Computational Soundness of Indistinguishability Properties without Computable Parsing

Hubert Comon-Lundh[1], Masami Hagiya[2], Yusuke Kawamoto[1], and Hideki Sakurada[3]

[1] LSV, CNRS, ENS Cachan and INRIA, France*
[2] University of Tokyo, Japan
[3] NTT Communication Science Laboratories, NTT Corporation, Japan

Abstract. We provide a symbolic model for protocols using public-key encryption and hash function, and prove that this model is computationally sound: if there is an attack in the computational world, then there is an attack in the symbolic (abstract) model. Our original contribution is that we deal with the security properties, such as anonymity, which cannot be described using a single execution trace, while considering an unbounded number of sessions of the protocols in the presence of active and adaptive adversaries. Our soundness proof is different from all existing studies in that it does not require a computable parsing function from bit strings to terms. This allows us to deal with more cryptographic primitives, such as a preimage-resistant and collision-resistant hash function whose input may have different lengths.

1 Introduction

There are two main approaches to the analysis of protocol security. The first considers an attacker modeled as a probabilistic polynomial-time (PPT) interactive Turing machine (ITM) and a protocol is an unbounded number of copies of ITMs. The attacker is assumed to control the network and can schedule the communications and send fake messages. The security property is defined as an indistinguishability game: the protocol is secure if, for any attacker A, the probability that A gets an advantage in this game is negligible. A typical example is the *anonymity* property, by which an attacker should not be able to distinguish between two networks in one of which identities have been switched. The difficulty with such computational security notions lies in the problem of obtaining detailed proofs: they are in general unmanageable, and cannot be verified by automatic tools.

The second approach relies on a formal model: bit strings are abstracted by formal expressions (terms), the attacker is any formal process, and security properties, such as anonymity, can be expressed by the observational equivalence of processes. This model is much simpler: there is no coin tossing, no complexity bounds, and the attacker is given only a fixed set of primitive operations (the

* This work has been supporter by the ANR project ProSe.

function symbols in the term algebra). Therefore it is not surprising that security proofs become much simpler and can sometimes be automatized. However, the drawback is that we might miss some attacks because the model might be too abstract.

Starting with work of Abadi and Rogaway [2] and Backes, Pfitzmann and Waidner [4], there have been several results showing the *computational soundness* of the formal models: we do not miss any attacks when considering the abstract model, provided that the security primitives satisfy certain properties; for instance IND-CPA or IND-CCA in the case of encryption. Such results avoid the weaknesses of both approaches to security.

In their original work, Abadi and Rogaway only considered symmetric encryption and a passive attacker with some other minor restrictions. This has been extended in a number of directions. For instance, Backes et al. consider active attackers and several cryptographic primitives and show simulatability theorems, which imply the computational soundness of some formal model [4,6]. There are also several other soundness results, typically for some trace properties [13,9] and, more recently, for equivalence properties [7].

In the present work, we show a soundness result for active attackers and public-key encryption and hash functions as cryptographic primitives. Our result extends the previous work in the following respects:

1. In addition to security properties that can be checked on each trace, i.e., each individual sequence of events, we consider equivalence properties. Therefore, our work does not fit into the general framework of [3], which only considers trace properties. Actually, we need rather different proof techniques, such as tree oracles and tree transformations. The only previous results concerning equivalence properties in the presence of active attackers are [7], in which only symmetric encryption and a particular class of processes is considered, and [12], in which only a fixed number of protocol instances is considered. We do not assume here any bound on the number of protocol instances.
2. Our soundness proof does not use a *computable parsing function* from bit strings to terms. This is a major difference from all existing studies on computational soundness. It allows us to deal with a *preimage-resistant and collision-resistant hash function* whose input may have different lengths, for which the soundness of process calculi cannot be obtained by [7]'s proof technique with the computable parsing function, as we detail later.
3. Unlike [7], we do not restrict ourselves to the so-called "simple processes". Simple processes are parallel compositions of replicated processes that are just finite sequences of inputs/outputs and tests, without conditional branching. We consider here a larger fragment of the applied π-calculus of [1]: we allow negative tests and non-trivial processes in both branches of conditional, as well as arbitrary replications. We keep however two important restrictions. First we assume that the processes are determinate: every copy of a process has first to generate communication channels and disclose them. In this way, the attacker can schedule to which copy of a process a message is sent. Furthermore, we do not allow private communication channels. Consid-

ering such private channels would require to consider timing attacks, or (for instance) to assume that an attacker cannot distinguish between terminating and non-terminating processes, as investigated in [15].

4. We included hash functions in our set of primitives in order to illustrate the usefulness of dropping the parsing assumptions. Previous (positive) results on the computational soundness of hash functions either assume stronger properties of the hash functions than the standard preimage-resistance and collision-resistance ([5,11,10,8]), or they assume that all plaintexts have the same size ([5]). In any case, all these studies do not consider both an active attacker and indistinguishability properties. On the other hand, we assume that the hash function is only applied to nonces (possibly of different lengths).
5. Finally, we observe that the ability of the computational attacker to observe the length of bit strings cannot be soundly represented using a (reasonable) symbolic length function. Up to our knowledge, this problem has never been considered in the papers on computational soundness. We propose here a new solution where each plaintext term is associated with a label representing its *expected* length in the symbolic model, and protocols accept only input of expected length in the computational model. This is a reasonable assumption that can be easily implemented.

Our proof relies on ideas that are similar to [7]: each process is associated with a *computation tree*, which records all possible executions of the process. The observational equivalence between two processes implies some labeled bisimilarity between the computation trees. Computation trees are used as *tree oracles* in the computational model. The computational indistinguishability between two processes is then equivalent to that between the two tree oracles. The proof proceeds by successive transformations of the computation trees, in such a way that the attacker wins a game (may distinguish the tree oracles) iff he wins the game against the transformed trees. Eventually, the computation trees are simple enough: it is straightforward that the attacker cannot win.

There are however important differences between our work and [7], which we summarize now. Without computable parsing, we cannot rely on the same notion of computation trees as in [7]; in [7], when the attacker submits a bit string to the tree oracle, the bit string is parsed into a term and the branch of the computation tree labeled with this term is then taken. In this paper, we consider symbolic computation trees, in which all term labels that satisfy the same conditions are gathered together: now the computation trees are finitely branching and the edges are labeled with formulas rather than terms. The formulas are evaluated on the attacker's input, whether in the computational or in the symbolic model. In addition, since the formulas are arbitrary Boolean combinations of atomic formulas, we may allow arbitrary conditional branching (which is not the case in [7]).

Symbolic computation trees however introduce new difficulties, since the previous transformations are no longer valid. We need therefore additional trans-

formations, as well as a *partial unraveling* of the computation tree.

