Indistinguishability Beyond Diff-Equivalence in ProVerif

Vincent Cheval
Inria Paris, France

Itsaka Rakotonirina
MPI-SP, Germany

***Abstract*—When formalising cryptographic protocols, privacy-type properties such as strong flavours of secrecy, anonymity or unlinkability, are often modelled by indistinguishability statements. Proving them is notoriously more challenging than trace properties which benefit from a well-established tool support today. State-of-the-art techniques often exhibit significant limitations, e.g., consider only a bounded number of protocol sessions, or prove *diff-equivalence*—a fine-grained, structure-guided notion of indistinguishability that commonly yields unnecessarily pessimistic analyses.**

In this paper, we design, implement and evaluate the first general framework for proving indistinguishability properties, for an unbounded number of protocol sessions, going beyond the scope of diff-equivalence. For that we relax the structural requirements of ProVerif, a state-of-the-art tool, through a notion of *session decomposition*, intuitively allowing a dynamic restructuration of the proofs. We can then verify in a modular way various, more realistic models of indistinguishability such as may-testing equivalence, by exhibiting for each relation a sufficient condition on ProVerif's output ensuring that it holds. We implement our approach into a prototype and showcase the gain in scope through several case studies.

1. Introduction

Cryptographic protocols are distributed programs enforcing the security of sensitive communications between several heterogeneous entities. Due to their distributed nature and their execution over untrusted networks, they exhibit complex behaviours making their analysis prohibitively tedious. Security properties of interest commonly include *trace properties* such as (weak) secrecy or authentication, benefiting today from a strong automation support. One may typically cite the ProVerif [1] and Tamarin [2] tools, that are able to handle full-fledged industrial protocols such as, TLS 1.3 [3], [4], [5], the 5G standard [6] or Signal [7], [8].

Some classical security notions are however modelled by more complex (hyper)properties including *indistinguishability*. They take the form of equivalence relations in concurrent process algebra such as the applied π-calculus [9], and capture strong privacy-type properties, like anonymity or unlinkability. Consider for example the prototypical scenario of a Basic Access Control protocol (BAC): reading devices at an airport interact with RFID chips of travelers' e-passports. We consider a process $S(k) = P(k) \mid R(k)$, where P and R (left abstract here for simplicity) are, respectively, processes modelling a passport and a reader, that have agreed on an identity-related value k. The unlinkability of identities across multiple sessions is formalised by a statement:

$$! \,\mathsf{new}\, k; !\, S(k) \;\approx\; !\,\mathsf{new}\, k; S(k) \qquad (1)$$

The ! (*replication*) models an unbounded number of copies of the subsequent process, $\mathsf{new}\, k$ indicates a fresh value k, and $\approx$ is an equivalence relation modelling the indistinguishability of two hypothetical scenarios, from the point of view of an active adversary who controls the network. Rephrasing, (1) models the impossibility to observe a difference between situations where all passports are different (right-hand side), and where some of them may interact multiple times with readers (left-hand side). In particular, a tool-assisted proof would have to accommodate for replicated processes of significantly different structures.

Yet, most analysis techniques for such properties are either limited to a bounded number of copies of S as in [10], or to *diff-equivalence* as in ProVerif, Tamarin or Maude-NPA [11]. Diff-equivalence is a fine-grained relation relying on strong structural assumptions on the processes to be proved equivalent, significantly hindering its application scope. It is typically insufficient to prove unlinkability statements such as (1), except using specialised tools dedicated to specific types of protocols [12], [13]. Diff-equivalence based proofs unsuccessfully attempt to *statically* match each (duplicated) instance of P and R from the right process, each with a (fresh) indistinguishable instance from the left process. Successful proofs require *dynamic* extensions of this argument, where this matching is different for each execution of each instance of P and R. Such data-dependent arguments are arguably beyond the capabilities of most protocol analysers for an unbounded number of sessions.

Contributions. We extend ProVerif to support modular proofs of may-testing and observational equivalences, and their pre-orders, for an unbounded number of sessions.

1) We introduce a notion of *session decomposition*, inspired from symmetry reductions in the bounded case [10]. Used as a substitute to ProVerif's internal representation of processes, it weakens the tool's requirements on their structure, to express coarser-grained security relations.
2) We then adapt ProVerif's approach—namely, converting processes into a set of Horn clauses—to account for session decompositions. The resulting clauses are saturated by ProVerif's usual solver, and we exhibit sufficient conditions on the saturated output; each ensures

one relation (may-testing, observational equivalence, preorders...) on the initial processes. This shows in particular the modularity of our notion of session decomposition in that it is not tied to a specific equivalence relation.

3) We propose for each of the above-mentioned conditions a general, semantic version, as well as a refinement better suited for automated verification. We then implement the overall procedure and use it to successfully verify various examples escaping the current scope of ProVerif.

Proofs have been omitted for the sake of readability and can be found in our technical report, with implementation files and benchmarks [14]. All files have also been submitted as supplementary material.

Related Work. Proofs of equivalence properties in security protocols roughly follow two trends, commented below.

On the one hand, proofs for a bounded number of sessions put an emphasis on precision and termination of the analysis at the cost of not supporting replication [15], [16], [17]. Among these, the SAT-Equiv tool may notably yield proofs for an unbounded number of sessions occasionally, but also imposes syntactic restrictions on processes that ProVerif is not subject to. We also mention [10], which inspired our notion of session decomposition: it develops symmetry-based techniques for trace equivalence that allows them to prove efficiently unlinkability properties. Our approach is however more widely applicable, due to our support for unbounded processes, but also through the multiple security relations we handle. However, by being only based on sufficient conditions, our prototype is currently unable to exhibit attack traces, i.e., to disprove equivalence.

On the other hand, for an unbounded number of sessions, most tools only support indistinguishability notions in the spirit of diff-equivalence, typically ProVerif [1], Tamarin [2] or Maude-NPA [11]. Although an extension of ProVerif partially relaxes the underlying structural requirements [18], it is subsumed by our current approach in terms of scope. [18] also yields algorithmically less efficient procedures on examples where the two approaches overlap: its procedure can be seen as a *static* enumeration of all potential session decompositions, followed by a tentative of proof for each of them. On the contrary, our native integration of session decompositions makes our single-proof procedure lighter to process. Other extensions allow some form of dynamic reasoning for protocols with synchronisation [19]. Annotating processes with synchronisation points thus makes proofs beyond diff-equivalence possible; this however requires manual annotations, is limited in scope (synchronisations are prohibited under replications), and requires synchronisation assumptions not arising in our framework.

An important related work is Ukano [12], a frontend to an altered version of ProVerif exhibiting sufficient conditions under which the decision of trace equivalence can be reduced to a combination of diff-equivalence and trace properties. These conditions are however only valid for specific properties of a syntactically restricted class of protocols. In particular, although Ukano is applicable to most examples we specifically consider in this paper, our more general and modular approach significantly broadens the technique's scope by not being subject to such restrictions (more details discussed in Section 7.2). The approach has also been carried in Tamarin [13] to analyse stateful protocols.

2. Unlinkability As a Motivating Example

We give an overview of our results by outlining a proof of unlinkability of a simplified BAC e-passport protocol in our model. This will give an insight of the proof techniques we develop in the subsequent technical sections.

Modelling. We consider an access control protocol where a passport P communicates with a reader R, after a preliminary chip reading allowing the reader to retrieve a passport secret k. Subsequent cryptographic operations are described by *function symbols* senc and sdec, verifying the identity $\mathsf{sdec}(\mathsf{senc}(x,y,z),z) \;\to\; x$. An expression $\mathsf{senc}(m,r,k)$ models the symmetric encryption of a message m using a key k and randomness r, whereas $\mathsf{sdec}(u,k)$ models the decryption of u with k. A minimal version of the BAC protocol [20] can then be described as follows, first in informal Alice-Bob notation:

$$\begin{array}{lll} P \to R: & n & \\ R \to P: & \mathsf{senc}(n,r,k) & \text{received as } x \\ P \to R: & \mathsf{ok} & \text{if } \mathsf{sdec}(x,k) = n \\ & \mathsf{error} & \text{otherwise} \end{array}$$

That is, P sends a freshly generated nonce n to R in clear as a challenge, and awaits as a response an encryption of n under their shared secret k. Then, the result of the protocol (ok or error) is output. This protocol is assumed to be carried out over an untrusted network, compromised by an adversary that may intercept messages, but also replay, forge or inject them. The desired unlinkability property is then:

After an arbitrary number of sessions of the protocol, the adversary cannot infer whether a same passport took part to several of these sessions.

This security property should hold despite the adversary being able, e.g., to change the recipient of some messages and observe the resulting final ok/error message. Formally, the roles of P and R are represented by *processes* of the applied π-calculus, written:

$$\begin{aligned} P(k) =\; & \mathsf{new}\, n; \mathsf{out}(c,n); \mathsf{in}(c,x); \\ & \mathsf{if}\ \mathsf{sdec}(x,k) = n\ \mathsf{then}\ \mathsf{out}(c,\mathsf{ok}); 0 \\ & \mathsf{else}\ \mathsf{out}(c,\mathsf{error}); 0 \\ R(k) =\; & \mathsf{new}\, r; \mathsf{in}(d,y); \mathsf{out}(d, \mathsf{senc}(y,r,k)); 0 \end{aligned}$$

Here the 0 indicates a terminated process. The instructions $\mathsf{in}(u,x)$ and $\mathsf{out}(u,v)$ model inputs and outputs on a communication channel u. The $\mathsf{new}\, n$ formalises the fresh generation of a name n, unknown to the adversary until output on the channel c. The formulation of unlinkability we consider is the *equivalence* $S_{\mathit{left}} \approx S_{\mathit{right}}$, with:

$$\begin{aligned} S_{\mathit{left}} &= !\, \mathsf{new}\, k; !\, (P(k) \mid R(k)) \\ S_{\mathit{right}} &= !\, \mathsf{new}\, k; (P(k) \mid R(k)) \end{aligned}$$

where $A \mid B$ is the parallel composition of A and B, and $!\,A$ is the parallel composition of an unbounded number of copies of A. The right side of the equivalence is a process modelling a scenario where all sessions involve different passports, whereas the left side may involve several times the same passports in different sessions. The equivalence relation $\approx$ may then be chosen among several common choices such as *observational equivalence* $\approx_o$ or *may-testing equivalence* $\approx_m$ to model adversarial indistinguishability.

Instrumentation. The first step of our approach to prove $S_{\mathit{left}} \approx S_{\mathit{right}}$ is to convert processes into *instrumented processes*. This internal representation of ProVerif intuitively interprets parallel processes as indexed sequences of processes. E.g., the process S_{right} is translated into the form:

$$\{\!\!\{ !_i^{o_1[i]} P_{\mathit{right}} \}\!\!\} \mid \{\!\!\{ !_i^{o_2[i]} R_{\mathit{right}} \}\!\!\}$$

with P_{right} and R_{right} respective translations of P and R. The multiset notation $\{\!\!\{\cdot\}\!\!\}$ (here, only singletons) indicates a set of parallel processes with a similar structure. Processes with a different structure are put in different multisets separated by a $\mid$ operator. This is what we call a *session decomposition* in this paper. Inside these multisets, an annotated replication $!_i^o P$ can be seen as a collection of copies of P, indexed by the variable i, and uniquely identified during the analysis by the *occurrence o*. In particular, P_{right} and R_{right} are mostly identical to P and R, except that they have their private names freshly renamed and indexed by i, e.g. n into $n_r[i]$. The argument i indicates a dependency, expressing that $n_r[i]$ is a different fresh name for each different copy of the process (i.e., for each instance of i). This thus makes the new instruction superfluous. For example:

$$R_{\mathit{right}} = \mathsf{in}^{o_r[i]}(d, y); \mathsf{out}(d, \mathsf{senc}(y, r_r[i], k_r[i])); 0$$

Again, the occurrence $o_r[i]$ is used for making reference to the tagged instruction (here, the input) during the analysis later. So far, most of this material exists in ProVerif; the novel ingredient is that, when translating S_{left}, we obtain a session decomposition similar to the previous one, despite the different initial number of replications:

$$\{\!\!\{ !_{i,j}^{o_3[i,j]} P_{\mathit{left}} \}\!\!\} \mid \{\!\!\{ !_{i,j}^{o_4[i,j]} R_{\mathit{left}} \}\!\!\}$$

The process S_{left} can indeed also be seen as a collection of copies of P and R, albeit for different data dependencies. This is reflected in this session decomposition by the double index $[i, j]$, where i indexes to the first replication and j the second. The indexation of R is therefore this time:

$$R_{\mathit{left}} = \mathsf{in}^{o_\ell[i,j]}(d, y); \mathsf{out}(d, \mathsf{senc}(y, r_\ell[i, j], k_\ell[i])); 0$$

In short, instead of requiring that two equivalent processes have (syntactically) the same structure, our analysis criterion is based on the components (multisets) of the session decomposition, regardless of their sizes. Theorem 1 shows that instrumentation preserves process equivalences.

Translation Into Clauses. Once the session decomposition is over, the algorithm translates the instrumented processes into Horn clauses (Section 5):

$$F_1 \wedge \cdots \wedge F_n \rightarrow F$$

These clauses intuitively model ProVerif's internal reasoning and include atomic formulae of various forms. Some clauses, already present in the baseline version of ProVerif, describe for example the adversary's capabilities, such as:

$$\mathrm{input}(x, y) \wedge \mathrm{msg}(x, z, y', z') \wedge y \neq y' \rightarrow \mathrm{bad}$$

The fact $\mathrm{input}(x, y)$ models that the attacker can listen on the channel x in an execution of S_{left}, and y in an equivalent execution of S_{right}. The clause represents the fact that the attacker can distinguish the two executions (fact bad) when a message was sent on x in S_{left} but on a different channel y' in S_{right} (fact $\mathrm{msg}(x, z, y', z')$). Other facts $\mathrm{ev}(e, e')$ indicates that an *event* e occurs in S_{left}, and simultaneously e' occurs in S_{right}. Also, importantly, $\mathrm{att}(u, v)$ indicates that the adversary is able to compute u (resp. v) using the knowledge accumulated in S_{left} (resp. S_{right}) by intercepting outputs. Finally, we introduce in this work novel events $\mathit{repl}(o)$ (*replication events*), indicating that a replication labelled with the occurrence o is unfolded. We typically use them to express correspondences between occurrences (e.g., o_3 and o_1 in the example), which in turn helps capturing our notion of session decomposition within clauses. For example, the following clause describes the execution of the first output of P in S_{left} and S_{right}

$$\mathrm{ev}(\mathit{repl}(o_3[i, j]), \mathit{repl}(o_1[i'])) \rightarrow \mathrm{att}(n_l[i, j], n_r[i'])$$

Rephrasing, when an occurrence of the replication $o_3[i, j]$ is executed in S_{left}, while $o_1[i']$ is executed in S_{right}, the adversary learns the corresponding values of n (as they are publicly output). Note in particular that the premise of the clause relates events occurring at different structural levels of processes (i.e., o_3 is under two nested replications, unlike o_1), which is not a natural feature of diff-equivalence.

Verification. After generating all clauses, they are saturated using ProVerif's internal solver. When none of the saturated clauses concludes bad, we can directly conclude that diff-equivalence holds, as in ProVerif. The key novelty of our approach is that even when some saturated clauses conclude bad, we may still be able to prove equivalences as follows.

