros as plaintexts (for honest keys).

Let us now explain how B behaves on a failure case :

- As the only honest encryptions are encryptions of zeros, all honest encryptions that are performed are deducible. Every non deducible term has a subterm which is a honest encryption unknown to the adversary or a secret honest nonce. Then finding a non deducible term breaks the INT-CTXT property or is information theoretically impossible. With this property, if B finds a non deducible term in an invalid equality or receives a non deducible term, it breaks INT-CTXT. This can not occur with non negligible probability, which rules out the cases 3, 4 and 5 from lemma 12.

– In the case 6 where a honest decryption succeeds when it should fail, the parsing function ensures that the term was not some already produced encryption. This breaks INT-CTXT.
– In the case 7 where we have an equality between two encryptions with two different honest keys and we break the "confusion freeness" of the cryptographic scheme, i.e. INT-CTXT (see [MW04]).
– If the case 8 occurs with non-negligible probability, then there exists a particular key on which it occurs. In that case for some $0 < l < \eta$:

$$\mathbf{Pr}\{r, r' \xleftarrow{R} U, k \xleftarrow{R} \mathcal{K}, \mathcal{E}(0^l, k, r) = \mathcal{E}(0^l, k, r)\}\text{is not negligible}$$

This can not occur with an IND-CPA scheme. If it was the case the following adversary would win with non negligible probability against IND-CPA
A takes $l \xleftarrow{R} \{1, \cdots, \eta\}$
A gets $m = \mathcal{O}(0^l)$
A gets $m' = \mathcal{O}(1^l)$
If $m = m'$ A sends out 0 otherwise A outputs $b \xleftarrow{R} \{0, 1\}$
A would win with non negligible probability against IND-CPA which contradicts the hypotheses.
– The only cases left are 9 in which two honest names are equals or the cases 10 and 11 in which the adversary guesses a name or an encryption which is still not generated. As there is a polynomial number of names and an exponential number of possible values the case 9 can not occur with non negligible probability. By information theoretic arguments the cases 10 and 11 can only occur with an exponentially small probability, as the random can only be used once and there is an exponential number of values for the encryption of the same constant (as the encryption scheme is IND-CPA).

As no failure case can occur with a non negligible probability, and there is only a finite number of failure cases, the proof is complete.

H Secrecy proof for the protocol of Section 6

H.1 Proof

We want to prove an over-approximation on which k_{AB} cyphertexts can be obtained by the attacker. First of all let us note that the secrecy of k_{AB} is ensured by the fact that it is never used as a plaintext subterm.

Let $(k_1^i)_{i\in\mathbb{N}}$ (resp. $(k_2^i)_{i\in\mathbb{N}}$, $(k_3^i)_{i\in\mathbb{N}}$, $(N^i)_{i\in\mathbb{N}}$) be the successive renaming of k_1 (resp. k_2, k_3, N) in the process P_B, we use the same renaming convention for the randomness used in the encryptions in P_B.

We prove the following result for every trace s of $(\nu k_{AB}, s_{AB})!P_A\|!P_B$.

Lemma 14. *Let s be a trace of $(\nu k_{AB}, s_{AB})!P_A\|!P_B$. Let $E, X_w, X_c, Q\|\phi$ be such that*

$$\emptyset, \emptyset, \emptyset, (\nu k_{AB}, s_{AB})!P_A\|!P_B \xrightarrow{s} E, X_w, X_c, Q\|\phi$$

1. $\phi \not\vdash_E k_2$ *and* $\phi \not\vdash_E k_3$
2. *for every* $u \in \mathcal{T}$, *if* $\phi \vdash_E u$ *then*

$$\nu\tilde{n}, (\{k_2^i\}_{k_3^i}^{r_1^i})_{i\in\mathbb{N}}, (k_1^i)_{i\in\mathbb{N}}, (r_2^i)_{i\in\mathbb{N}}, (N^i)_{i\in\mathbb{N}} \vdash_E u$$

   where $\tilde{n}$ *stands for all possible instantiation of the restricted names of* P_A.
3. *for every* u, r *if* $\phi \vdash_E \{u\}_{k_{AB}}^r$ *then*
   - $u = \langle u_1, u_2 \rangle$
   - $\nu\tilde{n}, (\{k_2^i\}_{k_3^i}^{r_1^i})_{i\in\mathbb{N}}, (k_1^i)_{i\in\mathbb{N}}, (r_2^i)_{i\in\mathbb{N}}, (N^i)_{i\in\mathbb{N}} \vdash_E u_1$
   - *there exists* $i \in \mathbb{N}$ *such that* $u_2 = N^i$ *or* $u_2 = k_3^i$

Proof. The lemma trivially holds when the trace is empty, and any trace should begin with an input.