We also differ from [7] in three other respects:

1. We consider hash functions (in the standard model), whose inputs may have different structures (we use two different constructors for nonces of different lengths): this is an example where we cannot assume a computable parsing function. On the other hand, the impossibility result of [5] shows that the BRISM/UC results cannot be extended to hash functions in the standard model, when the plaintexts may have different structures; the BRISM framework relies on the existence of a computable parsing function, allowing to translate bit strings into terms.
2. The soundness of indistinguishability requires a symbolic length function: we need to provide the symbolic attacker with a capability to distinguish terms, whose implementations are bit strings of different lengths. This is difficult because, for instance, a pair $\langle u, u\rangle$ and a ciphertext $\{u\}^r_{\mathsf{ek}(k)}$ may or not have the same length, depending on the security parameter, while a symbolic length function cannot depend on a security parameter; should $\langle u, u\rangle$ and $\{u\}^r_{\mathsf{ek}(k)}$ get the same symbolic length or not ? A symbolic length function is hardly sound with respect to the computational length.
   The solution adopted in [7] is to assume that the length of any cryptographic primitive applied to some arguments is a homogeneous function of the lengths of its arguments. Typically, in case of a linear function, the length $|[\![\langle u, v\rangle]\!]_\eta|$ of the computational interpretation of a pair, with respect to the security parameter η must be $\alpha \times |[\![u]\!]_\eta| + \beta \times |[\![v]\!]_\eta| + \gamma \times \eta$. Then, by induction, we may factor out η from the length of the interpretation of any term and get (in)equalities on lengths, independently of η.
   This is a strong restriction on the implementation since, for instance, the pairing operation does not have a constant overhead; it depends linearly on the security parameter.
   We propose another solution here, relying on labels, that is more realistic: messages received by honest agents do have expected lengths. Such an expected length is represented by a symbolic label such that two identical labels yield the same computational length of messages in an honest execution. On the computational side, we assume that the honest agents check the lengths of messages that they receive or encrypt, which is easy to implement, so that the actual length matches the expected one. It turns out that these assumptions, together with some weak length-regularity of the cryptographic primitives, are sufficient for the soundness result, as we show in this paper.
3. [7] uses a trace mapping property, whose use in the indistinguishability games is unclear. We formalize a new transformation of computation trees, in which computational traces that do not have a symbolic counterpart yield a failure node.

We do not have the space to present here the result in detail, which will appear as a research report. We sketch our symbolic model in Section 2, emphasizing the unusual components. Similarly, we sketch our computational interpre-

tation in Section 3 and finally sketch the main steps of the proofs of our main result in Section 4.

2 The symbolic model

2.1 Terms

We rely on a variant of the applied π-calculus [1]. Terms are built from names (out of a set $\mathcal{N}$), variables (out of a set $\mathcal{X}$), the *constructor function symbols* $\mathsf{FuncC} = \{\mathsf{n}^1(\_), \mathsf{n}^2(\_), \langle \_, \_ \rangle, \mathsf{h}(\_), \mathsf{ek}(\_), \mathsf{dk}(\_), \mathsf{cert}(\_), \{\_\}_\_\}$ and the *destructor function symbols* $\mathsf{FuncD} = \{\pi_1(\_), \pi_2(\_), \mathsf{dec}(\_, \_)\}$. Let $\mathcal{F} = \mathsf{FuncC} \cup \mathsf{FuncD}$. $\mathsf{n}^1(\_)$ and $\mathsf{n}^2(\_)$ are two constructor symbols for nonces. They can get only names as arguments. They are intended to produce names of different lengths. $\mathsf{cert}(\_)$ is a constructor for public keys certificates, $\mathsf{h}(\_)$ is a hash function symbol and $\mathsf{ek}(\_), \mathsf{dk}(\_)$ are intended to represent encryption and decryption keys, respectively. These two symbols can only take names as arguments.

All the function symbols are available to the attacker, except for $\mathsf{cert}(\_)$. Term is the set of ground terms built on these function symbols, the names and a set of constants Const. $\mathsf{Term}(\mathcal{X})$ are the terms that may additionally contain variables from $\mathcal{X}$. *Constructor terms* are terms that do not contain symbols from FuncD. For any expression or set of expressions S, $\mathsf{Var}(S)$ is the set of variables occurring free in S.

The function symbols satisfy the equations of Fig. 1. In these equations,

$$\begin{aligned} \pi_1(\langle x, y \rangle) &= x \\ \pi_2(\langle x, y \rangle) &= y \\ \mathsf{dec}(\{x\}^r_{\mathsf{ek}(k)}, \mathsf{dk}(k)) &= x \quad \text{if } k, r \in \mathsf{Name} \end{aligned}$$

Fig. 1. Equational specification of the algebra

the variables x, y range over any ground constructor term. This corresponds to a "call-by-value" interpretation of the destructors. In other words, the implementation is strict: if an argument of a destructor is not a constructor term, we cannot apply these equations to cancel the destructor. The set of equations is infinite, as there are symbols k, r, r' that range over all possible names. Then, Term might be seen as a quotient algebra with respect to the congruence generated by the equations of Fig. 1. We ambiguously keep the same notation Term and $\mathsf{Term}(\mathcal{X})$ for the quotients. Term is then a $\mathcal{F}$-algebra: morphisms and first-order structures are defined as usual, referring to this quotient structure.

We orient all equations of Fig. 1 from left to right. This yields an infinite convergent term rewriting system on terms. The normal form of u is written as $u{\downarrow}$.

A *labeled term* is either a term or a symbolic expression obtained from a term by labeling some of its subterms with labels in Label. More formally, the set LTerm of labeled terms is defined by:

$$\mathsf{LTerm} ::= \mathsf{Term} \mid \mathcal{F}(\mathsf{LTerm}, \cdots, \mathsf{LTerm}) \mid \mathsf{Term}{:}\mathsf{Label} \mid \mathcal{F}(\mathsf{LTerm}, \cdots, \mathsf{LTerm}){:}\mathsf{Label}$$

For instance, $\{\langle \mathsf{n}^1(r), \mathsf{n}^2(r'){:}l_{n^2}\rangle{:}l_1\}^{r''}_{\mathsf{ek}(k)}$ and $\{\mathsf{h}(\mathsf{n}^1(r){:}l_{n^1}){:}l_2\}^{r'}_{\mathsf{ek}(k)}$ are labeled terms. Intuitively, $u{:}l$ represents a message whose length is expected to be l in an honest protocol execution.

The rewrite rules of Fig. 1 can only be applied to unlabeled instances of the variables; when we rewrite a labeled term, the labels of the rule instances are implicitly removed. In this way, we keep the confluence and termination of the rewrite system and rewriting a labeled term yields a labeled term (for instance, we do not get $u{:}l{:}l'$).