Intuitively, a clause $H \rightarrow \mathrm{bad}$ indicates that there may be distinguishable executions of S_{left} and S_{right} that satisfy H. Conversely, the *saturation* procedure of ProVerif ensures that if some executions of S_{left} and S_{right} are distinguishable then there must exist a saturated clause $H \rightarrow \mathrm{bad}$ where H is satisfied by these executions (Theorem 2). Thus, to prove may-testing inclusion $\sqsubseteq_m$ for example, it suffices to build a *session matching* (Section 6.1) mapping the replication events of any execution of S_{left} to the replication occurrences of S_{right}, that can *falsify* (Section 6.2) the hypotheses H of all saturated clauses $H \rightarrow \mathrm{bad}$. In our example, it means for instance mapping instantiations of $\mathit{repl}(o_3[i, j])$ to adequate instantiations of $o_1[i']$.

In other words, our conditions imply that all executions of S_{left} can be matched by an execution of S_{right} that satisfy none of the hypotheses H of the clauses concluding bad. This thus entails that these two executions are not distinguishable (Theorem 3). We exhibit similar conditions implying observational equivalence $\approx_o$ and its pre-order $\sqsubseteq_o$ (simulation) [14]. In our implementation, we can let the prototype prove the "best" (i.e., finest) relations it can. In our example, the tool automatically shows that $S_{left} \sqsubseteq_m S_{right}$ and $S_{right} \sqsubseteq_o S_{left}$, which appears to be the optimal result.

3. Model

In this section, we present the process calculus and the notions of equivalence studied in this paper. Most of it is standard material from the theory of ProVerif.

3.1. Syntax

We assume a classical term algebra, that is, an infinite set of *variables* $\mathcal{V}$, an infinite set of *names* $\mathcal{N} = \mathcal{N}_{\mathsf{pub}} \uplus \mathcal{N}_{\mathsf{prv}}$ representing atomic values (public and private, respectively), and a finite set of *function symbols* (with their arity) called a *signature*. Specifically, we consider two distinct sets $\mathcal{F}_d$ and $\mathcal{F}_c$ representing, respectively, *constructor* symbols (used to build messages, e.g., encryption or concatenation) and *destructor* symbols (representing operations on messages that may fail, e.g., decryption). Functions, names and variables can be combined to form *expressions*:

$$\begin{array}{lll} D ::= & a & \textit{atomic value } a \in \mathcal{N} \cup \mathcal{V} \\ & f(D_1, \dots, D_k) & \textit{application } f \in \mathcal{F}_d \cup \mathcal{F}_c \\ & \mathsf{fail} & \textit{failure} \end{array}$$

An expression is called a *term* when it does not contain destructors or the fail symbol, and is then referred by the grammar tokens M, N. Terms are therefore expressions that correspond, intuitively, to successful computations. We then consider *processes* modelling concurrent programs:

$$\begin{array}{lll} P, Q ::= & 0 & \textit{nil} \\ & \mathsf{out}(N, M); P & \textit{output} \\ & \mathsf{in}(N, x); P & \textit{input } (x \in \mathcal{V}) \\ & \mathsf{event}(M); P & \textit{event} \\ & P \mid Q & \textit{parallel composition} \\ & !\,P & \textit{replication} \\ & \mathsf{new}\, n; P & \textit{restriction } (n \in \mathcal{N}) \\ & \mathsf{let}\ x = D\ \mathsf{in}\ P\ \mathsf{else}\ Q & \textit{assignment} \end{array}$$

We already discussed most of these constructions during the motivation example, at the exception of events. Events are ignored during equivalence proofs, although they can be used in ProVerif to specify *axioms*, introduced in [21], that are trace properties guiding the internal decision procedure. They are admitted during verification, but those mentioned and introduced in this paper are (protocol-independent) properties that have been priorly proved manually.

The assignment instruction $\mathsf{let}\ x = D\ \mathsf{in}\ P\ \mathsf{else}\ Q$ attempts to evaluate D and executes $P(x)$ with the resulting value x in case of success, or executes a default process Q in case of failure. Assuming a symbol $\mathsf{Equals} \in \mathcal{F}_d$ with the rewrite rule $\mathsf{Equals}(x, x) \to \mathsf{ok}$, assignments can therefore be used to encode conditionals as in the motivation example.

The notion of evaluation underlying assignments is formalised through a set of *rewrite rules* $\ell \to r$, such as the rule $\mathsf{sdec}(\mathsf{senc}(x, y, z), z) \to x$ used in the motivation example. Formally, we refer to the usual notion of substitutions $\sigma = \{^{M_1}/_{x_1}, \dots, ^{M_n}/_{x_n}\}$, i.e., we write $M\sigma$ the term where all syntactic occurrences of x_i in M are replaced by M_i. We naturally extend the application of a substitution to expressions, processes, etc. ProVerif then operates using the *evaluation* of expressions $D \Downarrow U$ where U is either a term M or the constant fail. This evaluation is based on a standard notion of rewriting system normalising expressions with a call-by-value strategy (full details in Appendix A).

3.2. Operational Semantics

The semantics of processes characterises their behaviour when executed in a hostile environment modelling an adversary controlling the communication network. It operates on *configurations* $\mathcal{P}, \Phi$, which are tuples where $\mathcal{P}$ is a multiset of processes modelling all processes currently executed in parallel, and Φ is a called a *frame*, i.e., a substitution:

$$\Phi = \{^{M_1}/_{\mathsf{ax}_1}, \dots, ^{M_n}/_{\mathsf{ax}_n}\}$$

Intuitively, a frame records the knowledge obtained by the adversary by spying on outputs sent on the communication network. An entry $^{M_i}/_{\mathsf{ax}_i}$ indicates in particular that the adversary can access the value of M_i through the *handle* ax_i, which is a dedicated type of variable. In particular, the following notion formalises attacker's computations:

Definition 1 (Recipe)**.** A *recipe* ξ is an expression without private names, and with no variables except handles.

For example, a frame $\Phi = \{^{\mathsf{senc}(m,k)}/_{\mathsf{ax}_1}, ^{k}/_{\mathsf{ax}_2}\}$ with $m, k \in \mathcal{N}_{\mathsf{prv}}$ indicates that the attacker observed, successively, $\mathsf{senc}(m, k)$ and k. The term m can be thus computed by the adversary using the recipe $\xi = \mathsf{sdec}(\mathsf{ax}_1, \mathsf{ax}_2)$; that is, $\xi\Phi \Downarrow m$. Note in particular that the decryption key k is not used directly (as $\mathsf{sdec}(\mathsf{ax}_1, k)$ is not a valid recipe because it contains the private name k), but by reference through the handle ax_2. The actual semantics is then defined in Figure 1, as a labelled transition relation $\xrightarrow{\alpha}$ over configurations, where the label α is called an *action*:

- $\mathsf{in}(\xi, \zeta)$ materialises an input fetched from the adversary, where the input term is computed by the recipe ζ, and the underlying channel N is public in that it can be computed by the adversary through recipe ξ;
- $\mathsf{out}(\xi, \mathsf{ax})$ materialises an output sent on a channel publicly computable through ξ, and added to the adversary's knowledge as a frame entry at handle ax;
- and finally, events make the corresponding term appear as the transition label, and empty labels are used for miscellaneous rules without visible behaviours.

For the sake of readability, Figure 1 omits the rule for *internal communications*, that models a synchronous, private, and unobservable communication between an output and an input. This simplifies some later technical definitions, while handling such communications does not involve substantially novel ideas compared to regular communications.

$$
\begin{array}{ll}
\{\!\{0\}\!\}, \Phi \to \emptyset, \Phi & (\textsc{Nil}) \\
\{\!\{P \mid Q\}\!\}, \Phi \to \{\!\{P, Q\}\!\}, \Phi & (\textsc{Par}) \\
\{\!\{!P\}\!\}, \Phi \to \{\!\{P, !P\}\!\}, \Phi & (\textsc{Repl}) \\
\{\!\{\mathsf{new}\, a; P\}\!\}, \Phi \to \{\!\{P\{^{a'}/_{a}\}\}\!\}, \Phi & \\
\quad \text{if } a' \in \mathcal{N}_{\mathsf{prv}} \smallsetminus names(\mathcal{P}, P, \Phi) & (\textsc{Restr}) \\
\{\!\{\mathsf{out}(N, M); P\}\!\}, \Phi \xrightarrow{\mathsf{out}(\xi, \mathsf{ax})} \{\!\{P\}\!\}, \Phi' & \\
\quad \text{if } \Phi' = \Phi \cup \{^{M}/_{\mathsf{ax}}\},\ \mathsf{ax} \notin dom(\Phi),\ \xi\Phi \Downarrow N & (\textsc{Out}) \\
\{\!\{\mathsf{in}(N, x); Q\}\!\}, \Phi \xrightarrow{\mathsf{in}(\xi, \zeta)} \{\!\{Q\{^{M}/_{x}\}\}\!\}, \Phi & \\
\quad \text{if } \xi, \zeta \text{ recipes such that } \xi\Phi \Downarrow N \text{ and } \zeta\Phi \Downarrow M & (\textsc{In}) \\
\{\!\{\mathsf{event}(M); P\}\!\}, \Phi \xrightarrow{M} \{\!\{P\}\!\}, \Phi & (\textsc{Event}) \\
\{\!\{\mathsf{let}\ x = D\ \mathsf{in}\ P\ \mathsf{else}\ Q\}\!\}, \Phi \to \{\!\{R\}\!\}, \Phi & \\
\quad \text{if } R = P\{^{M}/_{x}\} \text{ if } D \Downarrow M;\ R = Q \text{ if } D \Downarrow \mathsf{fail} & (\textsc{Let}) \\
\mathcal{P} \cup \mathcal{Q}, \Phi \xrightarrow{\alpha} \mathcal{P}' \cup \mathcal{Q}, \Phi' \qquad \text{if } \mathcal{P}, \Phi \xrightarrow{\alpha} \mathcal{P}', \Phi' & (\textsc{Cont})
\end{array}
$$

Figure 1: Operational Semantics Between Configurations (simplified: no internal communications)

The proofs of our theorems for the full semantics can be found in [14].

Definition 2 (Trace)**.** A *trace* T of a configuration $\mathcal{C}_0$ is a sequence of transitions of Figure 1 starting from $\mathcal{C}_0$, written $T : \mathcal{C}_0 \xrightarrow{\alpha_1} \cdots \xrightarrow{\alpha_n} \mathcal{C}_n$. We may write $\mathcal{C}_0 \xRightarrow{\alpha_1 \cdots \alpha_n} \mathcal{C}_n$ when intermediary processes are not relevant. Note that in the work $\alpha_1 \cdots \alpha_n$, empty α_i, i.e., transitions without labels, are implicitly omitted. By extension, we may consider traces of a process P by interpreting P as the configuration $\{\!\{P\}\!\}, \emptyset$.

3.3. Security Properties

We now define the notions of process equivalence considered in this paper. They are typically used to model strong flavours of privacy-type properties of cryptographic protocols expressed in our process algebra. They are intuitively built over a notion of *static indistinguishability* that, intuitively, formalises that the adversary cannot compute a test that tells two given histories of observables apart. In the following, we implicitly assume that the signature includes a symbol binary symbol Equals defined by $\mathsf{Equals}(x, x) \to \mathsf{ok}$ that allows the adversary to run equality tests to violate indistinguishability. This static notion is then lifted to dynamic behaviours through the operational semantics. For that we consider a relation over words actions: we write

$$\alpha_1 \cdots \alpha_n \equiv \beta_1 \cdots \beta_p$$

if the two words $\alpha_1 \cdots \alpha_n$ and $\beta_1 \cdots \beta_p$ become identical after removing all event actions from them. Not considering event actions in the definition of equivalences models that they are rather annotations from the modeller than observables of the attacker. They are, instead, convenient to express axioms, that take in this paper the form of trace properties.

Definition 3 (Observational equivalence)**.** *Observational pre-order* (or *simulation*) $\sqsubseteq_o$ is defined as the largest relation $\mathcal{R}$ over configurations such that $\mathcal{C}\ \mathcal{R}\ \mathcal{C}'$ implies, if we write $\mathcal{C} = \mathcal{P}, \Phi$ and $\mathcal{C}' = \mathcal{P}', \Phi'$:

1) for all recipes ξ, $\xi\Phi \Downarrow \mathsf{fail}$ *iff* $\xi\Phi' \Downarrow \mathsf{fail}$;
2) if $\mathcal{C} \xrightarrow{\alpha} \mathcal{C}_1$ then there exists $\mathcal{C}_1'$ such that $\mathcal{C}' \xRightarrow{w} \mathcal{C}_1'$, $\alpha \equiv w$, and $\mathcal{C}_1\ \mathcal{R}\ \mathcal{C}_1'$.

Observational equivalence $\approx_o$ is defined as the largest *symmetric* relation $\mathcal{R}$ that satisfies properties 1, 2.

The usual definition of observational equivalence is context-based but often harder to handle in proofs. The above definition is a standard, more operational but equivalent characterisation called bisimilarity [9]. The following, coarser notion of equivalence, *may testing*, and corresponds to the indistinguishability of sets of traces. It intuitively states for any trace of either process, and for any computation the adversary may additionally do, an indistinguishable sequence of actions can be taken in the other process.

Definition 4 (May testing)**.** *May testing inclusion* between configurations, denoted $\mathcal{C} \sqsubseteq_m \mathcal{C}'$, holds when for all traces $\mathcal{C} \xRightarrow{w} (\mathcal{P}, \Phi)$ and for all sets of recipes S, there exists a trace $\mathcal{C}' \xRightarrow{w'} (\mathcal{P}', \Phi')$ such that $w \equiv w'$ and for all $\xi \in S$, $\xi\Phi \Downarrow \mathsf{fail}$ *iff* $\xi\Phi' \Downarrow \mathsf{fail}$. *May testing equivalence*, denoted $\approx_m$, is defined as $\sqsubseteq_m \cap \sqsupseteq_m$.

4. Instrumentation

From this section, we describe our approach to prove equivalence properties and their respective pre-orders. For the sake of readability, the presentation focuses on may-testing but verifying the other relations involves analogues techniques (details in [14]). Recalling the motivation example, the first step of our procedure is to convert processes into *instrumented* ones that record some data dependencies, and more importantly, compute *session decompositions* to materialise the internal symmetries of processes.