Let us reason by induction. Let $s.\alpha$ be a trace such that the invariant is true for s. Let $E, X_w, X_c, Q\|\phi$ be such that

$$\emptyset, \emptyset, \emptyset, (\nu k_{AB}, s_{AB})!P_A\|!P_B \xrightarrow{s} E, X_w, X_c, Q\|\phi$$

If α is an equation, the only point to check is 1. As the equation added should be well formed with respect to ϕ, and k_2, k_3 were secret, we know that k_2 and k_3 remain secret. The lemma still holds on $s.\alpha$.

If α is $c_{in}(u)$, the lemma still trivially holds on $s.\alpha$.

If α does not fall in the previous cases, let $c_{in}(u)$ be the last input in s. The only interesting case is the case in which a copy of P_B takes u for first input. As k_{AB} is secret and the lemma holds on s, if u is not of the form $\langle k, \{v, w\}_{k_{AB}}^-\rangle$ where k is a dishonest key and w is either an instance of k_3 or an instance of N the process simply rejects the message and the next step in the trace must be an input action. If u is of this form, the process checks that $\mathrm{M}(\mathrm{dec}(v, k))$ holds, possibly giving $\langle v, k, \mathrm{dec}\rangle$ to the adversary. As v, k must be deducible from $\nu\tilde{n}, (\{k_2^i\}_{k_3^i}^{r_1^i})_{i\in\mathbb{N}}, (k_1^i)_{i\in\mathbb{N}}, (r_2^i)_{i\in\mathbb{N}}, (N^i)_{i\in\mathbb{N}}$, then the lemma still holds after adding this knowledge to the frame. As shown previously adding an equation $\mathrm{dec}(v, k) = t$ can not yield to breaking our invariant and as t must be deducible from $\phi \cup \{y \mapsto \langle v, k, \mathrm{dec}\rangle\}$, t is deducible from $\nu\tilde{n}, (\{k_2^i\}_{k_3^i}^{r_1^i})_{i\in\mathbb{N}}, (k_1^i)_{i\in\mathbb{N}}, (r_2^i)_{i\in\mathbb{N}}, (N^i)_{i\in\mathbb{N}}$. If (possibly after adding an equation $\mathrm{dec}(v, k) = \bot$) the decryption does not succeed, the process should be waiting for an input. If the decryption $\mathrm{dec}(v, k) = t$ with $t \neq \bot$, then t is either the right member of an equation which was added at the previous step or $\mathrm{dec}(v, k)\downarrow_E$ and as k is known and v is deducible from $\nu\tilde{n}, (\{k_2^i\}_{k_3^i}^{r_1^i})_{i\in\mathbb{N}}, (k_1^i)_{i\in\mathbb{N}}, (r_2^i)_{i\in\mathbb{N}}, (N^i)_{i\in\mathbb{N}}$, then t is deducible from $\nu\tilde{n}, (\{k_2^i\}_{k_3^i}^{r_1^i})_{i\in\mathbb{N}}, (k_1^i)_{i\in\mathbb{N}}, (r_2^i)_{i\in\mathbb{N}}, (N^i)_{i\in\mathbb{N}}$. If $\mathrm{dec}(v, k) = t$, then the process outputs $\{t, w\}_{k_{AB}}^r$ for some secret r, and this cyphertext is of the form of k_{AB} cyphertext in the lemma, which can not break the invariant. The process then waits for an input.

All other cases are easy, as our invariant ensures that the process only decrypts with honest keys and checks equalities between honest encryption which

(with our restrictions on equations) can not add knowledge to the adversary nor lead to adding equations a simple look at the relations between input and output of a process ensures that we do not break the invariant.

Corollary 3. *s_{AB} only occurs under encryption by a honest key, and then remains secret for any attacker A interacting with $(\nu k_{AB}, s_{AB})(!P_A \| !P_B)$.*

Proof. The lemma ensures that the key k used for encrypting s_{AB} must be $\mathrm{dec}(u, v)$ with $\{u, v\}^r_{k_{AB}}$ deducible from $\nu\tilde{n}, (\{k_2^i\}_{k_3^i}^{r_1^i})_{i\in\mathbb{N}}, (k_1^i)_{i\in\mathbb{N}}, (r_2^i)_{i\in\mathbb{N}}, (N^i)_{i\in\mathbb{N}}$. As the lemma ensures that v can only be an instance of N or an instance of k_3, then as the decryption must succeed, v is an instance of k_3. The lemma ensures that u is deducible from $\nu\tilde{n}, (\{k_2^i\}_{k_3^i}^{r_1^i})_{i\in\mathbb{N}}, (k_1^i)_{i\in\mathbb{N}}, (r_2^i)_{i\in\mathbb{N}}, (N^i)_{i\in\mathbb{N}}$, and as $\mathrm{dec}(u, k_3^i)$ succeed for some $i \in \mathbb{N}$, and k_3^i is secret, the restrictions on the equations forces $u = \{k_2^i\}_{k_3^i}^{r_1^i}$, and then $k = k_2^i$ which, by the previous lemma, is secret.