2.2 Predicates, conditions, frames and static equivalence

Predicates are used either in honest processes, in order to check properties of the input terms, or by the attacker, in order to distinguish sequences of terms. We consider the following predicate symbols: M, EQ, EK, $IsEK$,IsN_1, IsN_2, PL, and HL, whose (informal) meaning is as follows. $M(u)$ holds on ground terms u such that $u{\downarrow}$ is a constructor term. EQ is the strict equality predicate: $EQ(u, v)$ implies $u{\downarrow} = v{\downarrow}$ and $M(u)$ and $M(v)$. EK holds on a ciphertext $\{u\}^w_v$ and a public key $\mathsf{ek}(k)$ when $v = \mathsf{ek}(k)$. $IsEK$ is true on pairs of an encryption key and a certificate of that key. IsN_1 holds on terms $\mathsf{n}^1(r)$ with $r \in \mathsf{Name}$ and IsN_2 holds on terms $\mathsf{n}^2(r)$ with $r \in \mathsf{Name}$. PL holds on two ciphertexts whose plaintexts have the same expected length, i.e. the same label. HL holds on two hash values whose plaintexts have the same label. A *condition* is a Boolean combination of atomic formulas. Examples of predicate interpretations are given in Example 1.

The *frames* usually record the messages that have been sent. Since we consider symbolic executions, we need to extend the classical definition to message templates that may contain variable. A *frame* is an an expression $\nu\overline{y}.\nu\overline{n}.\sigma$ where $\overline{y}$ is a finite set of variables, $\overline{n}$ is a finite set of names, σ is a substitution from a finite set of variables $\mathtt{dom}(\sigma)$ into $\mathsf{Term}(\mathcal{X})$ such that $\overline{y} \cap \mathtt{dom}(\sigma) = \emptyset$, and $\mathsf{Var}(\mathtt{codom}(\sigma)) \cap \mathtt{dom}(\sigma) = \emptyset$.

Given a frame ϕ, we write σ_ϕ the associated substitution, $\mathsf{bn}(\phi)$ is the associated sequence of bound names $\overline{n}$ and $\mathsf{bv}(\phi)$ is the associated sequence of bound variables $\overline{y}$. A *ground frame* ϕ is a frame such that $\mathsf{Var}(\mathtt{codom}(\sigma_\phi)) = \emptyset$. We recall here the definition of symbolic indistinguishability between ground frames (for general frames, this notion will be directly defined on computation trees).

Definition 1. Two ground frames ϕ_1 and ϕ_2 are *statically equivalent*, which is written as $\phi_1 \sim \phi_2$, if $\mathtt{dom}(\sigma_{\phi_1}) = \mathtt{dom}(\sigma_{\phi_2})$ and for any terms u and v such that cert does not occur in u, v, $\mathsf{Var}(u) \cup \mathsf{Var}(v) \subseteq \mathtt{dom}(\sigma_{\phi_1})$ and $(\mathsf{fn}(u) \cup \mathsf{fn}(v)) \cap (\mathsf{bn}(\phi_1) \cup \mathsf{bn}(\phi_2)) = \emptyset$, we have the following:

- For each $PR \in \{M, IsEK, IsN_1, IsN_2\}$, $\mathcal{M} \models PR(u\sigma_{\phi_1}{\downarrow})$ iff $\mathcal{M} \models PR(u\sigma_{\phi_2}{\downarrow})$.

– For each $PR \in \{EQ, EK, PL, HL\}$, $\mathcal{M} \models PR(u\sigma_{\phi_1}\downarrow, v\sigma_{\phi_1}\downarrow)$ iff $\mathcal{M} \models PR(u\sigma_{\phi_2}\downarrow, v\sigma_{\phi_2}\downarrow)$.

Example 1. In the examples, we omit the variables of the domains of σ_ϕ: they are always $x_1, \ldots, x_n$ where n is the length of the frame.

1. $\nu r.\ \mathsf{n}^1(r) \sim \nu r'.\ \mathsf{n}^1(r')$, while $\nu r.\ \mathsf{n}^1(r) \not\sim \nu r'.\ \mathsf{n}^2(r')$, since $\mathcal{M} \models IsN_1(\mathsf{n}^1(r))$ and $\mathcal{M} \not\models IsN_1(\mathsf{n}^2(r))$.
2. $\nu r.\ \{b{:}\, l\}^r_{\mathsf{ek}(a)} \sim \nu r.\ \{c{:}\, l'\}^r_{\mathsf{ek}(a)}$ iff $l = l'$, since $\mathcal{M} \models PL(\{b{:}\, l\}^r_{\mathsf{ek}(a)}, \{b'{:}\, l\}^{r'}_{\mathsf{ek}(a)})$ and $\mathcal{M} \models PL(\{c{:}l'\}^r_{\mathsf{ek}(a)}, \{b'{:}\, l\}^{r'}_{\mathsf{ek}(a)})$ iff $l = l'$. In this example, the recipe u is reduced to the variable x_1 and the recipe v is the ground (labeled) term $\{b'{:}\, l\}^{r'}_{\mathsf{ek}(a)}$.
3. $\nu a, r.\ \{b{:}\, l\}^r_{\mathsf{ek}(a)}, \mathsf{dk}(a) \not\sim \nu a, r.\ \{c{:}\, l\}^r_{\mathsf{ek}(a)}, \mathsf{dk}(a)$ if $b, c \in \mathcal{N}$ and a, b, c, r are pairwise distinct. It suffices to consider $u = \mathsf{dec}(x_1, x_2)$, $v = b{:}\ \mathcal{M} \models EQ(u\sigma_{\phi_1}\downarrow, v)$ while $\mathcal{M} \not\models EQ(u\sigma_{\phi_2}\downarrow, v)$.
4. $\nu a, a', r, r'.\ \{b{:}\, l\}^r_{\mathsf{ek}(a)}, \{b{:}\, l\}^{r'}_{\mathsf{ek}(a)}, \mathsf{ek}(a) \not\sim \nu a, a', r, r'.\ \{b{:}\, l\}^r_{\mathsf{ek}(a)}, \{b{:}\, l\}^{r'}_{\mathsf{ek}(a')}, \mathsf{ek}(a)$ using EK.

2.3 Processes

Processes are built as in the applied π-calculus [1], using the predicates and function symbols of the previous section. We do not recall here the syntax and the basic definitions. Let us explain the communication rule.

$$c(x{:}\, l).P \parallel \overline{c}(u{:}\, l).Q \quad \rightarrow \quad P\{x \mapsto u\} \parallel \{x \mapsto u\} \parallel Q$$

If a process $c(x{:}\, l).P$ is ready to receive a message on the channel c and another process is ready to emit the message u on channel c and if the two messages have the same label, then the network moves to a configuration in which x is replaced with u in P. The active substitution $\{x \mapsto u\}$ is kept (as a local memory of P).

While the attacker's processes are arbitrary processes (the attacker may re-label the terms as he wishes), protocols are specified as combinations of basic processes, using replication, name generation and parallel composition.