4.1. Instrumented Processes

We introduce the set $\mathcal{X}_\lambda$ of *session variables*, used to index replicated data, and are instantiated into *session identifiers*. Names are thus now represented by *name patterns*

$$\overline{n} = n[\mathsf{a}_1, \ldots, \mathsf{a}_k]$$

where each a_i is called an *argument pattern*, that is either a term (representing a prior input), or a session variable or identifier (representing a replication in scope). Public names a are implicitly interpreted as name patterns $a[]$. We define below the grammar of such instrumented processes.

$$
\begin{array}{rll}
P, Q ::= & 0 & \mathsf{in}^{\overline{n}}(N, x); P \\
& P \mid Q & \{\!\{!^{\overline{n}_1}_{\tilde{\mathsf{a}}_1} P_1; \ldots; !^{\overline{n}_k}_{\tilde{\mathsf{a}}_k} P_k\}\!\} \\
& \mathsf{event}(M); P & \mathsf{let}\ x = D\ \mathsf{in}\ P\ \mathsf{else}\ Q \\
& \mathsf{out}(N, M); P &
\end{array}
$$

with $n, n_1, \ldots, n_k$ names, $\tilde{a}_1, \ldots, \tilde{a}_k$ sets of argument patterns. The major addition compared to regular processes is the *session decomposition* $\{\!\{!^{\overline{n}_1}_{\tilde{\mathsf{a}}_1} P_1; \ldots; !^{\overline{n}_k}_{\tilde{\mathsf{a}}_k} P_k\}\!\}$. Intuitively, it gathers processes $P_1, \dots, P_k$ of same structure, whose replicated copies will be indexed by the session variables in $\tilde{\mathsf{a}}_1, \dots, \tilde{\mathsf{a}}_k$, recalling the motivating example. This notation extends to non-replicated processes P_i with $\tilde{\mathsf{a}}_i = \emptyset$.

The task of proving the equivalence of two processes P and Q rephrases, intuitively, to matching the observables (inputs and outputs) of P with those of Q. This is done mostly syntactically in ProVerif while, for session decompositions:

$$P = \{\!\{ !^{\overline{n}_1}_{\tilde{\mathsf{a}}_1} P_1; \dots; !^{\overline{n}_k}_{\tilde{\mathsf{a}}_k} P_k \}\!\} \qquad Q = \{\!\{ !^{\overline{m}_1}_{\tilde{\mathsf{b}}_1} Q_1; \dots; !^{\overline{m}_\ell}_{\tilde{\mathsf{b}}_\ell} Q_\ell \}\!\}$$

our approach may match any observable from a replicated copy of P_i with an equivalent one from a copy of Q_j. Roughly, the current procedure of ProVerif can only handle, in comparison, the case $k = \ell$ and may only match P_i with Q_i. ProVerif also has a requirement that P and Q have the same structure, that we also extend to our context. In the above case, this (recursively) means that all P_i, Q_j have the same structure, and that there are the same number of sessions on each side (non replicated processes, i.e., $\tilde{\mathsf{a}}_i = \emptyset$, counting for 1, and replicated processes, i.e., $\tilde{\mathsf{a}}_i \neq \emptyset$, for $+\infty$). In the following definition, we refer in particular as $\#\tilde{\mathsf{a}}_i \in \{1, +\infty\}$ to the corresponding number of sessions, also using the natural extension of addition to $\mathbb{N} \cup \{+\infty\}$.

Definition 5 (Control-flow equivalence)**.** *Control flow equivalence* $\approx_{cf}$ is the smallest relation on instrumented processes such that:

- $\{\!\{ !^{\overline{n}_1}_{\tilde{\mathsf{a}}_1} P_1, \dots, !^{\overline{n}_k}_{\tilde{\mathsf{a}}_k} P_k \}\!\} \approx_{cf} \{\!\{ !^{\overline{m}_1}_{\tilde{\mathsf{b}}_1} Q_1, \dots, !^{\overline{m}_l}_{\tilde{\mathsf{b}}_l} Q_l \}\!\}$ if for all i, j, $P_i \approx_{cf} Q_j$ and $\sum_{i=1}^{k} \#\tilde{\mathsf{a}}_i = \sum_{j=1}^{l} \#\tilde{\mathsf{b}}_j$;
- let any instructions $\alpha[P_1, \dots, P_n]$ and $\beta[Q_1, \dots, Q_n]$ of the same type (nil, input, output, event, parallel, let; thus $n \in [\![0, 2]\!]$). If for all $i \in [\![1, n]\!]$, $P_i \approx_{cf} Q_i$, then $\alpha \approx_{cf} \beta$.

4.2. From Processes to Instrumented Processes

We now detail how to transform a regular process into an instrumented one. Up to alpha renaming, we assume without loss of generality that processes bind names and variables at most once (by in or new instructions). Most of the instrumentation process intuitively consists of tagging inputs and replications by fresh occurrences to identify them, and to record the tags in scope in the replication's arguments. Such a procedure already exists in ProVerif as detailed in [21], and we only sketch it here, with a particular focus on the new material (session matchings). The full description can be found in Appendix A. The transformation takes the form of a ternary relation between processes $P \Downarrow_{\tilde{\mathsf{a}}} Q$, where P is a process, Q is an instrumented process and $\tilde{\mathsf{a}}$ is a sequence of argument patterns intuitively recording the data dependencies in scope. Typically, replications are instrumented as follows:

$$!\,P \;\Downarrow_{\tilde{\mathsf{a}}}\; \{\!\{ !^{o[\tilde{\mathsf{a}}, i]}_{\{i\}} P' \}\!\}$$

where, for some fresh occurrence o and session variable $i \in \mathcal{X}_\lambda$, $P \Downarrow_{\tilde{\mathsf{a}} \cdot i} P'$ with $\cdot$ being append operation for sequences. This means that i serves as a fresh placeholder for indexing the various copies of P. It is added to the record $\tilde{\mathsf{a}}$ for the next steps of the instrumentation $\Downarrow_{\tilde{\mathsf{a}} \cdot i}$, and to the dependencies of the fresh occurrence label o. The tagging is similar for inputs:

$$\mathsf{in}(N, x); P \;\Downarrow_{\tilde{\mathsf{a}}}\; \mathsf{in}^{o[\tilde{\mathsf{a}}_{|\lambda}]}(N, x); P'$$

with some fresh o and if we have $P \Downarrow_{\tilde{\mathsf{a}} \cdot x} P'$, and $\tilde{\mathsf{a}}_{|\lambda}$ refers to the sequence $\tilde{\mathsf{a}}$ with input terms removed, thus keeping only session variables/identifier. Only inputs and replications (i.e., the sources of unboundedness) need to be tagged with occurrences, meaning that $\Downarrow_{\tilde{\mathsf{a}}}$ is extended to most other process constructions (inputs, outputs, events, let) in the natural way, without modification. The only exception is the parallel operator which, in the same way as replication, has to produce a session matching:

$$P \mid Q \;\Downarrow_{\tilde{\mathsf{a}}}\; \{\!\{ !^{o[\tilde{\mathsf{a}}]}_{\emptyset} P' \}\!\} \mid \{\!\{ !^{o'[\tilde{\mathsf{a}}]}_{\emptyset} Q' \}\!\}$$

with some fresh o, o' and if we have $P \Downarrow_{\tilde{\mathsf{a}}} P'$, $Q \Downarrow_{\tilde{\mathsf{a}}} Q'$. This session matching is "blank" at the moment, that is, it does not record any potential structural symmetries between P and Q. Computing symmetries is managed by the second part of the transformation, taking the form of three *factorisation rules* $\rightsquigarrow$ that normalise the structure of session decompositions. We recall the intuition that $P \mid Q$ models two arbitrary parallel processes P and Q, while $\{\!\{ !^{\overline{n}}_{\tilde{\mathsf{a}}} P, !^{\overline{m}}_{\tilde{\mathsf{b}}} Q \}\!\}$ carries additional information about structural symmetries (i.e., $P \approx_{cf} Q$). These factorisation rules intuitively merge parallel processes into multisets when structural symmetries are identified. All rules are to be understood up to associativity and commutativity of the parallel operator, and are applied to any subprocesses of the instrumented process:

$$\begin{aligned} \{\!\{ !^{\overline{n}}_{\tilde{\mathsf{a}}} (P \mid Q) \}\!\} \cup S &\rightsquigarrow \{\!\{ !^{\overline{n}}_{\tilde{\mathsf{a}}} P \}\!\} \mid \{\!\{ !^{\overline{n}}_{\tilde{\mathsf{a}}} Q \}\!\} \mid S \\ \{\!\{ !^{\overline{n}}_{\tilde{\mathsf{a}}} \{\!\{ !^{\overline{n}_i}_{\tilde{\mathsf{a}}_i} P_i \}\!\}_{i=1}^{n} \}\!\} \cup S &\rightsquigarrow \{\!\{ !^{\overline{n}_i}_{\tilde{\mathsf{a}} \cup \tilde{\mathsf{a}}_i} P_i \}\!\}_{i=1}^{n} \mid S \\ \{\!\{ !^{\overline{n}}_{\tilde{\mathsf{a}}} P \}\!\} \cup S \mid \{\!\{ !^{\overline{m}}_{\tilde{\mathsf{b}}} Q \}\!\} \cup S' &\rightsquigarrow \{\!\{ !^{\overline{n}}_{\tilde{\mathsf{a}}} P, !^{\overline{m}}_{\tilde{\mathsf{b}}} Q \}\!\} \cup S \cup S' \\ &\qquad \text{if } P \approx_{cf} Q \end{aligned}$$

The first two rules present replication and parallel operators under a compact normalised form that facilitates the search for structural symmetries, using only the occurrences with the most dependencies (i.e., the deepest one) in case of nested replications. Note however that these two operations technically break priorly established symmetries, which is why they replace the multiset union ($\cup$) by a regular parallel composition ($|$). On the contrary, the last rule exhibits symmetries by merging session decompositions whose base processes P, Q are control-flow equivalent.

Definition 6 (Instrumentation)**.** Given a process P, we write $[\![P]\!]_i$ to refer to an *instrumentation* of P, i.e., an instrumented process such that $P \Downarrow_{\emptyset} \rightsquigarrow^* [\![P]\!]_i \not\rightsquigarrow$.

Example 1. We already illustrated in the motivation example how our notion of session decomposition helped treating more processes as control-flow equivalent in comparison with the usual approach of ProVerif. One step further, two processes whose syntax are significantly different, e.g., $(!\,P) \mid Q \mid (!\,\mathsf{new}\, n; !\,R)$ and $!\,S$, have instrumentations that may effectively be control-flow equivalent, when those of

P, Q, R and S are. Note that the our results do not require the relation $\rightsquigarrow$ to be confluent.

4.3. Instrumented Semantics

On single processes. We now adapt the operational semantics to instrumented processes. We give an intuition for a sample of important rules here, while the full definition can be found in Appendix A, since most of it is not novel compared to ProVerif's theory. Intuitively, it is a decision-oriented version of the operational semantics: in addition to the explicit replication labels, the attacker's operations are also represented explicitly, facilitating their encoding into Horn clauses for decisional purposes. As such, the semantics operates on *instrumented configurations* $\mathcal{P}, \mathcal{A}, \Lambda$, where $\mathcal{P}$ is a multiset of instrumented processes, $\mathcal{A}$ is a set of terms representing the attacker's knowledge and Λ is a set of name patterns tracking unfolded replications. For example:

$$\mathcal{P}, \mathcal{A}, \Lambda \to_i \mathcal{P}, \mathcal{A} \cup \{M\}, \Lambda \qquad \text{(I-APP)}$$

provided $f(M_1, \ldots, M_n) \Downarrow M$, for some $M_1, \ldots, M_n \in \mathcal{A}$ and $f/n \in \mathcal{F}_c \cup \mathcal{F}_d$. This rule application thus evidences that M is computable by the adversary, and adds it to the knowledge base $\mathcal{A}$. The adversary may also introduce its own (fresh) constants in computations, which take the form of name patterns $b_0[\lambda]$, with b_0 a fixed (but fresh) name and λ a session identifier:

$$\mathcal{P}, \mathcal{A}, \Lambda \to_i \mathcal{P}, \mathcal{A} \cup \{b_0[\lambda]\}, \Lambda \qquad \text{if } b_0[\lambda] \notin \mathcal{A} \quad \text{(I-GEN)}$$

Some terms $M, N \in \mathcal{A}$ may then later be used by, e.g.:

$$\{\!\{\mathsf{out}(N, M); P\}\!\}, \mathcal{A}, \Lambda \to_i \{\!\{P\}\!\}, \mathcal{A} \cup \{M\}, \Lambda \quad \text{(I-OUT)}$$

One other important rule is the one handling both replication and parallel composition, spawning a copy of a process with a fresh occurrence:

$$\begin{aligned}\{\!\{\{\!\{!^{\overline{o}}_{\tilde{\mathsf{a}}}P\}\!\} \cup S\}\!\}, \mathcal{A}, \Lambda &\xrightarrow{repl(\overline{o}\sigma)}_i \\ &\{\!\{P\sigma, \{\!\{!^{\overline{o}}_{\tilde{\mathsf{a}}}P\}\!\} \cup S\}\!\}, \mathcal{A}, \Lambda \cup \{\overline{o}\sigma\}\end{aligned} \qquad \text{(I-REPL)}$$

if $dom(\sigma) = \tilde{\mathsf{a}}$, $img(\sigma) \subseteq \mathbb{N}$, and $\overline{o}\sigma \notin \Lambda$. Intuitively, $P\sigma$ is an instance of one of the instrumented processes contained in the session decomposition $\{\!\{!^{\overline{o}}_{\tilde{\mathsf{a}}}P\}\!\} \cup S$ where σ indicates how the session variables are instantiated. In case of a non-replicated process, i.e., when $\tilde{\mathsf{a}} = \emptyset$, the condition $\overline{o}\sigma \notin \Lambda$ notably prevents it from being replicated more than once (since $\sigma = id$ when $\tilde{\mathsf{a}} = \emptyset$). Finally, we also mention the rule for inputs, that notably triggers a *precise event* $pre(\overline{o}, M)$:

$$\{\!\{\mathsf{in}^{\overline{o}}(N, x); Q\}\!\}, \mathcal{A}, \Lambda \xrightarrow{pre(\overline{o}, M)}_i \{\!\{Q\{^M/_x\}\}\!\}, \mathcal{A}, \Lambda \quad \text{(I-IN)}$$

Some internal *axioms* of ProVerif make references to such precise events to guide, and thus improve the precision of, the decision procedure. They are, in this paper, manually-proved trace properties that are protocol-independent. We use in particular a set of so-called *precise axioms*, described and implemented in [22], [21]. They formalise, intuitively, injectivity properties following from the freshness of occurrences, here $\overline{o}$. For example, one such axiom [22] intuitively states that if a same trace T contains two events $ev = pre(\overline{o}, M)$ and $ev' = pre(\overline{o}, M')$, then $M = M'$.

We then define a notion of instrumented traces. It comes with a weaker variant that executes replicated inputs right after they are unfolded; this intuitively does not induce a loss of generality when constructing equivalence proofs, while giving more information to guide the analysis.

Definition 7 (Instrumented trace)**.** A sequence of transitions $\mathcal{C}_0 \xrightarrow{\ell_1}_i \ldots \xrightarrow{\ell_n}_i \mathcal{C}_n$ is called an *instrumented trace*. We also write $\mathcal{C} \xrightarrow{\ell}_{wi} \mathcal{C}'$ (*weak transition*) when:

- either $\mathcal{C} \xrightarrow{\ell}_i \mathcal{C}'$ using any rule except I-REPL, or I-REPL if the replicated process does not start with an input;
- or $\ell = \ell_1 \cdot \ell_2$ and $\mathcal{C} \xrightarrow{\ell_1}_i \mathcal{C}'' \xrightarrow{\ell_2}_i \mathcal{C}'$ where $\mathcal{C} \xrightarrow{\ell_1}_i \mathcal{C}''$ is derived by rule I-REPL and $\mathcal{C}'' \xrightarrow{\ell_2}_i \mathcal{C}'$ is the application of the rule I-IN on the process replicated in $\mathcal{C} \xrightarrow{\ell_1}_i \mathcal{C}''$.