Corollary 4. *The process $(\nu k_{AB}, s_{AB})!P_B \| !P_A \| c_{in}(x).[x = s_{AB}].\overline{c_{error}}$ never emits on channel c_{error}.*

Proof. As proven previously, s_{AB} remains secret in any trace of the process $(\nu k_{AB}, s_{AB})(!P_A \| !P_B)$, then in any trace of $(\nu k_{AB}, s_{AB})!P_B \| !P_A \| c_{in}(x).[x = s_{AB}].\overline{c_{error}}$ in which the witness process has not taken any input. When the witness process receives an input it can not be s_{AB} as s_{AB} is still secret, and then, as s_{AB} is a honest name, no matter what was the input u the equation $u = s_{AB}$ can not be added which ensures that the condition is false.

H.2 Non trivial traces

Let us show that this process has traces that would not be present in another symbolic model. For simplicity reasons, we will omit the restrictions on our frames, as the set of restricted names is quite clear.

We show that any instance of N (recall that $N = \{N'\}^{r_N}_{k_N}$) can be learned by the adversary and that the process can produce an output $\{u, k_3^l\}^r_{k_{AB}}$ where u is any deducible term, with r some nonce and k_3^l an instance of k_3. Let N^i be the target instance of N. Let us assume that the adversary has "activated" the $(i-1)$-th version of P_A (it is clearly possible by doing $i-1$ inputs). From this

point, the following trace is correct, if ϕ is the current frame :

$$
\begin{array}{ll}
\phi \xrightarrow{\overline{c_{out}}(k_1^i,\{\{\{k_2^i\}_{k_3^i}^{r_1^i}\}_{k_1^i}^{r_2^i},N^i\}_{k_{AB}}^{r_3^i})} & \underbrace{\phi \cup \{x_1 \mapsto k_1^i, \{\{\{k_2^i\}_{k_3^i}^{r_1^i}\}_{k_1^i}^{r_2^i}, N^i\}_{k_{AB}}^{r_3^i}\}}_{\phi_1} \\
\xrightarrow{c_{in}(k_1^i,\{\{\{k_2^i\}_{k_3^i}^{r_1^i}\}_{k_1^i}^{r_2^i},N^i\}_{k_{AB}}^{r_3^i})} & \phi_1 \\
\xrightarrow{\overline{c_{out}}(\{\{k_2^i\}_{k_3^i}^{r_1^i},N^i\}_{k_{AB}}^{r_5^i})} & \underbrace{\phi_1 \cup \{x_2 \mapsto \{\{\{k_2^i\}_{k_3^i}^{r_1^i}, N^i\}_{k_{AB}}^{r_5^i}\}}_{\phi_2} \\
\xrightarrow{c_{in}(\{\{k_2^i\}_{k_3^i}^{r_1^i},N^i\}_{k_{AB}}^{r_5^i})} & \phi_2 \\
\xrightarrow{\overline{c_{out}}(\{N^i,k_3^i\}_{k_{AB}}^{r_4^i})} & \underbrace{\phi_2 \cup \{x_3 \mapsto \{N^i, k_3^i\}_{k_{AB}}^{r_4^i}\}}_{\phi_3} \\
\xrightarrow{c_{in}(k,\{\{N^i,k_3^i\}_{k_{AB}}^{r_4^i}\})} & \phi_3 \cup \{y_1 \mapsto \langle N^i, k, \mathrm{dec}\rangle\} \\
\xrightarrow{eq(\mathrm{dec}(N^i,k)=u)} & \phi_3 \cup \{y_1 \mapsto \langle N^i, k, \mathrm{dec}\rangle\} \\
\xrightarrow{\overline{c_{out}}(\{u,k_3^i\}_{k_{AB}}^{r_5^{i+1}})} & \phi_3 \cup \{y_1 \mapsto \langle N^i, k, \mathrm{dec}\rangle, x_4 \mapsto \{u, k_3^i\}_{k_{AB}}^{r_5^{i+1}}\}
\end{array}
$$

It is clear that with the last frame of this trace, N^i is deducible.