The basic processes are built using name generation, conditionals and sequences of input/output actions. We assume that all inputs are labeled variables. This is not a restriction, since the attacker may re-label the terms. We assume that all occurrences of plaintexts (of either ciphertexts or hashes) in the basic processes are labeled, and that before sending a message s the process always checks $M(s)$. This forbids sending ill-formed messages (or forwarding ill-formed message). Since, according to our semantics, ill-formed messages do not pass any test, the effect of message forwarding (moving the control point of some process) can be achieved with a well-formed message. We believe that this assumption is not a restriction.

The main restriction, with respect to the full applied π-calculus is that, each time a process is replicated, it must start with the generation of communication channels that are disclosed and then used as input/output channels. This ensures

the determinacy of processes: when the attacker sends a message on a channel c, there is at most one basic process that is able to receive a message on c.

Example 2. This is a simple process that first generates two channel names and disclose them, hence can be later replicated.

$$\begin{aligned} B(a,b,c,d) = \nu x, y.\ \nu i_{\mathsf{in}}, i_{\mathsf{out}}, r.\ c(x{:}\,l).\overline{c}(\langle i_{\mathsf{in}},\, i_{\mathsf{out}}\rangle).\\ i_{\mathsf{in}}(y{:}\,l).\ \mathsf{if}\ EQ(\pi_1(\mathsf{dec}(y,\,\mathsf{dk}(a)))), b) \wedge M(\pi_2(\mathsf{dec}(y,\,\mathsf{dk}(a)))) \wedge \mathit{IsEK}(\mathsf{ek}(b))\\ \mathsf{then}\ \overline{i_{\mathsf{out}}}(\{\pi_2(\mathsf{dec}(y,\,\mathsf{dk}(a))){:}\,l'\}^r_{\mathsf{ek}(b)})\\ \mathsf{else}\ \overline{i_{\mathsf{out}}}(\{d{:}\,l'\}^r_{\mathsf{ek}(b)}) \end{aligned}$$

Example 3. The following is an example of a protocol.

$$(\nu a)(\nu b)(\nu d)\ \big(\,!\,(\nu x)\ (\nu c_1, c_2)\ c(x{:}\,l)\ \overline{c}(\langle c_1,\, c_2\rangle)\ (!\,B(a,b,c_1,d))\|\ !B(b,a,c_2,d))\ \big)$$

The attacker, using the public channel c, may send a signal, which will give back fresh channel names c_1, c_2. This allows to get a copy of the (outermost) replicated process. Each of these channel names may then be used to request a copy of the corresponding instance of B.

Protocols may also include an initial setting, in which, for instance, some private keys are disclosed (static corruption).

We assume a number of (reasonable) properties of the protocols:

- Two occurrences of the same variable have the same label.
- The random seed r used in honest encryption terms $\{\_\}^r_{\_}$ only occur in that terms and the random seed k used for honest (i.e., certified) keys $\mathsf{ek}(k)$ and $\mathsf{dk}(k)$, are not used for any other purpose.
- Encryption keys with their certificates are sent to the attacker whenever they are generated.
- Only correct encryption keys are used for encryption (for unknown keys, *IsEK* is checked before encryption). This rules out the problem of keys that are forged by the attacker.
- Only hash values of nonces are produced by the protocols: for unknown plaintexts s, the protocol checks $\mathit{IsN}_1(s) \vee \mathit{IsN}_2(s)$ before hashing.
- There is no dynamic corruption: the protocols only disclose decryption key at the beginning of the execution.
- There is no key cycle: a key hierarchy ensures that the attacker cannot force the protocol to produce a cycle involving non-corrupted keys.

Finally, two processes P and Q are observationally equivalent, which we write $P \sim Q$ if, as usual, there is no context C such that $C[P]$ may emit on a channel a while $C[Q]$ cannot (or the converse).

3 Computational interpretation

3.1 Computational interpretation of terms and predicate symbols

Each function symbol f is associated with a function $[\![f]\!]$ from bit strings to bit strings that can be computed in deterministic polynomial time. These interpretations are assumed to satisfy the equations of Fig. 1, hence the set of bit strings

has a structure of $\mathcal{F}$-algebra. Let $\mathcal{SS}$ be a set of mappings from Name to $\{0,1\}^*$. Given $\tau \in \mathcal{SS}$, for any ground term u, $[\![u]\!]^\tau$ is the unique extension of τ into a homomorphism of $\mathcal{F}$-algebra. If u is a term with variables X and θ is a mapping from X into $\{0,1\}^*$, $[\![u]\!]^{\theta,\tau}$ is defined in a similar way. The interpretation of labeled terms is defined by ignoring the labels. Labels themselves are interpreted as natural numbers. The *security parameter* is the minimal length of $\tau(r)$ for $r \in \mathsf{Name}$.

In addition, we assume the following properties of the computational interpretation:

- The ranges of constructor function symbols are disjoint and disjoint from the interpretation of names. This assumption is necessary for a soundness result. However, we do not assume that the range of a function symbol is computable.
- We assume the following properties on lengths of the computational interpretation of names and nonces.
  - For each $b = 1, 2$ and any $r, r' \in \mathsf{Name}$, $\left|[\![\mathsf{n}^b(r)]\!]^\tau\right| = \left|[\![\mathsf{n}^b(r')]\!]^\tau\right|$, and that $\left|[\![\mathsf{n}^1(r)]\!]^\tau\right| < \left|[\![\mathsf{n}^2(r)]\!]^\tau\right| < |[\![r]\!]^\tau|$.
  - When τ is uniformly drawn (which we will assume in what follows), the distribution of $[\![\mathsf{n}^b(r)]\!]^\tau$ is uniform and covers all bit strings of the length $|[\![\mathsf{n}^b(r)]\!]^\tau|$.
  - $\left|[\![\mathsf{n}^1(r)]\!]^\tau\right|$ and $\left|[\![\mathsf{n}^2(r)]\!]^\tau\right|$ are polynomial in the security parameter η.

  Thanks to the assumptions, for instance, the lengths of keys are different from those of nonces in the computational model.
- We assume a weak notion of length regularity: Let $u{:}\, l$ be any labeled term occurring in a protocol such that $y_1{:}\, l_1, y_2{:}\, l_2, \ldots, y_n{:}\, l_n$ are all variables occurring in u. For any computational interpretation τ of names as bit strings, if terms $v_1, v_2, \ldots, v_n$ satisfy $|[\![v_i]\!]^\tau| = [\![l_i]\!]^\tau$ for $i = 1, 2, \ldots, n$, then $\left|[\![u\,\{y_1 \mapsto v_1, y_2 \mapsto v_2, \cdots, y_n \mapsto v_n\}]\!]^\tau\right| = [\![l]\!]^\tau$.