We write $\mathsf{wtrace}(\mathcal{C})$ the set of $\to_{wi}$-traces of $\mathcal{C}$, and $\mathsf{wrtrace}(\mathcal{C})$ its more permissive variant where the last transition of the trace may be an arbitrary $\to_i$ transition.

Considering a special event $repl_i$ (intuitively modelling a replication followed by an input), we define the satisfaction of events by instrumented traces as follows:

Definition 8 (Event satisfaction)**.** Let $\mathcal{C}_0$ be an instrumented configuration. Let $T : \mathcal{C}_0 \xrightarrow{\ell_1}_i \ldots \xrightarrow{\ell_n}_i \mathcal{C}_n$ be a trace in $\mathsf{wrtrace}(\mathcal{C}_0)$. We define the *satisfaction* of an event ev in T, denoted $T \vdash ev$, to hold when there is j such that $ev = \ell_j$, or $ev = repl_i(\overline{o}, M)$ and $\mathcal{C}_{j-2} \xrightarrow{repl(\overline{o}) \cdot pre(\overline{o}', M)}_{wi} \mathcal{C}_j$. The relation $\vdash$ is naturally extended to conjunction of events.

In particular, $repl(\overline{o})$ and $repl_i(\overline{o}, M)$ are called *replication events*, while the former is more specifically called a *strict replication event*. Given such events ev, we say that $\overline{o}$ is the *replication occurrence* of ev, denoted $\mathsf{orepl}(ev)$.

On biprocesses. In fact, to prove equivalence, ProVerif operates internally on *biprocesses*, that intuitively describe the joint execution of two processes to be proved equivalent. Their semantics is intuitively a straightforward extension of the instrumented semantics ensuring that the two paired processes follow the same execution flow. Formally:

Definition 9 (Biconfiguration)**.** A *biconfiguration* is a tuple $\mathcal{C}^2 = \mathcal{P}^2, \mathcal{A}^2, \Lambda^2$ where $\mathcal{P}^2$ is a multiset of pairs of instrumented processes, $\mathcal{A}^2$ is a set of pairs of terms and Λ^2 is a set of pairs of pattern names. We say that $\mathcal{C}^2$ is *well-formed* when for all $(P, Q) \in \mathcal{P}^2$, we have $P \approx_{cf} Q$, and *initial* when it additionally verifies: $\Lambda^2 = \emptyset$, $\mathcal{A}^2 = \{(a[], a[]) \mid a \in \mathcal{N}_{\mathsf{pub}}\}$ and all occurrence names in $\mathcal{P}^2$ appear at most once.

The main difference with ProVerif's analogue is that our more permissive notion of control-flow equivalence $\approx_{cf}$, through the modelling of session decompositions as unordered multisets, makes the well-formedness condition much less restrictive. We define the projection functions:

$$\mathsf{proj}_i(\mathcal{P}^2) = \{\!\{ P_i \mid (P_0, P_1) \in \mathcal{P}^2 \}\!\}$$

and, analogously, $\mathsf{proj}_i(\mathcal{A}^2)$, $\mathsf{proj}_i(\Lambda^2)$, $\mathsf{proj}_i(\mathcal{C}^2)$. The semantics is then given by a transition relation $\xrightarrow{\ell}_{i^2}$, where ℓ is a pair of (possibly empty) events of the instrumented semantics, resulting in *bitraces*. Typically, similarly to Rule I-APP, adding a new term in the adversary's knowledge base $\mathcal{A}$ is done concurrently in the two projection processes:

$$\mathcal{P}^2, \mathcal{A}^2, \Lambda^2 \rightarrow_{i^2} \mathcal{P}^2, \mathcal{A}^2 \cup \{(M, M')\}, \Lambda^2 \qquad (\text{I}^2\text{-APP})$$

if $(M_1, M_1'), \ldots, (M_n, M_n') \in \mathcal{A}^2$, $f/n \in \mathcal{F}_c \cup \mathcal{F}_d$ and $f(M_1, \ldots, M_n) \Downarrow M$ and $f(M_1', \ldots, M_n') \Downarrow M'$. Note in particular that the pair (M, M') can only be added when the two underlying computations $f(M_1, \ldots, M_n)$ and $f(M_1', \ldots, M_n')$ succeed: the verification procedure, through the notion of *convergence* defined in the next section, will consider this bitrace distinguishable when a such a pair of computations succeeds on one side but not on the other. The other rules are also straightforward to define, e.g., the output rule simply executes the instruction on both sides:

$$\begin{aligned}&\{\!\{(\mathsf{out}(N, M); P, \mathsf{out}(N', M'); P')\}\!\}, \mathcal{A}^2, \Lambda^2 \\ &\quad\rightarrow_{i^2} \{\!\{(P, P')\}\!\}, \mathcal{A}^2 \cup \{(M, M')\}, \Lambda^2\end{aligned} \qquad (\text{I}^2\text{-OUT})$$

if $(N, N') \in \mathcal{A}^2$. The full semantics operates similarly (Appendix A, Figure 4). The definitions of $\mathsf{wtrace}^2(\mathcal{C}^2)$ and $\mathsf{wrtrace}^2(\mathcal{C}^2)$ are analogue to their single-process variant, and so is the notion of event satisfaction $T^2 \vdash^2 (ev, ev')$.

4.4. Convergence Equivalence

Proofs in ProVerif then rely on a syntactic notion of *convergence* of bitraces to establish security properties [23], that we adapt to our setting. As mentioned before, we focus on the criterion for may-testing equivalence, as it illustrates most of the technical developments of our contribution. Analogous criteria for observational equivalence and its preorder are also detailed in [14]. Simply, a biconfiguration $\mathcal{C}$ is convergent if for every (instrumented) transition executable from $\mathsf{proj}_i(\mathcal{C})$, the same transition can be applied at the same position in $\mathsf{proj}_{1-i}(\mathcal{C})$. Formally:

Definition 10 (Convergence)**.** A (well-formed) biconfiguration $\mathcal{C}^2 = \mathcal{P}^2, \mathcal{A}^2, \Lambda^2$ *converges*, denoted $\mathcal{C}^2 {\downarrow\!\uparrow}$, when:

1) *The same communications can be done on the two sides.* If $(P, P') \in \mathcal{P}^2$, both starting with an input or an output on respective channels N and N', then for all $(L, L') \in \mathcal{A}^2$, $N = L$ *iff* $N' = L'$.
2) *The knowledge bases are statically indistinguishable.* If $(M_1, M_1'), \ldots, (M_n, M_n') \in \mathcal{A}^2$, $f/n \in \mathcal{F}_d$ then
$$f(M_1, \ldots, M_n) \Downarrow \mathsf{fail} \text{ iff } f(M_1', \ldots, M_n') \Downarrow \mathsf{fail}\,.$$
3) *The two sides make identical branchings.* If $(P, P') \in \mathcal{P}^2$, both starting with evaluations of respectively D and D' then $D \Downarrow \mathsf{fail}$ iff $D' \Downarrow \mathsf{fail}$.

By extension, we say that a bitrace is convergent if all of its intermediary biconfigurations are convergent.

We then propose a sound condition for may-testing, based on convergent bitraces, proved in [14], while the rest of the paper describes how we verify this condition in practice, using ProVerif's internal machinery among others.

Definition 11. A well-formed biconfiguration $\mathcal{C}^2$ is *may-testing convergent*, denoted $\pi_{\sqsubseteq_m}(\mathcal{C}^2)$, when for all $T \in \mathsf{wtrace}(\mathsf{proj}_0(\mathcal{C}^2))$, there exists a convergent bitrace $T^2 \in \mathsf{wtrace}^2(\mathcal{C}^2)$ such that $\mathsf{proj}_0(T^2) = T$.

Theorem 1 (Equivalence and convergence)**.** Let P_0, P_1 be two processes and $\mathcal{C}^2$ be an initial biconfiguration such that $\mathsf{proj}_j(\mathcal{C}^2) = [\![P_j]\!]_i$, for $j = 0, 1$. If $\pi_{\sqsubseteq_m}(\mathcal{C}^2)$ then $P_0 \sqsubseteq_m P_1$.

5. Clause Generation

We first present the clauses we use to describe instrumented processes, and their properties are stated in the next section. We use the notations for clauses (ev, att...) as in Section 2. We assume a (standard) set of clauses $\mathbb{C}_{\mathcal{A}}(\mathcal{C}^2)$ modelling the attacker's capabilities, as in [18], [21], and formalised in Appendix B. Given an initial biconfiguration $\mathcal{C}^2 = (\{\!\{(P, Q)\}\!\}, \mathcal{A}^2, \emptyset)$, the overall set of clauses characterising $\mathcal{C}^2$ is then defined as:

$$\mathbb{C}_{\mathcal{P}}(\mathcal{C}^2) = [\![|P, Q|]\!]\top\Box \;\cup\; \mathbb{C}_{\mathcal{A}}(\mathcal{C}^2)$$

where the algorithm $[\![|P, Q|]\!]\mathcal{H}r$ is partly detailed below (we give a sample of cases novel to this work, and the full definition is in Appendix B). In this notation, $\mathcal{H}$ is a conjunction of facts and disequalities (with $\top$ being the true value) and r is either a pair of terms or a $\Box$ (indicating an absence of value). Intuitively, $\mathcal{H}$ collects conditions that need to be satisfied for P and Q to be executed, and $r = (o_1, o_2)$ indicates that P and Q have been spawned with occurrence replications o_1 and o_2 respectively. In particular, it should be recorded in $\mathcal{H}$ that o_1 and o_2 should be matched together during an equivalence proof. This is done through the facts $F_{!_i}(r, x, x')$ and $F_!(r)$ whose values are $\top$ when $r = \Box$ and otherwise when $r = (o_1, o_2)$:

$$\begin{gathered} F_{!_i}(r, x, x') = \mathrm{ev}(\mathit{repl}_i(o_1, x), \mathit{repl}_i(o_2, x')) \\ F_!(r) = \mathrm{ev}(\mathit{repl}(o_1), \mathit{repl}(o_2)) \end{gathered}$$

$$\begin{aligned}
&[\![|S, S'|]\!]\mathcal{H}r = \textstyle\bigcup_{!^{\overline{o_1}}_{\tilde{\mathsf{a}}_1} P_1 \in S} \; \bigcup_{!^{\overline{o_2}}_{\tilde{\mathsf{a}}_2} P_2 \in S'} [\![|P_1, P_2|]\!]\mathcal{H}(\overline{o_1}, \overline{o_2}) \\
&[\![|\mathsf{in}^{\overline{o}}(N, x); P, \mathsf{in}^{\overline{o'}}(N', x'); P'|]\!]\mathcal{H}r = \\
&\quad [\![|P, P'|]\!]\mathcal{H}_1\Box \cup \{\mathcal{H} \wedge F_!(r) \rightarrow \mathrm{input}(N, N')\} \\
&\quad \text{with } \mathcal{H}_1 = \mathcal{H} \wedge \mathrm{msg}(N, x, N', x') \wedge F_{!_i}(r, x, x') \wedge \\
&\qquad\qquad \mathrm{ev}(\mathit{pre}(\overline{o}, x), \mathit{pre}(\overline{o'}, x')) \\
&[\![|\mathsf{out}(N, M); P, \mathsf{out}(N', M'); P'|]\!]\mathcal{H}r = \\
&\quad [\![|P, P'|]\!](\mathcal{H} \wedge F_!(r))\Box \\
&\quad \cup \{\mathcal{H} \wedge F_!(r) \rightarrow \mathrm{msg}(N, M, N', M')\}
\end{aligned}$$

Furthermore, given a bitrace T, we define the set of Horn clauses $\mathbb{C}_e(T) = \{\rightarrow \mathrm{ev}(ev, ev') \mid T \vdash^2 (ev, ev')\}$ representing all the events emitted in T.

Example 2. For our running example, we translate the two instrumented processes $\{\!\!\{!^{o_3[i,j]}_{i,j} P_{left}\}\!\!\} \mid \{\!\!\{!^{o_4[i,j]}_{i,j} R_{left}\}\!\!\}$ and $\{\!\!\{!^{o_1[i]}_{i} P_{right}\}\!\!\} \mid \{\!\!\{!^{o_2[i]}_{i} R_{right}\}\!\!\}$. Let us look more closely at the passport components, i.e. the translation $[\![\{\!\!\{!^{o_3[i,j]}_{i,j} P_{left}\}\!\!\}, \{\!\!\{!^{o_1[i]}_{i} P_{right}\}\!\!\}]\!]\top\Box$.

Recall that by instrumentation, P_{left} and P_{right} are the processes $P(k_\ell[i])$ and $P(k_r[i'])$ where the inputs have been associated with the occurrences $o'_\ell[i,j]$ and $o'_r[i']$, and the name n has been replaced by $n_\ell[i,j]$ and $n_r[i']$ respectively. The translation $[\![\{\!\!\{!^{o_3[i,j]}_{i,j} P_{left}\}\!\!\}, \{\!\!\{!^{o_1[i']}_{i'} P_{right}\}\!\!\}]\!]\top\Box$ will then record the replication occurrences $o_3[i,j]$ and $o_1[i']$. As the first actions in the processes P_{left} and P_{right} are outputs, it results:

$$F_! \to \mathrm{msg}(c, n_\ell[i,j], c, n_r[i'])$$

with $F_! = \mathrm{ev}(\mathit{repl}(o_3[i,j]), \mathit{repl}(o_1[i']))$ being propagated in the hypotheses of the remaining clauses. When translating the second actions, i.e., the inputs $\mathsf{in}^{o'_\ell[i,j]}(c,x)$ and $\mathsf{in}^{o'_r[i']}(c,x')$, only the precise event is added in the hypothesis and not the occurrence fact $F_{!_i}$ as such fact is only added when translating the first actions after a session matching. This yields the following hypothesis H to be propagated:

$$F_! \wedge \mathrm{msg}(c,x,c,x') \wedge \mathrm{ev}(\mathit{pre}(o'_\ell[i,j],x), \mathit{pre}(o'_r[i'],x'))$$

Finally, as in ProVerif, going through the conditional branch, we compute the success and failure conditions before translating the final output actions, yielding the following clauses:

$$\begin{array}{l}
H\sigma_1\sigma_2 \to \mathrm{msg}(c,\mathsf{ok},c,\mathsf{ok})\\
H\sigma_1 \wedge \forall z'.x' \neq \mathsf{senc}(z',k_r[i']) \to \mathrm{msg}(c,\mathsf{ok},c,\mathsf{error})\\
H\sigma_2 \wedge \forall z.x \neq \mathsf{senc}(z,k_\ell[i]) \to \mathrm{msg}(c,\mathsf{error},c,\mathsf{ok})\\
H \wedge \forall z'.x' \neq \mathsf{senc}(z',k_r[i']) \wedge \forall z.x \neq \mathsf{senc}(z,k_\ell[i,j])\\
\qquad \to \mathrm{msg}(c,\mathsf{error},c,\mathsf{error})
\end{array}$$

with $\sigma_1 = \{^{\mathsf{senc}(z,k_\ell[i])}/_x\}$ and $\sigma_2 = \{^{\mathsf{senc}(z',k_r[i'])}/_{x'}\}$.

