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.

P. Degano and J.D. Guttman (Eds.): POST 2012, LNCS 7215, pp. 149–168, 2012.

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.

The issue of dishonest keys is also considered in [KT09]. Here, the authors identify sufficient conditions on protocols such that dishonest keys are provably harmless. These conditions are satisfied by a large family of key exchange protocols. These conditions may however be too strong. For example, the protocol we analyse at the end of this paper does not meet their conditions, while we can prove it secure using our framework.

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.

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

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.

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

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.

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

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.

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

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.

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

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).

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

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.

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

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. There are three main options for establishing soundness: either re-enforcing the security assumptions of the primitives, or identifying sufficient conditions for recovering soundness, or relaxing the symbolic models. In this paper, we have opted for the third option. 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.

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

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 $\sigma = \{x_1 \mapsto \{n_1\}^{r_1}_{k_1}, x_2 \mapsto \langle\{n_2\}^{r_2}_{n_1}, \{n_3\}^{r_3}_{n_2}\rangle\}$. 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

$\Phi_1, \Phi_2 ::=$	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
`if` Φ `then` P `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 `if` Φ `then` P `else` $\overline{c_{out}}(\bot)$ as $[\Phi]P$.

Fig. 1. Syntax of Formula and Processes

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.

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 $\mathcal{T}/E \cup E_0$. $E \cup E_0$

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

Fig. 2. Structural equivalence

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 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*

- $E \not\vDash (s = t)$
- *if $(v \neq w) \in E$, $E \cup \{s = t\} \not\vDash v = w$*
- *$E \cup \{s = t\} \not\vDash n = n'$ with n, n' names and $n \neq n'$*
- *$E \cup \{s = t\} \not\vDash \{u\}_k^r = \{u'\}_{k'}^{r'}$ with $k, k' \in \mathcal{K}_1$ and $k \neq k'$ or $r \neq r'$*
- *$E \cup \{s = t\} \not\vDash u = v$ 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. $s = \{u\}^r_k$ *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* $E \models \mathrm{M}(s)$*,* $t{\downarrow_E} = k$ *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* $E \models \mathrm{M}(t) \wedge \mathrm{M}(u)$ *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 $\{y_1 \mapsto x_1, \cdots, y_n \mapsto x_n\}$. $\phi\backslash X$ 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 $P \equiv \nu\tilde{n}[\Phi_1]P_1\| \cdots \|[\Phi_n]P_n\|Q$ 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

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

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

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

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', $A \xrightarrow{*} C[\bar{c}(s) \cdot A']$, then for some context C', term s' and process B', $B \xrightarrow{*} C'[\bar{c}(s') \cdot B']$*
- *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 :*

$$(\nu \bar{n})[(\nu \bar{x}_1, \bar{n}_1 B_1 \| \sigma_1) \| \cdots \| (\nu \bar{x}_k, \bar{n}_k B_k \| \sigma_k) \| \\ !(\nu \bar{z}_1, l_1, \bar{m}_1 \overline{c_{out}}(\langle 1, l_1 \rangle) B'_1) \| \cdots \| !(\nu \bar{z}_n, l_n, \bar{m}_n \overline{c_{out}}(\langle n, l_n \rangle) B'_n)]$$

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 network of Communicating Turing Machine (CTM), and we can relate each state of a process to a state of the corresponding Turing Machine. We assume that each Turing Machine has an independent random tape. We denote by τ the set of random tapes of the

machines under consideration. The attacker controls the network: it is a CTM equipped with an additional control tape, that contains the CTM's id that will be used for the next communication. It may perform an internal transition, or a *send* (resp. *receive*) action, that copies the contents of its sending (resp.the designated CTM sending) tape to the designated CTM receiving (resp. its receiving) tape. In addition, it may perform a *new* action, creating a new copy of a machine implementing the replicated process specified on the control tape. 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. We write $[\![P]\!]^{\tau}_{\eta}$ for the implementation of the simple process P with security parameter η and randomness τ. We will often write A_{τ} for the attacker using the random tape specified by τ.

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. *Given τ, let (γ_i) be the sequence of configurations preceding a send action in the deterministic computation of $\mathcal{A}_\tau$ interacting with $[\![P]\!]^\tau_\eta$. The* execution sequence *of $\mathcal{A}_\tau \| [\![P]\!]^\tau_\eta$ is the sequence $\gamma_1, L_1 \xrightarrow{m_1} \gamma_2, L_2 \cdots \xrightarrow{m_n} \gamma_n, L_n$ where m_i is the content of the sending tape in γ_i, $L_i = L_{i-1} \cdot m_{i-1} \cdot R_i$ and R_i is the sequence of contents of the receiving tape along the computation from γ_{i-1} to γ_i ($L_1 = R_1$).*