- For each $PR \in \{M,\ \mathit{IsEK}\}$ and any term u,
  - $\mathcal{M} \models PR(u)$ iff $[\![PR]\!]([\![u]\!]^\tau) = 1$ holds for any $\tau \in \mathcal{SS}$, and
  - $\mathcal{M} \models \neg PR(u)$ iff $[\![PR]\!]([\![u]\!]^\tau) = 0$ holds for any $\tau \in \mathcal{SS}$.

  Note that this does not imply anything on the interpretation of PR on a bit string that is not the interpretation of any term.
- For each $PR \in \{\mathit{IsN}_1,\ \mathit{IsN}_2\}$ and any bit string m, $[\![PR]\!](m) = 1$ iff there are $u \in \mathsf{Term}$ and $\tau \in \mathcal{SS}$ such that $m = [\![u]\!]^\tau$ and $\mathcal{M} \models PR(u)$. For instance, we can implement $[\![\mathit{IsN}_b]\!](m)$ by checking whether the length of m is $[\![l_{n^b}]\!]^\tau$ or not.
- For any bit string m and any name k,
  - if $[\![M]\!]([\![\pi_i]\!](m)) = 1$ for $i = 1, 2$, then $[\![M]\!]([\![\mathsf{dec}]\!](m,\ [\![\mathsf{dk}(k)]\!]^\tau)) = 0$,
  - if $[\![M]\!]([\![\mathsf{dec}]\!](m,\ [\![\mathsf{dk}(k)]\!]^\tau)) = 1$, then$[\![M]\!]([\![\mathsf{dec}]\!](m,\ [\![\mathsf{dk}(k')]\!]^\tau)) = 0$ for any name k' such that $[\![\mathsf{dk}(k)]\!]^\tau \neq [\![\mathsf{dk}(k')]\!]^\tau$,
  - for any $m \in \{0,1\}^*$ and any $m' \in \{0,1\}^* \setminus \{[\![\mathsf{dk}(k)]\!]^\tau\ :\ \tau \in \mathcal{SS}\}$, $[\![M]\!]([\![\mathsf{dec}]\!](m, m') = 0$.

These assumptions cover the case where m is not the computational interpretation of any term. They can be ensured, for instance, by assuming that the decryption of a ciphertext with a wrong key returns an error. The previous work on computational soundness has implemented this by appending each ciphertext with the encryption key used to produce the ciphertext.
- The implementation is strict: For any $f \in \mathcal{F}$ and any bit string m, $[\![M]\!](m) = 0$ implies $[\![M]\!]([\![f]\!](\cdots m \cdots)) = 0$. For instance, $[\![M]\!]([\![\{u\}^r_{\mathsf{ek}(k)}]\!]^\tau) = 0$ when $u = \mathsf{dec}(\{s\}^{r'}_{\mathsf{ek}(k)}, \mathsf{dk}(k'))$.
- For any two terms u, v, $[\![EQ]\!]([\![u]\!]^\tau, [\![v]\!]^\tau) = 1$ iff $[\![M]\!]([\![u]\!]^\tau) = [\![M]\!]([\![v]\!]^\tau) = 1$ and $[\![u]\!]^\tau = [\![v]\!]^\tau$.

For a computational soundness result, we assume nothing on the computational interpretation of the predicates EK, PL, HL, which may (not) be available to a computational attacker.

3.2 Interactive Turing machines

The processes are interpreted as interactive Turing machines, which we do not recall here. Let us only highlight the specifics of our model.

The model of the network includes a store that records the IDs (interpretation of channel names) associated with each process. This allows us to consider nested replications: the attacker may refer to a given replicated process in a deterministic way using such channel IDs.

More importantly, each basic process, upon receiving a message on a channel c, checks that the length of the input bit string matches the expected length. It proceeds only there is a match: the machine in state $c(x{:}\,l).B$ may move to the state B only if the content of its input tape has length $[\![l]\!]$.

Definition 2. Two protocols P and Q are *computationally indistinguishable*, which we write $P \approx Q$, if, for any attacker's machine $\mathcal{A}$,

$$|\mathbf{Pr}\,[\tau : [\![P]\!]^\tau \| \mathcal{A} = 1] - \mathbf{Pr}\,[\tau : [\![Q]\!]^\tau \| \mathcal{A} = 1]|$$

is negligible in the security parameter.

3.3 Cryptographic assumptions

We assume the public-key encryption scheme to be IND-CCA2 and the hash function to be preimage-resistant and collision-resistant. For instance, preimage-resistance is stated as follows.

Given a security parameter η, a *hash function* [14] is a deterministic algorithm $\mathcal{H}$ that, given a key $k \in K_\mathcal{H}$ and a bit string $m \in M_\mathcal{H}$, outputs a hash value of m by k, whose length only depends on η (not on m), where $K_\mathcal{H}$ is a key space and $M_\mathcal{H}$ is a message space such that $m' \in M_\mathcal{H}$ implies $\{0,1\}^{|m'|} \subseteq M_\mathcal{H}$, and that each bit string in $M_\mathcal{H}$ is so long that it cannot be guessed by the attacker (i.e.

there is a polynomial p such that $\eta \leq p(\min\{|m| \mid m \in M_{\mathcal{H}}\}))$. $\mathcal{H}$ is preimage-resistant if the following probability is negligible in η for any PPT attacker $\mathcal{A}$ and any ℓ such that $\{0,1\}^\ell \subseteq M_{\mathcal{H}}$:

$$\Pr[\, k \xleftarrow{\$} K_{\mathcal{H}} \,;\, s \xleftarrow{\$} \{0,1\}^\ell \,;\, m := \mathcal{H}(k,s) \,;\, s' \leftarrow \mathcal{A}(1^\eta, k, m) : \mathcal{H}(k,s') = m \,].$$

We also assume the following (which are necessary for the soundness result):

- The key certificates cannot be forged with a non-negligible probability.
- The length of a pair is longer than (or equal to) the sum of the lengths of its two components.
- The length of a ciphertext is strictly longer than the length of the corresponding plaintext.
- The length of a hash value (of a nonce) is strictly smaller than the length of the nonce. This rules out identities such as $\mathsf{h}(n) = n$ that could, otherwise, occur with a non-negligible probability.

4 The main result

Our main result states that we captured all distinguishing capabilities of a computational attacker in a symbolic model:

Theorem 1. *Let P and Q be two protocols. If $P \sim Q$, then $P \approx Q$.*

The rest of the paper is devoted to the sketch of the proof of this result. First we express the problem as the equivalent problem "$t_P \sim t_Q$ implies $t_P \approx t_Q$" where t_P is a *computation tree* that represents all possible execution sequences (see Section 4.1). Unlike [7], these trees have a finite outdegree, which is independent of the security parameter.

Next, we perform successive transformations T of the computation trees, transforming the problem into "$T(t_P) \sim T(t_Q)$ implies $T(t_P) \approx T(t_Q)$". The computation trees $T(t_P)$ are no longer the computation trees of processes and that is why we use the detour through computation trees. Let us consider some of these transformations in more details.