As mentioned, we also consider the clauses $\mathbb{C}_\mathcal{A}(\mathcal{C}^2)$ describing the attacker capabilities. They include in particular:

$$\begin{array}{c}
\to \mathrm{att}(c,c) \qquad \to \mathrm{att}(\mathsf{ok},\mathsf{ok}) \qquad \to \mathrm{att}(\mathsf{error},\mathsf{error})\\
\mathrm{msg}(x,y,x',y') \wedge \mathrm{att}(x,x') \to \mathrm{att}(y,y')\\
\mathrm{att}(x,y) \wedge \mathrm{att}(x,y') \wedge y \neq y' \to \mathrm{bad}
\end{array}$$

The first three clauses represents the fact that the attacker knows the initial constants. The fourth clause indicates that if the attacker knows a channel, it can learn the messages sent over it. Finally, the fifth clause models the fact that by deducing twice the same messages on the left side but different messages on the right side, the attacker can distinguish the two traces, i.e. the bitrace is not convergent. In our example, it translates the fact that a bitrace satisfying the conditional in the passport on one side side but failing it on the other side will lead to a non-convergent bitrace.

Soundness. We aim to use the saturation procedure of ProVerif [21] that applies axioms when saturating the set of Horn clauses. We reuse in particular their notions of derivation of a fact from a set of clauses $\mathbb{C}$, which is intuitively a tree whose nodes are labelled by Horn clauses from $\mathbb{C}$ and their edges are labelled by ground facts that are instances of the Horn clauses labeling the nodes. As we consider a more general semantics and a larger set of clauses than in [21], we need show that non-convergent bitraces lead to bad being derivable in the initial set of Horn clauses.

Theorem 2. Let $\mathcal{C}^2$ be an initial biconfiguration. For all $T^2 \in \mathsf{wrtrace}^2(\mathcal{C}^2)$, if T^2 does not converge then there exists a derivation of bad from $\mathbb{C}_\mathcal{P}(\mathcal{C}^2) \cup \mathbb{C}_e(T^2)$.

As previously mentioned, we reused the saturation procedure of ProVerif [21], and can rely on its correctness: the derivability of bad is preserved through saturation.

Proposition 1 ([21, Theorem 4])**.** Let $\mathcal{C}^2$ be an initial biconfiguration. For all $T^2 \in \mathsf{wrtrace}^2(\mathcal{C}^2)$, if there exists a derivation of bad from $\mathbb{C}_\mathcal{P}(\mathcal{C}^2) \cup \mathbb{C}_e(T^2)$ then there exists a derivation of bad from $\mathsf{saturate}(\mathbb{C}_\mathcal{P}(\mathcal{C}^2)) \cup \mathbb{C}_e(T^2)$.

Though we usually denote the hypotheses of a clause as H, when denoting them as $H \wedge \phi$, we implicitly consider the hypotheses to be split into a conjunction H of facts and a conjunction ϕ of term disequalities. In that case, given a substitution σ, we denote by $\sigma \models \phi$ the traditional first order logic satisfaction relation on term disequalities. By combining Theorem 2 and Proposition 1, we thus obtain the cornerstone property of our procedure.

Corollary 1. Let $\mathcal{C}^2$ be an initial biconfiguration. If $T^2 \in \mathsf{wrtrace}^2(\mathcal{C}^2)$ does not converge then there exists $H \wedge \phi \to \mathrm{bad}$ in $\mathsf{saturate}(\mathbb{C}_\mathcal{P}(\mathcal{C}^2))$ and a substitution σ such that $\sigma \models \phi$ and for all $\mathrm{ev}(ev, ev') \in H$, $T^2 \vdash^2 (ev, ev')\sigma$.

Example 3. If $\mathcal{C}^2_{\mathsf{BAC}}$ is an initial biconfiguration corresponding to the processes of the running example, the set $\mathsf{saturate}(\mathbb{C}_\mathcal{P}(\mathcal{C}^2_{\mathsf{BAC}}))$ contains two clauses deriving bad:

$$\begin{array}{l}
C_1 = \mathrm{ev}(\mathit{repl}(o_3[i_1,j_1]), \mathit{repl}(o_1[i])) \wedge\\
\qquad \mathrm{ev}(\mathit{repl}_i(o_4[i_2,j_2], n_l[i_1,j_1]), \mathit{repl}_i(o_2[i], n_r[i])) \wedge\\
\qquad H^1_{pre} \wedge i_1 \neq i_2 \to \mathrm{bad}\\
C_2 = \mathrm{ev}(\mathit{repl}(o_3[i,j]), \mathit{repl}(o_1[i_1])) \wedge\\
\qquad \mathrm{ev}(\mathit{repl}_i(o_4[i,j'], n_l[i,j]), \mathit{repl}_i(o_2[i_2], n_r[i_1])) \wedge\\
\qquad H^2_{pre} \wedge i_1 \neq i_2 \to \mathrm{bad}
\end{array}$$

The first clause corresponds to the case where on S_{right}, the nonce $n_r[i]$ of the reader was sent to the correct passport, as indicated by $\mathit{repl}_i(o_2[i], n_r[i])$, thus ok will be output. On the left side however, $\mathit{repl}_i(o_4[i_2,j_2], n_l[i_1,j_1])$ and $i_1 \neq i_2$ indicate that the nonce $n_l[i_1,j_1]$ generated by a reader having the key $k_l[i_1]$ was sent to a passport with the key $k_l[i_2]$. Since $i_1 \neq i_2$, the keys are different hence error will be output. The second clause is the dual of the first one, i.e., the test will succeed on S_{left} but fail on S_{right}. Here, H^1_{pre} and H^2_{pre} contains the precise events (omitted). The sets of initial and saturated clauses can be displayed by running our implementation [14] on the file `running_example.pv`

6. Semantics Conditions for Equivalence

Relying on the characterisation of Corollary 1, we now state the conditions implying may-testing. Recall that in [14], we present extended conditions which additionally allow to imply observational equivalence and its preorder.

6.1. Session Matching

In this section, we introduce notions that allow us to reason over how sessions can be matched. For example, when a biconfiguration reaches a pair of session decompositions (S, S'), any process $!^{o[\tilde{\mathsf{a}}]}_{\tilde{\mathsf{b}}} P$ from S can potentially be matched with a process $!^{o'[\tilde{\mathsf{a}}']}_{\tilde{\mathsf{b}}'} P'$ from S'. All these potential matchings can be pre-computed statically as follows:

Definition 12 (Potential matching)**.** Let P, Q two processes such that $P \approx_{cf} Q$. We define the set $\mathsf{pm}(P, Q)$, called the *set of potential matching* of P and Q:

- $\mathsf{pm}(S, S') = \bigcup_{!^{\overline{o}}_{\tilde{\mathsf{a}}} P \in S, !^{\overline{o}'}_{\tilde{\mathsf{a}}'} P' \in S'} \{(o, o')\} \cup \mathsf{pm}(P, P')$
- $\mathsf{pm}(\alpha[P_1, \dots, P_n], \beta[Q_1, \dots, Q_n]) = \bigcup_{i=1}^{n} \mathsf{pm}(P_i, Q_i)$ with α, β two instructions of same nature (nil, input, output, event, let, parallel). Note that, thus, $0 \leq n \leq 2$.

Given a biconfiguration $\mathcal{C}^2 = (\{\!\{(P, Q)\}\!\}, \mathcal{A}^2, \emptyset)$, we denote by $\mathsf{pm}(\mathcal{C}^2)$ the set $\mathsf{pm}(P, Q)$.

We can also syntactically determine on the initial processes how many session identifiers an occurrence o must take as arguments, denoted $\mathsf{ar}^{\lambda}_{\mathcal{C}}(o)$, that is when $!^{o[\tilde{\mathsf{a}}]}_{\tilde{\mathsf{b}}} P$ in $\mathcal{C}^2$, $\mathsf{ar}^{\lambda}_{\mathcal{C}}(o)$ is the number of session identifiers in $\tilde{\mathsf{a}}$.

Example 4. Coming back to our running example, we have $\mathsf{pm}(\mathcal{C}^2_{\mathsf{BAC}}) = \{(o_3, o_1); (o_4, o_2)\}$, $\mathsf{ar}^{\lambda}_{\mathcal{C}}(o_1) = \mathsf{ar}^{\lambda}_{\mathcal{C}}(o_2) = 1$ and $\mathsf{ar}^{\lambda}_{\mathcal{C}}(o_3) = \mathsf{ar}^{\lambda}_{\mathcal{C}}(o_4) = 2$.

To further reason on the replication occurrences in the initial biconfiguration $\mathcal{C}^2$, we define the partial order relation $o \prec_{\mathcal{C}^2} o'$ to hold when o' is in the scope of o in the processes of $\mathcal{C}^2$, that is when $!^{\overline{o}}_{\tilde{\mathsf{a}}} P$ occurs in $\mathcal{C}^2$ and $!^{\overline{o}'}_{\tilde{\mathsf{a}}'} Q$ occurs in P. We denote $\preceq_{\mathcal{C}^2}$ the reflexive closure of $\prec_{\mathcal{C}^2}$. We naturally extend these notions to replication patterns, i.e. $o[\tilde{\mathsf{a}}] \prec_{\mathcal{C}^2} o'[\tilde{\mathsf{a}}']$ when $o \prec_{\mathcal{C}^2} o'$ and $\tilde{\mathsf{a}}$ is a prefix of $\tilde{\mathsf{a}}'$.

We show the relation between the set $\mathsf{pm}(\mathcal{C}^2)$ and the order $\prec_{\mathcal{C}^2}$ on bitraces in the following lemma. It typically shows that all replication bievents satisfied by a bitrace have potentially matched occurrences.

Lemma 1 (Consistency)**.** Let $\mathcal{C}^2$ be an initial biconfiguration. For all $T^2 \in \mathsf{wrtrace}^2(\mathcal{C}^2)$, if $T^2 \vdash^2 (ev_0, ev_1)$ and $T^2 \vdash^2 (ev_2, ev_3)$ with $\mathsf{orepl}(ev_i) = o_i[\tilde{\mathsf{a}}_i]$ for $i = 0 \dots 3$:

- $(o_0, o_1), (o_2, o_3) \in \mathsf{pm}(\mathcal{C}^2)$
- $o_0[\tilde{\mathsf{a}}_0] \prec_{\mathcal{C}^2} o_2[\tilde{\mathsf{a}}_2]$ if and only if $o_1[\tilde{\mathsf{a}}_1] \prec_{\mathcal{C}^2} o_3[\tilde{\mathsf{a}}_3]$

We now arrive to the main tool for matching sessions. Recall that to prove may-testing, we need to show that for all traces T in $\mathsf{proj}_0(\mathcal{C}^2)$, we can find a corresponding convergent bitrace T^2 of C^2 with $\mathsf{proj}_0(T^2) = T$. In particular, for every replication event satisfied by T, e.g. $T \vdash ev = \mathit{repl}(o[\tilde{\mathsf{a}}])$, there must be a corresponding $ev' = \mathit{repl}(o'[\tilde{\mathsf{a}}'])$ such that $T^2 \vdash^2 (ev, ev')$. Intuitively, a session matching is a function that associates each such event ev with a corresponding occurrence $o'[\tilde{\mathsf{a}}']$. However, $\tilde{\mathsf{a}}'$ may not only contain session identifiers but also terms corresponding to previous inputs. To avoid reasoning on the messages input in $\mathsf{proj}_1(T^2)$, we strip $\tilde{\mathsf{a}}'$ from these input terms, i.e. $\tilde{\mathsf{a}}'_{|\lambda}$, to only preserve the session identifiers. In such a case, we say that $o'[\tilde{\mathsf{a}}'_{|\lambda}]$ is a *pure replication pattern*.

Definition 13 (Session matching)**.** We consider an initial biconfiguration $\mathcal{C}^2 = (\{\!\{(P, Q)\}\!\}, \mathcal{A}^2, \emptyset)$. A *session matching* is a partial function ρ from ground replication events, to pure replication patterns such that

- $\rho(ev) = o'[\tilde{\mathsf{a}}']$ with $\mathsf{orepl}(ev) = o[\tilde{\mathsf{a}}]$ implies $(o, o') \in \mathsf{pm}(\mathcal{C}^2)$, $\mathsf{ar}^{\lambda}_{\mathcal{C}^2}(o') = |\tilde{\mathsf{a}}'|$;
- $\mathsf{orepl}(ev) \preceq_{\mathcal{C}^2} \mathsf{orepl}(ev')$ if and only if $\rho(ev) \preceq_{\mathcal{C}^2} \rho(ev')$

Example 5. Coming back to our running example, consider a trace $T \in \mathsf{proj}_0(\mathcal{C}^2_{\mathsf{BAC}})$ that executes a single passive session between the passport and the reader in S_{left}, the trace T would satisfy three events: $ev_1 = \mathit{repl}(o_3[1, 1])$ (unfolding of P_{left}), $ev_2 = \mathit{repl}(o_4[1, 1])$ (unfolding of R_{left}), and $ev_3 = \mathit{repl}_i(o_4[1, 1], n_l[1, 1])$ (unfolding of R_{left} with $n_l[1, 1]$ the message input by R_{left} and sent by P_{left}).

A session matching from P_{left} to P_{right} could be the function ρ such that $\rho(ev_1) = o_1[1]$ and $\rho(ev_2) = \rho(ev_3) = o_2[1]$ which would also correspond to a single session between the passport and the reader in S_{right}.

6.2. Falsifying conditions

To satisfy our may-testing predicate (Definition 11), we need to find for all traces T a convergent bitrace T^2 such that $\mathsf{proj}_0(T^2) = T$. Due to Corollary 1, we know that a non-convergent bitrace T^2 yields a clause $H \to \mathrm{bad}$ in $\mathsf{saturate}(\mathbb{C}_{\mathcal{P}}(\mathcal{C}^2))$ such that all events of H are satisfied by T^2. Therefore, by showing that we can build a bitrace T^2 that *falsifies* all hypotheses of the clauses of $\mathsf{saturate}(\mathbb{C}_{\mathcal{P}}(\mathcal{C}^2))$ deriving bad, we will ensure that T^2 converges. We thus define such sufficient *falsifying conditions*.

Given a formula ϕ and a set of variables $\tilde{x}$, we denote by $\phi_{|\tilde{x}}$ the formula where all variables of ϕ not in $\tilde{x}$ are replaced by fresh names; thus only keeping variables from $\tilde{x}$. Since formulas in clauses are only composed of disequalities, we have that for all substitutions σ, $\sigma \models \phi$ implies $\sigma \models \phi_{|\tilde{x}}$.