Let us now define *symbolic traces* of a process P as the sequence of terms exchanged with the adversary:

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 $i \le n$ if $\alpha_i = c_{in}(t_i)$, then $P_{i-1} = P' \| \phi$ with ϕ a frame and $\phi \vdash_{E_{i-1}} t_i$.*

The full abstraction property states that a computational execution is the interpretation of some symbolic trace:

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 $k = n$ and $\forall j \le n$*

- *$\alpha_{n_j} = c_{in}(t_j)$ and $[\![t_j]\!]^\tau_\eta = m_j$*
- *If $P \xrightarrow{\alpha_1.\cdots.\alpha_{n_{j+1}-1}} E^j, X^j_c, X^j_w, Q^j \| \phi^j$ with Q^j not containing active substitutions, then*
  - *$[\![Q^j]\!]^\tau_\eta = \gamma_j$*
  - *$[\![\phi_j \cap \{x \mapsto t | t \in \mathcal{T}, x \in \mathcal{V}_1\}]\!]^\tau_\eta = L_j$*
  - *$\forall (s = t) \in E^j, [\![s]\!]^\tau_\eta = [\![t]\!]^\tau_\eta$*
  - *$\forall (s \neq t) \in E^j, [\![s]\!]^\tau_\eta \neq [\![t]\!]^\tau_\eta$*

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

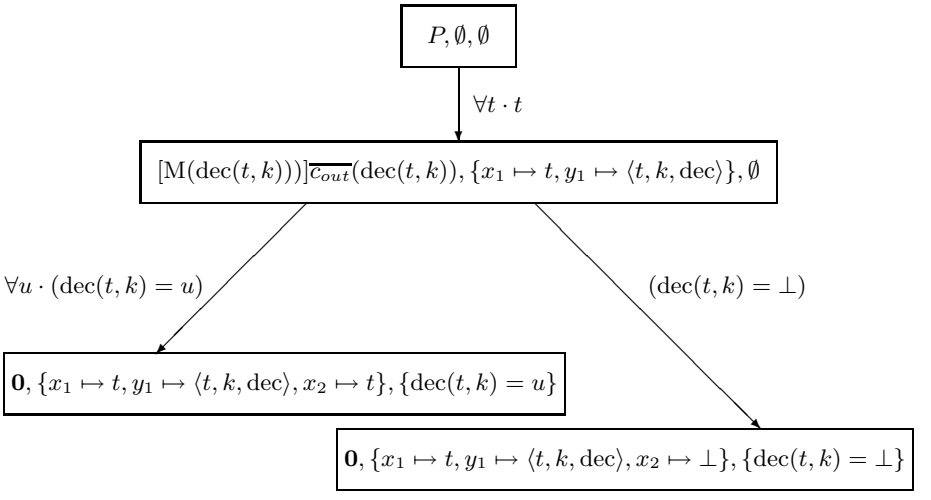
We are now able to give our full abstraction theorem :

Theorem 1. *Let P be a simple process without key cycles. For every PPT $\mathcal{A}$, for every security parameter η, the sequence* Messages(P, η, τ) *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.



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

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.
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 $\{N'\}^{r_N}_{k_N}$ where k_N is a secret key known only to **A**. We actually simply need N to be of type cyphertext to make our example more interesting.

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

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 $(\nu N', k_N, r_N)$.

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

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 :

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

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.

A full version of this paper, including the proofs, is available on the Cryptology ePrint Archive at `http://eprint.iacr.org/2012/008`.

References

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

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

[AR00] Abadi, M., Rogaway, P.: Reconciling Two Views of Cryptography: the Computational Soundness of Formal Encryption. In: Watanabe, O., Hagiya, M., Ito, T., van Leeuwen, J., Mosses, P.D. (eds.) TCS 2000. LNCS, vol. 1872, pp. 3–22. Springer, Heidelberg (2000)

[BHO09] Bana, G., Hasebe, K., Okada, M.: Computational Semantics for First-Order Logical Analysis of Cryptographic Protocols. In: Cortier, V., Kirchner, C., Okada, M., Sakurada, H. (eds.) Formal to Practical Security. LNCS, vol. 5458, pp. 33–56. Springer, Heidelberg (2009)

[Bla05] Blanchet, B.: 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] Blanchet, B.: 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] Backes, M., Pfitzmann, B.: Symmetric encryption in a simulatable dolev-yao style cryptographic library. In: Proc. IEEE Computer Security Foundations Workshop (2004)

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

[CC11] Comon-Lundh, H., Cortier, V.: 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 2011). LIPIcs, vol. 9, pp. 29–44 (2011)

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

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

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

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

[RS98] Ryan, P., Schneider, S.: An attack on a recursive authentication protocol: a cautionary tale. Information Processing Letters 65, 7–10 (1998)

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