1. The first transformation aims at ensuring more properties of the computation trees and therefore enables the next step: we *partially unravel* the computation trees, unfolding some conditions (see Section 4.2). This transformation may yield computation trees whose branching degree depends on the security parameter. The difficulty lies in proving that they are still polynomially simulatable.
2. The second transformation is a classical one: thanks to the absence of key cycles, following the ordering on keys, we may replace plaintexts of encryptions by uncorrupted keys with a constant of the same length as the plaintext (see Section 4.3). This step requires the IND-CCA2 property of the public-key encryption scheme. After this step, $t \sim t'$ iff the total unravelings $U(t)$ and $U(t')$ of t and t' respectively (which are infinitely branching trees) are identical up to renaming, which we write $U(t) \simeq U(t')$.

3. The third transformation rules out coincidences (see Section 4.4): in the resulting computation tree, the conditions explicitly state that two distinct names are distinct and that two hash values of distinct terms are distinct. Though this is trivial in the symbolic model, it may happen by chance in the computational one. We also need here to rely on collision-resistance of the hash function.
4. The fourth transformation rules out guesses made in advance (see Section 4.5): in the resulting computation tree, the conditions state explicitly that a random term that has not been produced yet, cannot be computed. This is more tricky than it looks, and relies in particular on the assumption that the length of a pair is longer than the sum of the lengths of its components. It also rules out the computation of a nonce from its hash value, thanks to preimage-resistance.

After these transformations, we can conclude that $t \approx t'$, thanks to a trace mapping property (see Section 4.6).

4.1 Computation trees

A computation tree is a finitely branching tree whose nodes are labeled with pairs consisting a process (a state) and a frame and whose each edge is labeled with a variable, a channel name and a condition. For any node of a computation tree, given a variable x and a channel name c, the disjunction of all the conditions Φ such that (x, c, Φ) labels some edge departing from the node is a tautology and any two such conditions cannot be satisfied together.

We may associate a computation tree to any protocol: Roughly speaking, if a process $P_0 \| Q$ is structurally equivalent to $(\nu \overline{n})(c(x{:}\, l).P_1 \| Q)$ and $(\nu \overline{m})(P_0 \| Q, \phi)$ is labeling a node, we add an edge $(\nu \overline{m})\,(P_0 \| Q, \phi) \xrightarrow{x,c,\Phi} (\nu \overline{m}, \overline{n})(P_2 \| Q, \phi \uplus \phi')$ if there is a (sequence of) test Φ in P_1, whose satisfaction yields the output of ϕ' and the remaining process P_2.[4] Any symbolic trace (i.e., any sequence of triples (x, c, s) where x is a variable, c is a channel name and s is a ground term) that can be produced by an attacker process corresponds to an instance θ of a path in the process computation tree, such that, at any step as above, there is a term u such that $u\sigma_\phi\theta{\downarrow} = x\theta$ (in other words, $x\theta$ is deducible from the corresponding instance of the frame).

Given a sample τ, each computation tree t is also associated with a tree oracle $\mathcal{O}_{t,\tau}$: when the oracle is queried with a bit string m, a variable x and a channel ID $[\![c]\!]^\tau$, it evaluates (in the computational model) the conditions departing from the root node and associated with (x, c). Exactly one of them is satisfied: this corresponds to an edge $t \xrightarrow{x,c,\Phi} t'$. Then the oracle replies sending the computational interpretation of the frame labeling the root of t' and then behaves as $\mathcal{O}_{t',\tau}$. Two tree oracles are indistinguishable if no polynomial time

[4] The names that are bound in front of the state/frame cannot be renamed, unless they are renamed in the whole subtree.

attacker can guess with a significant advantage which of the two oracles he is interacting with.

Example 4. The computation tree of the protocol $P = \nu i_{\mathsf{in}}, i_{\mathsf{out}}, r, k.\ c_B(x).\ \overline{c_B}(\langle i_{\mathsf{in}}, i_{\mathsf{out}}\rangle)$. B is shown in Fig. 2 where $\bar{n} = i_{\mathsf{in}}, i_{\mathsf{out}}, r, k$ and B is the following basic process:

$$
\begin{array}{l}
i_{\mathsf{in}}(y).\mathsf{if}\ EQ(\pi_1(y), \mathsf{n}^1(r)) \\
\quad \mathsf{then}\ \ \mathsf{if}\ M(\mathsf{dec}(\pi_2(y),\ \mathsf{dk}(k))) \\
\quad\quad \mathsf{then}\ \overline{i_{\mathsf{out}}}(\mathsf{dec}(\pi_2(y),\ \mathsf{dk}(k))).\mathbf{0} \\
\quad\quad \mathsf{else}\ \mathbf{0} \\
\quad \mathsf{else}\ \mathbf{0}.
\end{array}
$$

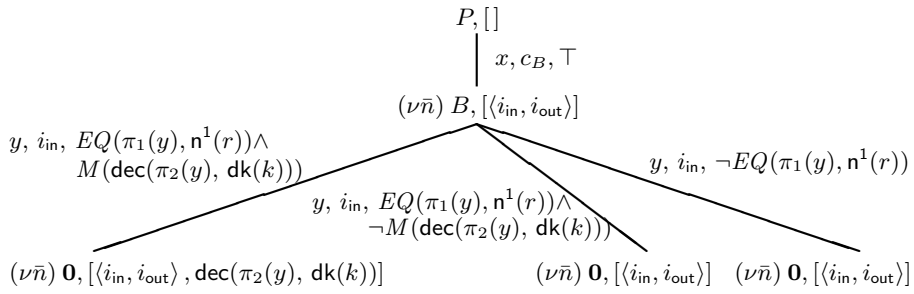

Fig. 2. Example of a computation tree

4.2 Partial unraveling

The goal is to replace plaintexts with fixed bit strings, thanks to IND-CCA2. However, in some cases, it would not be correct, because the terms occurring in the frames or the conditions may contain variables. For instance in $\mathsf{dec}(\{u\colon l\}^r_x, y)$, we may replace u with a fixed term of expected length l only if the instance of the term does not contain a redex, in other words unless $x = \mathsf{ek}(k)$ and $y = \mathsf{dk}(k)$ for some name k. The basic idea is to narrow these terms, which may require to split the conditions, depending on whether there is a key generated so far such that $x = \mathsf{ek}(k)$ and $y = \mathsf{dk}(k)$.

We prove that we can unravel a computation tree, in such a way that the resulting tree is both computationally and symbolically indistinguishable from the original tree and such that any occurrence of an encryption/decryption is *safe*: either the key is explicitly an encryption/decryption key that has been generated before, or the condition implies that this is not the case. Furthermore, we prove that the tree oracle can still be simulated in polynomial time.