Definition 14 (Falsifying condition)**.** Let $C = (\phi \wedge H \wedge \bigwedge_{i=1}^{n} F_i \to \mathrm{bad})$ be a clause such that the F_is are replication events and H contains any other facts. The *falsifying condition* of C, denoted $\mathsf{falsify}(C)$, is (Ω, ϕ') where:

- Ω is the function $[\mathsf{proj}_0(F_i) \mapsto y_i]_{i=1}^{n}$ where the y_is are fresh distinct variables
- $\phi' = \forall \tilde{x}_1 \setminus \tilde{x}_0 . (\neg \phi_{|\tilde{x}_{1|\lambda}} \vee \bigvee_{i=1}^{n} y_i \neq o_i[\tilde{\mathsf{a}}_{i|\lambda}])$

where $\mathsf{orepl}(\mathsf{proj}_1(F_i)) = o_i[\tilde{\mathsf{a}}_i]$ for $i = 1 \dots n$ and $\tilde{x}_j = \mathit{vars}(\mathsf{proj}_j(H)) \cup \bigcup_{i=1}^{n} \mathit{vars}(\mathsf{proj}_j(F_i))$ for $j = 0, 1$

Intuitively, when $\mathsf{falsify}(C) = (\Omega, \phi)$, the function Ω can be seen as a session matching on *open* terms, i.e. that contain variables. The formula ϕ represents sufficient conditions for the hypotheses of the biclause C to be falsified. Therefore, any session matching that is an instantiation of Ω and that verifies ϕ will falsify the hypotheses of C.

Example 6. Consider the clauses C_1 and C_2 from Example 3. We have $\mathsf{falsify}(C_1) = (\Omega_1, \phi_1)$ where:

- $\Omega_1(\mathit{repl}(o_3[i_1, j_1])) = y_1$
- $\Omega_1(\mathit{repl}_i(o_4[i_2, j_2], n'[i_1, j_1])) = y_2$
- $\phi_1 = \forall i . (y_1 \neq o_1[i] \vee y_2 \neq o_2[i])$

and $\mathsf{falsify}(C_2) = (\Omega_2, \phi_2)$ where

- $\Omega_2(\mathit{repl}(o_3[i,j])) = y_1$
- $\Omega_2(\mathit{repl}_i(o_4[i,j'], n'[i,j])) = y_2$
- $\phi_2 = \forall i_1, i_2.(y_1 \neq o_1[i_1] \vee y_2 \neq o_2[i_2] \vee i_1 = i_2)$

Using Corollary 1, we show that a bitrace with a corresponding session matching that falsifies the hypotheses of all clauses deriving bad is necessary convergent.

Definition 15 (Falsification)**.** Let $\mathcal{C}^2$ be an initial biconfiguration. Let $T \in \mathsf{wrtrace}(\mathsf{proj}_0(\mathcal{C}^2))$. Let $C = H \to \mathrm{bad}$ be a clause with $\mathsf{falsify}(C) = (\Omega, \phi)$. Let ρ be a session matching of $\mathcal{C}^2$.

We define $T, \rho \vdash_s \mathsf{falsify}(C)$ to hold when for all substitutions σ, if $T \vdash \mathsf{proj}_0(H\sigma)$ and for all $ev \in dom(\Omega)$, $ev\sigma \in dom(\rho)$ and $\Omega(ev)\sigma = \rho(ev\sigma)$ then $\sigma \models \phi$.

Lemma 2 (Convergence criterion)**.** Let $\mathcal{C}^2$ be an initial biconfiguration and $\mathbb{C}_{sat}^{\mathrm{bad}}$ the clauses of $\mathsf{saturate}(\mathbb{C}_{\mathcal{P}}(\mathcal{C}^2))$ deriving bad. Let ρ be a session matching of $\mathcal{C}^2$, and $T^2 \in \mathsf{wrtrace}^2(\mathcal{C}^2)$. If

- $T^2 \vdash^2 (ev, ev')$ and $\mathsf{orepl}(ev') = o[\tilde{\mathsf{a}}]$ implies $ev \in dom(\rho)$ and $\rho(ev) = o[\tilde{\mathsf{a}}_{|\lambda}]$
- for all $C \in \mathbb{C}_{sat}^{\mathrm{bad}}$, $\mathsf{proj}_0(T^2), \rho \vdash_s \mathsf{falsify}(C)$

then T^2 converges.

Lemma 2 is the core lemma that allows us to prove may-testing equivalence (as well as the other equivalences).

Theorem 3 (May-testing)**.** Let $\mathcal{C}^2$ be an initial biconfiguration and $\mathbb{C}_{sat}^{\mathrm{bad}}$ the clauses of $\mathsf{saturate}(\mathbb{C}_{\mathcal{P}}(\mathcal{C}^2))$ deriving bad. If for all $T \in \mathsf{wtrace}(\mathsf{proj}_0(\mathcal{C}^2))$, there exists a session matching of $\mathcal{C}$ such that:

- $\{ev \text{ replication event} \mid T \vdash ev\} \subseteq dom(\rho)$
- for all $C \in \mathbb{C}_{sat}^{\mathrm{bad}}$, $T, \rho \vdash_s \mathsf{falsify}(C)$

then $\pi_{\sqsubseteq_m}(\mathcal{C})$ holds.

7. Practical Verification of Equivalences

7.1. Skolemisation

To prove may-testing by relying on Theorem 3, we need to generate session matchings falsifying the hypotheses of all biclauses deriving bad. Although these session matchings may concretely depend on the considered traces, such dependencies are tedious to capture in practice. We therefore implement a stronger but simpler condition where, given a biclause C and its falsifying condition $\mathsf{falsify}(C) = (\Omega, \phi)$, we build *one* session matching satisfying $\mathsf{falsify}(C)$ for *any* instantiation of $dom(\Omega)$. In fact, if we denote $dom(\Omega) = \{ev_i\}_{i=1}^n$ with $y_i = \Omega(ev_i)$, the satisfiability of $\mathsf{falsify}(C)$ can intuitively be interpreted as a first-order formula:

$$\forall ev_1.\dots,\forall ev_n.\exists y_1.\dots,\exists y_n.\phi$$

Building on this intuition, we will get rid of existential quantifiers via a *Skolemisation* process formalised below. To that end, we consider an additional set of name $\mathcal{N}_s$ that will be used as Skolem functions.

Definition 16 (Skolemisation)**.** Let $\mathcal{C}^2$ be an initial biconfiguration. Let $C = (H \wedge \phi \to \mathrm{bad})$ be a biclause. Assume that $\mathsf{falsify}(C) = (\Omega, \phi')$. We say that a substitution σ is a *Skolemisation* of $\mathsf{falsify}(C)$ when $dom(\sigma) = img(\Omega)$, $\phi_{|vars_0(C)} \models \phi'\sigma$ and for all $ev, ev' \in dom(\Omega)$,

- if $\mathsf{orepl}(ev) = o[\tilde{\mathsf{a}}]$ then
$$\Omega(ev)\sigma = o'[s_1[\tilde{\mathsf{a}}_1], \dots, s_k[\tilde{\mathsf{a}}_k]]$$
with $(o, o') \in \mathsf{pm}(\mathcal{C}^2)$, $k = \mathsf{ar}_{\mathcal{C}^2}^{\lambda}(o')$, $s_1, \dots, s_k \in \mathcal{N}_s$ and $\tilde{\mathsf{a}}_1, \dots, \tilde{\mathsf{a}}_k$ only contain subterms of ev
- $\mathsf{orepl}(ev) \preceq_{\mathcal{C}^2} \mathsf{orepl}(ev')$ iff $\Omega(ev)\sigma \preceq_{\mathcal{C}^2} \Omega(ev')\sigma$

Note that given a biclause deriving bad, there are finitely many potential Skolemisations of $\mathsf{falsify}(C)$ (modulo renaming of the names from $\mathcal{N}_s$). Our implementation typically consists of finding Skolemisation substitutions for all clauses in $\mathbb{C}_{sat}^{\mathrm{bad}}$ that satisfy the conditions of Theorem 4.

Example 7. Coming back to Example 6, we can consider the following Skolemisation substitutions σ_1 and σ_2 of $\mathsf{falsify}(C_1)$ and $\mathsf{falsify}(C_2)$ respectively:

- $\sigma_1 = \{y_1 \mapsto o_1[s_1[i_1, j_1]]; y_2 \mapsto o_2[s_2[i_2, j_2, i_1, j_1]]\}$
- $\sigma_2 = \{y_1 \mapsto o_1[s_3[i,j]]; y_2 \mapsto o_2[s_3[i,j]]\}$

where $s_1, s_2, s_3, s_4 \in \mathcal{N}_s$. Notice that $\models \phi_1\sigma_1$ and $\models \phi_2\sigma_2$.

Our theorems also relies on a simplification function of Horn clauses used in the saturation procedure of ProVerif (see [21, Section 3.2.5]) that, given a clause C, either $C\!\downarrow$ returns a set of simpler clauses or $\bot$. The details of this function are out of the scope of this paper and we only use its following property: for all clauses $C = (H \to \mathrm{bad})$, if $C\!\downarrow = \bot$ then for all traces T of the initial configuration, for all substitutions σ, $T \not\vdash H\sigma$. That is, no trace can satisfy the hypotheses of the clause.

Theorem 4 (May-testing)**.** Let $\mathcal{C}^2$ be an initial biconfiguration and $\mathbb{C}_{sat}^{\mathrm{bad}}$ the clauses of $\mathsf{saturate}(\mathbb{C}_{\mathcal{P}}(\mathcal{C}^2))$ deriving bad. For all clauses $C \in \mathbb{C}_{sat}^{\mathrm{bad}}$, we assume a Skolemisation substitution σ_C of $\mathsf{falsify}(C)$. We also assume that for all $C = (H \to \mathrm{bad}), C' = (H' \to \mathrm{bad}) \in \mathbb{C}_{sat}^{\mathrm{bad}}$ with $\mathsf{falsify}(C) = (\Omega, \phi)$, $\mathsf{falsify}(C') = (\Omega', \phi')$, and $H_{all} = \mathsf{proj}_0(H) \wedge \mathsf{proj}_0(H')$, the following conditions hold for all $ev \in dom(\Omega)$, for all $ev' \in dom(\Omega')$:

1) Let us write $(t, t') = (\mathsf{orepl}(ev), \mathsf{orepl}(ev'))$ when ev or ev' is a strict replication event, and $(t, t') = (ev, ev')$ otherwise. If $\alpha = mgu(t, t')$ then
$$(H_{all} \wedge \Omega(ev)\sigma_C \neq \Omega(ev)\sigma_{C'} \to \mathrm{bad})\alpha\!\downarrow = \bot$$
2) If $\alpha = mgu(\Omega(ev)\sigma_C, \Omega(ev')\sigma_{C'})$ then
$$(H_{all} \wedge \mathsf{orepl}(ev)_{|\lambda} \neq \mathsf{orepl}(ev')_{|\lambda} \to \mathrm{bad})\alpha\!\downarrow = \bot$$

Then $\pi_{\sqsubseteq_m}(\mathcal{C}^2)$ holds.

The first condition of the theorem intuitively verifies that a concrete event cannot be associated to two different replication patterns (otherwise the session matching we build would not be a function). The second condition verifies that the session matching we build is injective on the replication occurrences (as required by the second bullet point in Definition 13).

7.2. Experiments

Building on Theorem 4, we implemented a prototype for verifying may-testing. The source code is available in [14]. Note that our prototype also implements the extended procedure that can additionally prove observational preorder and observational equivalence. Table 1 displays the benchmarks we performed on several protocols from the literature.

First, we included a couple of toy examples showcasing the may-testing, observational preorder and observational equivalence, as well as the lighter structural restrictions required by our approach compared to the baseline approach of ProVerif. We also verified the protocols included in ProVerif's distribution as sanity checks to ensure that our new algorithm does not induce a drop in expressivity and performance for previously provable protocols.

Second, our experiments include protocols such as BAC [20] (various properties not limited to the minimal model of our motivation example), Hash-Lock [24], LAK [25], PACE [26], [27], Helios [28], Feldhofer [29]. Most of these models were taken from the Ukano tool that we compare against [12].

Finally, we also show that our approach surpasses Ukano on, e.g., unlinkability proofs in a simplified standard TLS handshake. Indeed, Ukano requires that:

1) the protocol must be a two-party protocol between an initiator I and a responder R;
2) the processes of the initiator and responder cannot use the full range of ProVerif syntax, e.g., else-branches can only be null or restricted outputs of constants;
3) equivalences are syntactically restricted to be of the form:

$$!\mathsf{new}\, k; (!P_i \mid !P_R) \approx !\mathsf{new}\, k; (P_i \mid P_R)$$

4) the protocol must admit an appropriate *idealisation function* and must satisfy two sufficient conditions called *well-authentication* and *frame opacity*.

These conditions prevent Ukano to handle TLS. Indeed, a TLS handshake starts with a negotiation phase where the server and the client must agree on the TLS protocol version, the Diffie-Hellman (DH) groups and other cryptographic algorithms. During this phase, the server may send an HelloRetryRequest depending on the DH groups and key shares the client sent. Such requests subsequently affect the control flow of the protocol, which cannot be modelled with the restriction on else-branches discussed in Item 2. Besides, even if one considered a simpler version of TLS where Server and Client would have already agreed on the handshake parameters, the conditions 1 and 3 combined limit the security guarantees. In TLS, the identity of the client is typically its long term public key. In the above equivalence statements, k would thus play the role of the private key associated to the public key of the client. Thus, Ukano could only prove unlinkability of TLS *without* revealing the public keys of the clients (as outputting them in P_I or P_R would trivially break unlinkability). Our approach can typically handle such models using a third outputting process in parallel. In fact, our experiment showed that

Protocol	Properties	Our Proofs	Time	U	PV
ProVerif distribution					
EKE	WeakSec.	$✓\approx_o$	2s	-	$✓\approx_o$
BAC (prv. ch)	Unlink.	✗		✗	✗
NSPK	StrongSec.	$✓\approx_o$	1s	-	$✓\approx_o$
Prv. Auth.	Anon.	$✓\approx_o$	1s	✗	$✓\approx_o$
WMF	StrongSec.	$✓\approx_o$	1s	-	$✓\approx_o$
Ukano distribution					
BAC+AA+PA	Anon.	$✓\sqsubseteq_m \cap \sqsupseteq_o$	1s	$✓\approx_m$	✗
BAC+AA+PA	Unlink.	$✓\sqsubseteq_m \cap \sqsupseteq_o$	1s	$✓\approx_m$	✗
Feldhofer	Unlink.	$✓\sqsubseteq_m \cap \sqsupseteq_o$	1s	$✓\approx_m$	✗
Hash Lock	Unlink.	$✓\sqsubseteq_m \cap \sqsupseteq_o$	1s	$✓\approx_m$	✗
LAK	Unlink.	$✓\sqsubseteq_m \cap \sqsupseteq_o$	1s	$✓\approx_m$	✗
PACE	Unlink.	$✓\sqsubseteq_m \cap \sqsupseteq_o$	3m20s	$✓\approx_m$	✗
Simplified TLS					
Basic	Unlink.	$✓\sqsubseteq_m \cap \sqsupseteq_o$	2s	✗	✗
With HRR	Unlink.	$✓\sqsubseteq_m \cap \sqsupseteq_o$	1m48s	-	✗
With HRR, PSK	Unlink.	✗		-	✗
Other models					
Running example	Unlink.	$✓\sqsubseteq_m \cap \sqsupseteq_o$	1s	$✓\approx_m$	✗
Helios	Vote Prv.	$✓\approx_o$	1s	-	$✓\approx_o$
Toy Simu 1,2		$✓\sqsubseteq_o \cap \sqsupseteq_o$	1s	-	✗
Toy Flow 1		$✓\sqsubseteq_o \cap \sqsupseteq_o$	1s	-	✗

Table 1: Benchmarks
$✓\mathcal{R}$: proof of the relation $\mathcal{R}$ -: syntactically not in the scope of the tool
✗: in the scope but the tool fails to prove
U: Results using Ukano PV: Results using ProVerif Vanilla

on our simplified TLS models, even when considering no negotiation phase and no public key revealed, Ukano still fails to prove the two conditions discussed in Item 4.

Limitations. To go beyond our simplified TLS models, we also tried to apply our prototype on the ProVerif models of [5] which also proves unlinkability and anonymity of TLS clients. Their model describes in extensive details the protocol and consider some TLS extensions such as ECH and Pre-Shared Keys. Their proof is based on diff-equivalence but relies heavily on complex *restrictions* and manual reasoning that is, in our opinion, out of reach of standard ProVerif users. As such, with our prototype, we aimed to provide a fully automatic proof of client unlinkability that do not rely on such complex reasoning. However, our procedure showed a theoretical limitation as the TLS-ECH model in [5] relies on global states such as tables to encode the Pre-Shared Keys. Although we do not prevent the use of tables, as soon as the protocol need to *desynchronise* their access, our procedure fails to show equivalence. Such problems also occur when the protocol relies on desynchronised private channels (e.g., BAC with a private channel to distribute the key between reader and passport). In practice, the TLS-ECH model in [5] also raised issues by its size (several thousand lines of code) and our prototype could not complete the verification even after 48H of computation. There was already a bottleneck in the instrumentation and the clause generation.

8. Conclusion

We introduced new techniques to automatically verify process equivalences. As state-of-the-art tools for unbounded number of sessions are limited to the verification of *diff-equivalence*, our work is the first that is able to prove coarser-grained equivalences such as similarity and may-testing equivalence on syntactically unrestricted processes. We provided semantically sound conditions for proving these equivalences based on the set of saturated Horn clauses generated by ProVerif. We show how we satisfy these conditions in practice and implemented an extension of the tool.

Our work opens several directions for future work but, beyond those already discussed in the experimental section, one seems to be particularly interesting. One key hurdle (notably still pending even in the released version of ProVerif) is the handling of *trace restrictions* in equivalence proofs. In the context of may-testing, this would mean proving:

For all traces T_0 of P verifying φ_0, there exists an equivalent trace T_1 of Q verifying φ_1.

where φ_0, φ_1 are user specified trace properties. Such double restrictions are naturally incompatible with refinement-based proofs as in ProVerif, i.e., that rely on a fine-grained relation such as diff-equivalence to prove coarser ones. One possible direction would be to investigate whether an approach like ours directly targetting a coarser-grained, trace-based equivalence such as may-testing may circumvent such issues.

Acknowledgements. This work received funding from the France 2030 program managed by the French National Research Agency under grant agreement No. ANR-22-PECY-0006.

References

[1] B. Blanchet, B. Smyth, V. Cheval, and M. Sylvestre, *Automatic Cryptographic Protocol Verifier, User Manual and Tutorial*, available at https://prosecco.gforge.inria.fr/personal/bblanche/proverif/manual.pdf, 2020.

[2] D. Basin, C. Cremers, J. Dreier, S. Meier, R. Sasse, and B. Schmidt, *Tamarin prover manual*, available at https://tamarin-prover.github.io/, 2019.

[3] K. Bhargavan, B. Blanchet, and N. Kobeissi, "Verified models and reference implementations for the TLS 1.3 standard candidate," in *IEEE Symposium on Security and Privacy, (S&P)*, 2017.

[4] C. Cremers, M. Horvat, J. Hoyland, S. Scott, and T. van der Merwe, "A comprehensive symbolic analysis of TLS 1.3," in *ACM Conference on Computer and Communications Security (CCS)*, 2017.

[5] K. Bhargavan, V. Cheval, and C. A. Wood, "A symbolic analysis of privacy for TLS 1.3 with encrypted client hello," in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, CCS 2022, Los Angeles, CA, USA, November 7-11, 2022*, H. Yin, A. Stavrou, C. Cremers, and E. Shi, Eds. ACM, 2022, pp. 365–379. [Online]. Available: https://doi.org/10.1145/3548606.3559360

[6] D. A. Basin, J. Dreier, L. Hirschi, S. Radomirovic, R. Sasse, and V. Stettler, "A formal analysis of 5G authentication," in *ACM Conference on Computer and Communications Security (CCS)*, 2018.

[7] N. Kobeissi, K. Bhargavan, and B. Blanchet, "Automated verification for secure messaging protocols and their implementations: A symbolic and computational approach," in *IEEE European Symposium on Security and Privacy (EuroS&P)*, 2017.

[8] K. Cohn-Gordon, C. Cremers, L. Garratt, J. Millican, and K. Milner, "On ends-to-ends encryption: Asynchronous group messaging with strong security guarantees," in *ACM Conference on Computer and Communications Security (CCS)*, 2018.

[9] M. Abadi, B. Blanchet, and C. Fournet, "The applied pi calculus: Mobile values, new names, and secure communication," *Journal of the ACM (JACM)*, 2017.

[10] V. Cheval, S. Kremer, and I. Rakotonirina, "Exploiting symmetries when proving equivalence properties for security protocols," in *ACM Conference on Computer and Communications Security (CCS)*, 2019.

[11] S. Santiago, S. Escobar, C. Meadows, and J. Meseguer, "A formal definition of protocol indistinguishability and its verification using maude-npa," in *International Workshop on Security and Trust Management*, 2014.

[12] L. Hirschi, D. Baelde, and S. Delaune, "A method for unbounded verification of privacy-type properties," *J. Comput. Secur.*, 2019.

[13] D. Baelde, S. Delaune, and S. Moreau, "A method for proving unlinkability of stateful protocols," in *IEEE Computer Security Foundations Symposium (CSF)*, 2020.

[14] V. Cheval and I. Rakotonirina, "Indistinguishability beyond diff-equivalence in proverif," 2023, Technical Report and implementation. https://www.dropbox.com/sh/t0bwuppbjab0dka/AADFc5E-zDlLPMyvF80fa1Sua?dl=0.

[15] R. Chadha, Ş. Ciobâcă, and S. Kremer, "Automated verification of equivalence properties of cryptographic protocols," in *Programming Languages and Systems —Proceedings of the 21th European Symposium on Programming (ESOP'12)*, 2012.

[16] V. Cortier, S. Delaune, and A. Dallon, "Sat-equiv: an efficient tool for equivalence properties," in *Proceedings of the 30th IEEE Computer Security Foundations Symposium (CSF'17)*, 2017.

[17] V. Cheval, S. Kremer, and I. Rakotonirina, "DEEPSEC: Deciding equivalence properties in security protocols - theory and practice," in *IEEE Symposium on Security and Privacy (S&P)*, 2018.

[18] V. Cheval and B. Blanchet, "Proving more observational equivalences with proverif," in *Proceedings of the International Conference on Principles of Security and Trust (POST)*, 2013.

[19] B. Blanchet and B. Smyth, "Automated reasoning for equivalences in the applied pi calculus with barriers," in *CSF 2016*, 2016.

[20] PKI-Task-Force, "PKI for machine readable travel documents offering ICC read-only access," International Civil Aviation Organization, Tech. Rep., 2004.

[21] B. Blanchet, V. Cheval, and V. Cortier, "Proverif with lemmas, induction, fast subsumption, and much more," in *Proceedings of the 43th IEEE Symposium on Security and Privacy (S&P'22)*. IEEE Computer Society Press, May 2022.

[22] V. Cheval, V. Cortier, and M. Turuani, "A little more conversation, a little less action, a lot more satisfaction: Global states in proverif," in *Proceedings of the 31st IEEE Computer Security Foundations Symposium (CSF'18)*. Oxford, UK: IEEE Computer Society Press, Jul. 2018.

[23] B. Blanchet, "Automatic verification of correspondences for security protocols," *Journal of Computer Security*, vol. 17, no. 4, pp. 363–434, 2009.

[24] A. Juels and S. A. Weis, "Defining strong privacy for RFID," *ACM Trans. Inf. Syst. Secur.*, vol. 13, no. 1, pp. 7:1–7:23, 2009. [Online]. Available: https://doi.org/10.1145/1609956.1609963

[25] T. van Deursen and S. Radomirovic, "Attacks on RFID protocols," *IACR Cryptol. ePrint Arch.*, p. 310, 2008. [Online]. Available: http://eprint.iacr.org/2008/310

[26] J. Bender, M. Fischlin, and D. Kügler, "Security analysis of the PACE key-agreement protocol," in *Information Security, 12th International Conference, ISC 2009, Pisa, Italy, September 7-9, 2009. Proceedings*, 2009.

[27] J. Bender, Ö. Dagdelen, M. Fischlin, and D. Kügler, "The PACE|aa protocol for machine readable travel documents, and its security," in *Financial Cryptography and Data Security - 16th International Conference, FC 2012, Kralendijk, Bonaire, Februray 27-March 2, 2012, Revised Selected Papers*, 2012.

[28] V. Cortier, N. Grimm, J. Lallemand, and M. Maffei, "A type system for privacy properties," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*, 2017.

[29] M. Feldhofer, S. Dominikus, and J. Wolkerstorfer, "Strong authentication for RFID systems using the AES algorithm," in *Cryptographic Hardware and Embedded Systems - CHES 2004: 6th International Workshop Cambridge, MA, USA, August 11-13, 2004. Proceedings*, M. Joye and J. Quisquater, Eds., 2004.

Appendix A. Term Algebra and Instrumentation

First we give the full definition of the evaluation of expressions:

Definition 17 (Evaluation of expressions)**.** We assume, for each destructor g, an ordered list of rewrite rules

$$\mathsf{def}(g) = [g(U_{i,1}, \dots, U_{i,n}) \to U_i]_{i=1}^{k}$$

where $U_{i,j}, U_j$ are either terms or the constant fail. We say that the expression D *evaluates* to U (U being either a term or the constant fail), denoted $D \Downarrow U$, by:

- if $D \in \mathcal{V} \cup \mathcal{N} \cup \{\mathsf{fail}\}$ then $D \Downarrow D$;
- if $D = f(D_1, \dots, D_n)$ and $D_1 \Downarrow V_1, \dots, D_n \Downarrow V_n$:
  - if $f \in \mathcal{F}_c$ and $\mathsf{fail} \in \{V_1, \dots, V_n\}$ then $D \Downarrow \mathsf{fail}$;
  - otherwise if $f \in \mathcal{F}_c$ then $D \Downarrow f(V_1, \dots, V_n)$;
  - otherwise $f \in \mathcal{F}_d$ and we let $D' = g(V_1, \dots, V_n)$. Then either D' cannot be rewritten by any rule of $\mathsf{def}(g)$ and $D \Downarrow \mathsf{fail}$, or otherwise $D \Downarrow V$ where $D' \to V$ by the first rule of $\mathsf{def}(g)$ applicable to D'.

Figures 2, 3, and 4 respectively describe the instrumentation, and the instrumented semantics on (bi)configurations.

Appendix B. Clause Generation

We detail below the complete set of clauses used for the decision of equivalences. First of all, we display the clauses describing the adversary's capabilities. They are mostly adapted from [18], [21]. We consider the set of public names $\mathcal{A}_0$ in the initial biconfiguration $\mathcal{C} = (\{\!\{(P,Q)\}\!\}, \mathcal{A}_0, \emptyset)$.

For each $a \in \mathcal{A}_0, \mathrm{att}(a[], a[])$ (RInit)
$\mathrm{att}(b_0[i], b_0[i])$ (RGen)
$\mathrm{att}(\mathsf{fail}, \mathsf{fail})$ (RFail)
For each h, for all $h(U_1, \dots, U_m) \to U \mid\mid \phi$ in $\mathsf{def}(h)$,
for all $h(U'_1, \dots, U'_m) \to U' \mid\mid \phi'$ in $\mathsf{def}(h)$,
$\bigwedge_{i=1}^{m} \mathrm{att}(U_i, U'_i) \wedge \phi \wedge \phi' \to \mathrm{att}(U, U')$ (Rf)
$\mathrm{msg}(x, y, x', y') \wedge \mathrm{att}(x, x') \to \mathrm{att}(y, y')$ (Rl)
$\mathrm{att}(x, x') \wedge \mathrm{att}(y, y') \to \mathrm{msg}(x, y, x', y')$ (Rs)
$\mathrm{att}(x, x') \to \mathrm{input}(x, x')$ (RIn)
$\mathrm{input}(x, y) \wedge \mathrm{msg}(x, z, y', z') \wedge y \neq y' \to \mathrm{bad}$ (RIBad1)
$\mathrm{input}(y, x) \wedge \mathrm{msg}(y', z, x, z') \wedge y \neq y' \to \mathrm{bad}$ (RIBad2)
$\mathrm{att}(x, \mathsf{fail}) \to \mathrm{bad}$ (RBad1)
$\mathrm{att}(\mathsf{fail}, x) \to \mathrm{bad}$ (RBad2)

We will denote $\mathbb{C}_{\mathcal{A}}(\mathcal{C}) = \{$(RInit), (RGen), (RFail), (Rf), (Rl), (Rs), (RIn), (RIBad1), (RIBad2), (RBad1), (RBad2)$\}$.

We then give the definition of the clause generation $[\![|P,Q|]\!]\mathcal{H}r$ in Figure 5. We recall in particular the notation from Section 5:

- $F_{!_i}(r, x, x') = \top$ when $r = \square$ and otherwise:
  $$F_{!_i}(r, x, x') = \mathrm{ev}(repl_i(\mathsf{proj}_0(r), x), repl_i(\mathsf{proj}_1(r), x'))$$
- $F_!(r) = \top$ when $r = \square$ and otherwise:
  $$F_!(r) = \mathrm{ev}(repl(\mathsf{proj}_0(r)), repl(\mathsf{proj}_1(r)))$$

We define an evaluation on open terms, i.e. terms with variables, that we use in the generation of Horn clauses. We support in particular a ProVerif feature where, if $\ell \to r$ is a rewriting rule defining one behaviour of the destructor g, the user may write

$$\ell \to r \mid\mid \phi \quad \in \mathsf{def}(g)$$

to mean that the rewriting rule may only be applied under the condition ϕ. We omit the details of specification formalism for ϕ here as it is irrelevant for our contributions.