For example, let us consider the computation tree t_1 shown in Fig. 3 that has only one subtree t_2. For brevity, the states and the bindings are omitted from Fig. 3.

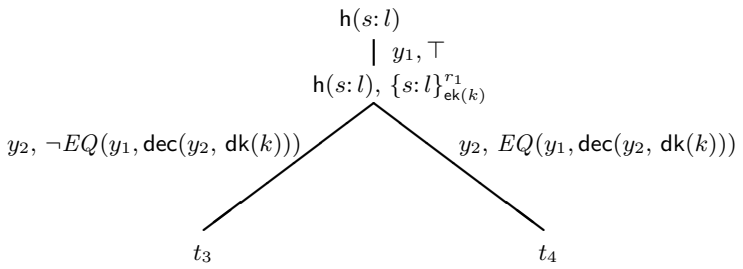

Fig. 3. Example of a computation tree t_1

The partial unraveling $U_{\mathsf{ek}}(t_1)$ of t_1 is shown in Fig. 4. In $U_{\mathsf{ek}}(t_1)$, the edge from t_2 to t_3 is split into the two edges to t_{30} and t_{31} such that $C_0 = \neg EQ(y_1, s) \wedge EQ(y_2, \{s{:}\,l\}^{r_1}_{\mathsf{ek}(k)})$ and $C_1 = \neg EQ(y_1, \mathsf{dec}(y_2,\ \mathsf{dk}(k))) \wedge \neg EQ(y_2, \{s{:}\,l\}^{r_1}_{\mathsf{ek}(k)})$. Similarly, the edge from t_2 to t_4 is split into the edges to t_{40} and t_{41}. Then $U_{\mathsf{ek}}(t_1)$ is a safe computation tree.

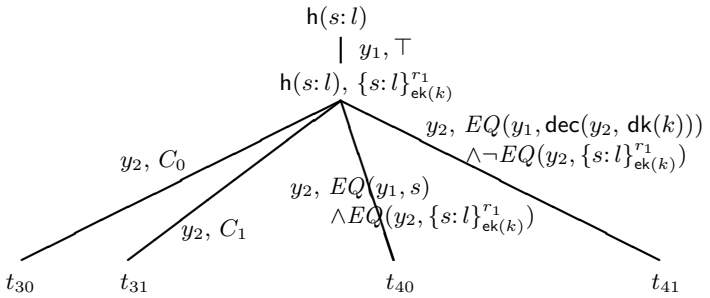

Fig. 4. The partial unraveling $U_{\mathsf{ek}}(t_1)$ of t_1

4.3 Replacing plaintexts

To such safe computation trees, we may apply the pattern function Ω, replacing each encryption $\{u{:}\,l\}^{r}_{\mathsf{ek}(k)}$ with $\{\square_l{:}\,l\}^{r}_{\mathsf{ek}(k)}$. This yields again a computation tree which is both symbolically and computationally indistinguishable from the original one, unless we break IND-CCA2.

For example, the tree $\Omega(U_{\mathsf{ek}}(t_1))$ is shown in Fig. 5 where $C_0' = \neg EQ(y_1, s) \wedge EQ(y_2, \{\square_l{:}\,l\}^{r_1}_{\mathsf{ek}(k)})$ and $C_1' = \neg EQ(y_1, \mathsf{dec}(y_2,\ \mathsf{dk}(k))) \wedge \neg EQ(y_2, \{\square_l{:}\,l\}^{r_1}_{\mathsf{ek}(k)})$. Note that the plaintext s inside the ciphertext $\{s{:}\,l\}^{r_1}_{\mathsf{ek}(k)}$ is replaced with the constant $\square_l$, whose expected length l is the same as s.

After this step, $t \sim t'$ iff the total unravelings of t and t' respectively (which are infinitely branching trees) are identical up to renaming.

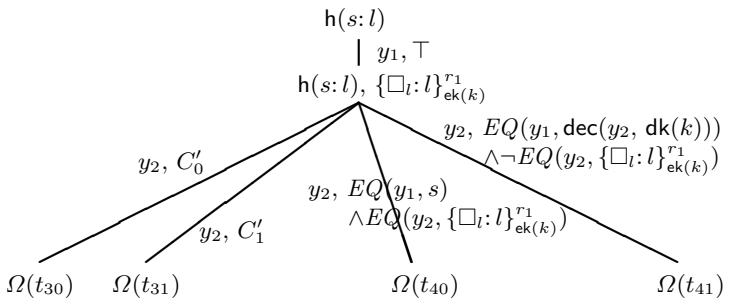

Fig. 5. $\Omega(U_{\mathsf{ek}}(t_1))$

4.4 Ruling out coincidences

Roughly speaking, we add to any condition Φ, labeling an edge of the computation tree, the conditions $\neg EQ(r, r')$ for distinct names r, r', as well as the conditions $\neg EQ(\mathsf{h}(u), \mathsf{h}(v))$ for distinct terms $\mathsf{h}(u), \mathsf{h}(v)$ that appear either in the current frame or condition. We also add similar constraints for keys. This relies, for instance, on collision-resistance.

4.5 Ruling out predictions

This is a bit more involved: we need to introduce new predicate symbols (for the purpose of the proof only). For instance, we consider a predicate $NP(K, u, v)$, which holds, given a substitution σ, if for any destructor context C using decryption keys in K, we have $C[u]\sigma\downarrow \neq v\sigma\downarrow$. We may express for instance that, when a name n is generated, if the currently available keys are in K, the last attacker input x cannot contain anything that depends on n. This is expressed by adding the constraint $NP(K, x, n)$. We use here preimage-resistance of the hash function, expressing that a nonce cannot be guessed from its hash.

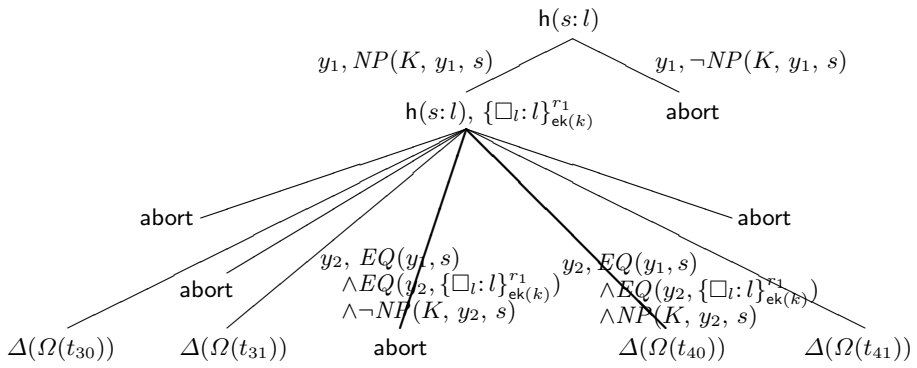

Fig. 6. $\Delta(\Omega(U_{\mathsf{ek}}(t_1)))$

For instance, if we apply the transformation Δ to $\Omega(U_{\mathsf{ek}}(t_1))$, ruling out predictions, the edge from $\Omega(U_{\mathsf{ek}}(t_1))$ to $\Omega(U_{\mathsf{ek}}(t_2))$ is split in two by adding $NP(K, y_1, s)$, which expresses the impossibility of computing the nonce s from y_1, or $\neg NP(K, y_1, s)$, and the edge from $\Omega(U_{\mathsf{ek}}(t_2))$ to $\Omega(t_{40})$ is split in two by adding $NP(K, y_2, s)$ or $\neg NP(K, y_2, s)$, as shown in Fig. 6. We omit details of the other edges that are split similarly. $\neg NP(K, y_1, s)$, for instance, implies $EQ(C[y_1], s)$ for some destructor context C. Hence the satisfaction of this condition implies the existence of an attacker on the preimage-resistance of the hash function; the execution aborts in this case.

4.6 Trace mapping

We show that the conditions resulting from all previous transformations are either unsatisfiable or else satisfiable both in the computational and in the symbolic model. It follows that, for every computational trace, which corresponds to a path in the computation tree and an assignment of variables that satisfies all the conditions along this path, there is also a symbolic trace that corresponds to the same path. This is what we call *trace mapping*. It is slightly different from the usual trace mapping, which states that every computational trace is an interpretation of a symbolic trace, with an overwhelming probability. First, thanks to our previous transformation steps, we get a property with the probability 1 (not with an overwhelming probability only). Next, we do not state that the computational trace is an interpretation of the symbolic one: we only state that they satisfy the same conditions. This is sufficient to conclude: thanks to trace mapping, for every sequence of attacker inputs s_i, if $\mathcal{O}_{t,\tau}$ replies the sequence $[\![u_r]\!]^\tau$, then there is a sequence of symbolic attacker inputs v_i yielding the sequence u_r of symbolic replies of $U(t)$. Now, since $U(t) \simeq U(t')$, there is also a sequence u'_r of replies of $U(t')$ such that, for some τ', $[\![u'_r]\!]^{\tau'} = [\![u_r]\!]^\tau$ is the sequence of replies of the oracle $\mathcal{O}_{t',\tau'}$ on the input sequence s_i. Hence $t \approx t'$. (And we do not need $[\![v_i]\!]^\tau = s_i$.)

5 Conclusion

We managed to get a computational soundness result for observational equivalence, without any parsing assumption. This result holds for a large subset of the applied π-calculus. We believe that the same method can be applied to other primitives, though, as this proof shows, it might be long and tedious.

However, we learned several lessons from this work. For instance, we attacked the problem of the symbolic length from another angle, using a trade-off between computational assumptions and assumptions on the protocols. We also showed how collision-resistance (resp. preimage-resistance) of hash functions can be used in soundness proofs. And maybe, more importantly, we identified several assumptions that look necessary for soundness results, showing the limitations of the method.

Acknowledgments

We thank Dominique Unruh and Véronique Cortier for valuable discussions. We also thank the anonymous reviewers for helpful comments.

References

1. Martín Abadi and Cédric Fournet. Mobile values, new names, and secure communication. In *Proc. of the 28th ACM Symposium on Principles of Programming Languages (POPL'01)*, pages 104–115, 2001.
2. Martín Abadi and Phillip Rogaway. Reconciling two views of cryptography (the computational soundness of formal encryption). *Journal of Cryptology*, 15(2):103 – 127, 2002.
3. Michael Backes, Dennis Hofheinz, and Dominique Unruh. CoSP: A general framework for computational soundness proofs. Cryptology ePrint Archive, Report 2009/080, 2009. `http://eprint.iacr.org/`.
4. Michael Backes, Birgit Pfitzmann, and Michael Waidner. A composable cryptographic library with nested operations. In *Proc. of the 10th ACM Concerence on Computer and Communications Security (CCS'03)*, pages 220–230, 2003.
5. Michael Backes, Birgit Pfitzmann, and Michael Waidner. Limits of the BRSIM/UC soundness of Dolev-Yao models with hashes. In *Proc. of 11th European Symposium on Research in Computer Security (ESORICS'06)*, volume 4189 of *Lecture Notes in Computer Science*, pages 404–423, 2006.
6. Michael Backes, Birgit Pfitzmann, and Michael Waidner. The reactive simulatability (RSIM) framework for asynchronous systems. *Information and Computation*, 205(12):1685–1720, 2007.
7. Hubert Comon-Lundh and Véronique Cortier. Computational soundness of observational equivalence. In *Proc. of the 15th ACM Conference on Computer and Communications Security (CCS'08)*, pages 109–118, 2008.
8. Véronique Cortier, Steve Kremer, Ralf Küsters, and Bogdan Warinschi. Computationally sound symbolic secrecy in the presence of hash functions. In *Proceedings of the 26th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'06)*, volume 4337 of *Lecture Notes in Computer Science*, pages 176–187. Springer, 2006.
9. Véronique Cortier and Bogdan Warinschi. Computationally sound, automated proofs for securityprotocols. In *Proc. of 14th European Symposium on Programming (ESOP'05)*, volume 3444 of *Lecture Notes in Computer Science*, pages 157–171, 2005.
10. Flavio D. Garcia and Peter van Rossum. Sound and complete computational interpretation of symbolic hashes in the standard model. *Theor. Comput. Sci.*, 394(1-2):112–133, 2008.
11. Romain Janvier, Yassine Lakhnech, and Laurent Mazaré. Computational soundness of symbolic analysis for protocols using hash functions. *Electr. Notes Theor. Comput. Sci.*, 186:121–139, 2007.
12. Yusuke Kawamoto, Hideki Sakurada, and Masami Hagiya. Computationally sound symbolic anonymity of a ring signature. In *Proc. of Joint Workshop on Foundations of Computer Security, Automated Reasoning for Security Protocol Analysis and Issues in the Theory of Security (FCS-ARSPA-WITS'08)*, pages 161–175, 2008.

13. Daniele Micciancio and Bogdan Warinschi. Soundness of formal encryption in the presence of active adversaries. In *Proc. of Theory of Cryptography Conference (TCC'04)*, volume 2951 of *Lecture Notes in Computer Science*, pages 133–151, 2004.
14. Phillip Rogaway and Thomas Shrimpton. Cryptographic hash-function basics: Definitions, implications, and separations for preimage resistance, second-preimage resistance, and collision resistance. In *Proc. of Fast Software Encryption (FSE)*, volume 3017 of *Lecture Notes in Computer Science*, pages 371–388, 2004.
15. Dominique Unruh. Termination-insensitive computational indistinguishability (and applications to computational soundness). In *Proc. of the 24th IEEE Computer Security Foundations Symposium (CSF 2011)*. IEEE Computer Society, June 2011.