$U \Downarrow^2 (U, \emptyset, \top)$ if U is a may-fail term

$g(D_1, \dots, D_n) \Downarrow^2 (V\sigma_u, \sigma'\sigma_u, \phi'\sigma_u \wedge \phi\sigma_u)$
if $(D_1, \dots, D_n) \Downarrow^2 ((U_1, \dots, U_n), \sigma', \phi')$,
$g(V_1, \dots, V_n) \to V \mid\mid \phi \in \mathsf{def}(g)$ and
σ_u is the most general unifier of (U_1, V_1), …, (U_n, V_n)

$(D_1, \dots, D_n) \Downarrow^2 ((U_1\sigma_n, \dots, U_{n-1}\sigma_n, U_n), \sigma\sigma_n, \phi\sigma_n \wedge \phi_n)$
if $(D_1, \dots, D_{n-1}) \Downarrow^2 ((U_1, \dots, U_{n-1}), \sigma, \phi)$
and $D_n\sigma \Downarrow^2 (U_n, \sigma_n, \phi_n)$

$$0 \Downarrow_{\tilde{a}} 0$$
$$\mathsf{out}(N,M);P \Downarrow_{\tilde{a}} \mathsf{out}(N,M);P' \qquad \text{if } P \Downarrow_{\tilde{a}} P'$$
$$\mathsf{in}(N,x);P \Downarrow_{\tilde{a}} \mathsf{in}^{o[\tilde{a}|\lambda]}(N,x);P' \qquad \text{if } P \Downarrow_{\tilde{a}\cdot x} P' \text{ and } o \in \mathcal{N}_{\mathsf{in}} \text{ fresh}$$
$$\mathsf{event}(M);P \Downarrow_{\tilde{a}} \mathsf{event}(M);P' \qquad \text{if } P \Downarrow_{\tilde{a}} P'$$
$$\mathsf{new}\, n;P \Downarrow_{\tilde{a}} P' \qquad \text{if } P\{^{n'[\tilde{a}]}/_{n}\} \Downarrow_{\tilde{a}} P' \qquad n' \text{ fresh name pattern}$$
$$\mathsf{let}\ x = D\ \mathsf{in}\ P\ \mathsf{else}\ Q \Downarrow_{\tilde{a}} \mathsf{let}\ x = D\ \mathsf{in}\ P'\ \mathsf{else}\ Q' \qquad \text{if } P \Downarrow_{\tilde{a}} P' \text{ and } Q \Downarrow_{\tilde{a}} Q'$$
$$P \mid Q \Downarrow_{\tilde{a}} \{\!\!\{ !^{o[\tilde{a}]}_{\emptyset} P' \}\!\!\} \mid \{\!\!\{ !^{o'[\tilde{a}]}_{\emptyset} Q' \}\!\!\} \qquad \text{if } P \Downarrow_{\tilde{a}} P',\ Q \Downarrow_{\tilde{a}} Q', \text{ and } o, o' \in \mathcal{N} \text{ are fresh}$$
$$!P \Downarrow_{\tilde{a}} \{\!\!\{ !^{o[\tilde{a},i]}_{\{i\}} P' \}\!\!\} \qquad \text{if } P \Downarrow_{\tilde{a}\cdot i} P', \text{ and } o \in \mathcal{N}, i \in \mathcal{X}_{\lambda} \text{ fresh}$$

Figure 2: Transformation into Instrumented Processes

$$\{\!\!\{0\}\!\!\}, \mathcal{A}, \Lambda \rightarrow_i \emptyset, \mathcal{A}, \Lambda \tag{I-Nil}$$
$$\{\!\!\{P \mid Q\}\!\!\}, \mathcal{A}, \Lambda \rightarrow_i \{\!\!\{P, Q\}\!\!\}, \mathcal{A}, \Lambda \tag{I-Par}$$
$$\{\!\!\{\{\!\!\{!^{\overline{o}}_{\tilde{a}} P\}\!\!\} \cup S\}\!\!\}, \mathcal{A}, \Lambda \xrightarrow{repl(\overline{o}\sigma)}_i \{\!\!\{P\sigma, \{\!\!\{!^{\overline{o}}_{\tilde{a}} P\}\!\!\} \cup S\}\!\!\}, \mathcal{A}, \Lambda \cup \{\overline{o}\sigma\} \qquad \text{if } dom(\sigma) = \tilde{a},\ img(\sigma) \subseteq \mathbb{N},\ \overline{o}\sigma \notin \Lambda \tag{I-Repl}$$
$$\{\!\!\{\mathsf{let}\ x = D\ \mathsf{in}\ P\ \mathsf{else}\ Q\}\!\!\}, \mathcal{A}, \Lambda \rightarrow_i \{\!\!\{R\}\!\!\}, \mathcal{A}, \Lambda \qquad \text{with } R = \{\!\!\{P\{^M/_x\}\}\!\!\} \text{ if } D \Downarrow M \text{ and } R = Q \text{ if } D \Downarrow \mathsf{fail} \tag{I-Let}$$
$$\{\!\!\{\mathsf{out}(N,M);P\}\!\!\}, \mathcal{A}, \Lambda \rightarrow_i \{\!\!\{P\}\!\!\}, \mathcal{A} \cup \{M\}, \Lambda \qquad \text{if } N \in \mathcal{A} \tag{I-Out}$$
$$\{\!\!\{\mathsf{in}^{\overline{o}}(N,x);Q\}\!\!\}, \mathcal{A}, \Lambda \xrightarrow{pre(\overline{o},M)}_i \{\!\!\{Q\{^M/_x\}\}\!\!\}, \mathcal{A}, \Lambda \qquad \text{if } N, M \in \mathcal{A} \tag{I-In}$$
$$\{\!\!\{\mathsf{event}(M);P\}\!\!\}, \mathcal{A}, \Lambda \xrightarrow{M}_i \{\!\!\{P\}\!\!\}, \mathcal{A}, \Lambda \tag{I-Event}$$
$$\emptyset, \mathcal{A}, \Lambda \rightarrow_i \emptyset, \mathcal{A} \cup \{M\}, \Lambda \qquad \text{if } M_1, \ldots, M_n \in \mathcal{A},\ f/n \in \mathcal{F}_c \cup \mathcal{F}_d \text{ and } f(M_1, \ldots, M_n) \Downarrow M \tag{I-App}$$
$$\emptyset, \mathcal{A}, \Lambda \rightarrow_i \emptyset, \mathcal{A} \cup \{b_0[\lambda]\}, \Lambda \qquad \text{with } b_0 \text{ a fixed name not appearing in } P \text{ or } Q, \text{ and } b_0[\lambda] \notin \mathcal{A},\ \lambda \text{ session identifier} \tag{I-Gen}$$
$$\mathcal{P} \cup \mathcal{Q}, \mathcal{A}, \Lambda \xrightarrow{\alpha}_i \mathcal{P}' \cup \mathcal{Q}, \mathcal{A}', \Lambda' \qquad \text{if } \mathcal{P}, \mathcal{A}, \Lambda \xrightarrow{\alpha}_i \mathcal{P}', \mathcal{A}', \Lambda' \tag{I-Cont}$$

Figure 3: Instrumented Semantics on Configurations

$$\{\!\!\{(0,0)\}\!\!\}, \mathcal{A}^2, \Lambda^2 \rightarrow_{i^2} \emptyset, \mathcal{A}^2, \Lambda^2 \tag{I^2-Nil}$$
$$\{\!\!\{(P \mid Q, P' \mid Q')\}\!\!\}, \mathcal{A}^2, \Lambda^2 \rightarrow_{i^2} \{\!\!\{(P,P'), (Q,Q')\}\!\!\}, \mathcal{A}^2, \Lambda^2 \tag{I^2-Par}$$
$$\begin{aligned} &\{\!\!\{(\{\!\!\{!^{\overline{o}}_{\tilde{a}} P\}\!\!\} \cup S, \{\!\!\{!^{\overline{o'}}_{\tilde{a}'} P'\}\!\!\} \cup S')\}\!\!\}, \mathcal{A}^2, \Lambda^2 \xrightarrow{(repl(\overline{o}\sigma), repl(\overline{o'}\sigma'))}_{i^2} \\ &\quad \{\!\!\{(P\sigma, P'\sigma'), (\{\!\!\{!^{\overline{o}}_{\tilde{a}} P\}\!\!\} \cup S, \{\!\!\{!^{\overline{o'}}_{\tilde{a}'} P'\}\!\!\} \cup S')\}\!\!\}, \mathcal{A}^2, \Lambda^2 \cup \{(\overline{o}\sigma, \overline{o'}\sigma')\} \\ &\quad \text{if } dom(\sigma) = \tilde{a},\ dom(\sigma') = \tilde{a}',\ img(\sigma) \cup img(\sigma') \subseteq \mathbb{N},\ \overline{o}\sigma \notin \mathsf{proj}_0(\Lambda^2),\ \overline{o'}\sigma' \notin \mathsf{proj}_1(\Lambda^2) \end{aligned} \tag{I^2-Repl}$$
$$\begin{aligned} &\{\!\!\{(\mathsf{let}\ x = D\ \mathsf{in}\ P\ \mathsf{else}\ Q, \mathsf{let}\ x' = D'\ \mathsf{in}\ P'\ \mathsf{else}\ Q')\}\!\!\}, \mathcal{A}^2, \Lambda^2 \rightarrow_{i^2} \{\!\!\{(R, R')\}\!\!\}, \mathcal{A}^2, \Lambda^2 \\ &\quad \text{with } (R,R') = (P\{^M/_x\}, P'\{^{M'}/_{x'}\}) \text{ if } D \Downarrow M \text{ and } D' \Downarrow M', \text{ and } (R,R') = (Q,Q') \text{ if } D \Downarrow \mathsf{fail} \text{ and } D' \Downarrow \mathsf{fail} \end{aligned} \tag{I^2-Let}$$
$$\{\!\!\{(\mathsf{out}(N,M);P, \mathsf{out}(N',M');P')\}\!\!\}, \mathcal{A}^2, \Lambda^2 \rightarrow_{i^2} \{\!\!\{(P,P')\}\!\!\}, \mathcal{A}^2, \Lambda^2 \cup \{(M,M')\} \qquad \text{if } (N,N') \in \mathcal{A}^2 \tag{I^2-Out}$$
$$\begin{aligned} &\{\!\!\{(\mathsf{in}^{\overline{o}}(N,x);Q, \mathsf{in}^{\overline{o'}}(N',x');Q')\}\!\!\}, \mathcal{A}^2, \Lambda^2 \xrightarrow{(pre(\overline{o},M), pre(\overline{o'},M'))}_{i^2} \{\!\!\{(Q\{^M/_x\}, Q'\{^{M'}/_{x'}\})\}\!\!\}, \mathcal{A}^2, \Lambda^2 \\ &\quad \text{if } (N,N'), (M,M') \in \mathcal{A}^2 \end{aligned} \tag{I^2-In}$$
$$\{\!\!\{(\mathsf{event}(M);Q, \mathsf{event}(M');Q')\}\!\!\}, \mathcal{A}^2, \Lambda^2 \xrightarrow{(M,M')}_{i^2} \{\!\!\{(Q,Q')\}\!\!\}, \mathcal{A}^2, \Lambda^2 \tag{I^2-Event}$$
$$\begin{aligned} &\emptyset, \mathcal{A}^2, \Lambda^2 \rightarrow_{i^2} \emptyset, \mathcal{A}^2 \cup \{(M,M')\}, \Lambda^2 \\ &\quad \text{if } (M_1,M_1'), \ldots, (M_n,M_n') \in \mathcal{A},\ f/n \in \mathcal{F}_c \cup \mathcal{F}_d \text{ and } f(M_1,\ldots,M_n) \Downarrow M \text{ and } f(M_1',\ldots,M_n') \Downarrow M' \end{aligned} \tag{I^2-App}$$
$$\emptyset, \mathcal{A}^2, \Lambda^2 \rightarrow_{i^2} \emptyset, \mathcal{A}^2 \cup \{(b_0[\lambda], b_0[\lambda])\}, \Lambda^2 \qquad \text{if } (b_0[\lambda], b_0[\lambda]) \notin \mathcal{A}^2 \tag{I^2-Gen}$$
$$\mathcal{P}^2 \cup \mathcal{Q}^2, \mathcal{A}^2, \Lambda^2 \xrightarrow{\alpha}_{i^2} \mathcal{P}^{2\prime} \cup \mathcal{Q}^2, \mathcal{A}^{2\prime}, \Lambda^{2\prime} \qquad \text{if } \mathcal{P}^2, \mathcal{A}^2, \Lambda^2 \rightarrow_{i^2} \mathcal{P}^{2\prime}, \mathcal{A}^{2\prime}, \Lambda^{2\prime} \tag{I^2-Cont}$$

Figure 4: Instrumented Semantics on (Well-Formed) Biconfigurations

$$\llbracket|0,0|\rrbracket \mathcal{H} r = \emptyset$$

$$\llbracket|S,S'|\rrbracket \mathcal{H} r = \bigcup_{!^{\overline{o_1}}_{\tilde{a}_1} P_1 \in S} \; \bigcup_{!^{\overline{o_2}}_{\tilde{a}_2} P_2 \in S'} \llbracket|P_1,P_2|\rrbracket \mathcal{H} \mathsf{diff}[\overline{o_1},\overline{o_2}]$$

$$\llbracket|P \mid Q, P' \mid Q'|\rrbracket \mathcal{H} r = \llbracket|P,P'|\rrbracket \mathcal{H} \square \cup \llbracket|Q,Q'|\rrbracket \mathcal{H} \square$$

$$\llbracket|\mathsf{in}^{\overline{o}}(N,x);P,\mathsf{in}^{\overline{o'}}(N',x');P'|\rrbracket \mathcal{H} r = \llbracket|P,P'|\rrbracket \mathcal{H}_1 \square \cup \{\mathcal{H} \wedge F_{!}(r) \rightarrow \mathrm{input}(N,N')\}$$

$$\text{with } \mathcal{H}_1 = \mathcal{H} \wedge \mathrm{msg}(N,x,N',x') \wedge F_{!_i}(r,x,x') \;\wedge\; \mathrm{ev}(\mathit{pre}(\overline{o},x),\mathit{pre}(\overline{o'},x'))$$

$$\llbracket|\mathsf{out}(N,M);P,\mathsf{out}(N',M');P'|\rrbracket \mathcal{H} r = \llbracket|P,P'|\rrbracket (\mathcal{H} \wedge F_{!}(r)) \square \cup \{\mathcal{H} \wedge F_{!}(r) \rightarrow \mathrm{msg}(N,M,N',M')\}$$

$$\llbracket|\mathsf{event}(ev);P,\mathsf{event}(ev');P'|\rrbracket \mathcal{H} r = \llbracket|P,P'|\rrbracket (\mathcal{H} \wedge F_{!}(r) \wedge \mathrm{ev}(ev,ev')) \square$$

$$\llbracket|\mathsf{let}\ x = D\ \mathsf{in}\ P\ \mathsf{else}\ Q, \mathsf{let}\ x' = D'\ \mathsf{in}\ P'\ \mathsf{else}\ Q'|\rrbracket \mathcal{H} r =$$

$$\bigcup \{\llbracket|P\sigma\sigma', P'\sigma\sigma'|\rrbracket (\mathcal{H}\sigma \wedge F_{!}(r)\sigma \wedge \phi) \square \mid$$

$$(D,D') \Downarrow^2 ((M,M'),\sigma,\phi) \wedge \sigma' = \{^{M}/_{x}, ^{M'}/_{x'}\}\}$$

$$\cup \bigcup \{\llbracket|Q\sigma, Q'\sigma|\rrbracket (\mathcal{H}\sigma \wedge F_{!}(r)\sigma \wedge \phi) \square \mid$$

$$(D,D') \Downarrow^2 ((\mathsf{fail},\mathsf{fail}),\sigma,\phi)\}$$

$$\cup \{\mathcal{H}\sigma \wedge F_{!}(r)\sigma \wedge \phi \rightarrow \mathrm{bad} \mid (D,D') \Downarrow^2 ((\mathsf{fail},M),\sigma,\phi)\}$$

$$\cup \{\mathcal{H}\sigma \wedge F_{!}(r)\sigma \wedge \phi \rightarrow \mathrm{bad} \mid (D,D') \Downarrow^2 ((M,\mathsf{fail}),\sigma,\phi)\}$$

Figure 5: Translation of Processes Into Clauses