Automated reasoning for equivalences in the applied pi calculus with barriers

Bruno Blanchet
Inria, Paris, France
Email: Bruno.Blanchet@inria.fr

Ben Smyth
Huawei Technologies Co. Ltd., France
Email: research@bensmyth.com

Abstract*—Observational equivalence allows us to study important security properties such as anonymity. Unfortunately, the difficulty of proving observational equivalence hinders analysis. Blanchet, Abadi & Fournet simplify its proof by introducing a sufficient condition for observational equivalence, called diff-equivalence, which is a reachability condition that can be proved automatically by ProVerif. However, diff-equivalence is a very strong condition, which often does not hold even if observational equivalence does. In particular, when proving equivalence between processes that contain several parallel components, e.g., $P \mid Q$ and $P' \mid Q'$, diff-equivalence requires that P is equivalent to P' and Q is equivalent to Q'. To relax this constraint, Delaune, Ryan & Smyth introduced the idea of swapping data between parallel processes P' and Q' at synchronisation points, without proving its soundness. We extend their work by formalising the semantics of synchronisation, formalising the definition of swapping, and proving its soundness. We also relax some restrictions they had on the processes to which swapping can be applied. Moreover, we have implemented our results in ProVerif. Hence, we extend the class of equivalences that can be proved automatically. We showcase our results by analysing privacy in election schemes by Fujioka, Okamoto & Ohta and Lee *et al.*, and in the vehicular ad-hoc network by Freudiger *et al.

I. Introduction

Cryptographic protocols are required to satisfy a plethora of security requirements. These requirements include classical properties such as secrecy and authentication, and emerging properties including anonymity [1], [2], [3], ideal functionality [4], [5], [6], and stronger notions of secrecy [7], [8], [9]. These security requirements can generally be classified as *indistinguishability* or *reachability* properties. Reachability properties express requirements of a protocol's reachable states. For example, secrecy can be expressed as the inability of deriving a particular value from any possible protocol execution. By comparison, indistinguishability properties express requirements of a protocol's observable behaviour. Intuitively, two protocols are said to be indistinguishable if an observer has no way of telling them apart. Indistinguishability enables the formulation of more complex properties. For example, anonymity can be expressed as the inability to distinguish between an instance of the protocol in which actions are performed by a user, from another instance in which actions are performed by another user.

Indistinguishability can be formalised as observational equivalence, denoted $\approx$. As a motivating example, consider an election scheme, in which a voter A voting v is formalised by a process $V(A, v)$. Ballot secrecy can be formalised by the equivalence

$$V(A, v) \mid V(B, v') \approx V(A, v') \mid V(B, v) \qquad (1)$$

which means that no adversary can distinguish when two voters swap their votes [2]. (We use the applied pi calculus syntax and terminology [5], which we introduce in Section II.)

A. Approaches to proving equivalences

Observational equivalence is the tool introduced for reasoning about security requirements of cryptographic protocols in the spi calculus [4] and in the applied pi calculus [5]. It was originally proved manually, using the notion of labelled bisimilarity [5], [10], [11] to avoid universal quantification over adversaries.

Manual proofs of equivalence are long and difficult, so automating these proofs is desirable. Automation often relies on symbolic semantics [12], [13] to avoid the infinite branching due to messages sent by the adversary by treating these messages as variables. For a bounded number of sessions, several decision procedures have been proposed for processes without else branches, first for a fixed set of primitives [14], [15], then for a wide variety of primitives with the restriction that processes are determinate, that is, their execution is entirely determined by the adversary inputs [16]. These decision procedures are too complex for useful implementations. Practical algorithms have since been proposed and implemented: SPEC [17] for fixed primitives and without else branches, APTE [18] for fixed primitives with else branches and non-determinism, and AKISS [19], [20] for a wide variety of primitives and determinate processes.

For an unbounded number of sessions, proving equivalence is an undecidable problem [14], [21], so automated proof techniques are incomplete. ProVerif automatically proves an equivalence notion, named diff-equivalence, between processes P and Q that share the same structure and differ only in the choice of terms [22]. Diff-equivalence requires that the two processes always reduce in the same way, in the presence of any adversary. In particular, the two processes must have the same branching behaviour. Hence, diff-equivalence is much stronger than observational equivalence. Maude-NPA [23] and Tamarin [24] also use that notion, and Baudet [25] showed that diff-equivalence is decidable for a bounded number of sessions and used this technique for proving resistance against off-line guessing attacks [26]. Decision procedures also exist for restricted classes of protocols: for an unbounded number of sessions, trace equivalence has a decision procedure for symmetric-key, type-compliant, acyclic protocols [27], which is too complex for useful implementation, and for ping-pong protocols [28], which is implemented in a tool.

B. Diff-equivalence and its limitations

The main approach to automate proofs of observational equivalence with an unbounded number of sessions is to use diff-equivalence. (In our motivating example (1), a bounded number of sessions is sufficient, but an unbounded number becomes useful in more complex examples, as in Section IV-B.) Diff-equivalence seems well-suited to our motivating example, since the processes $V(A,v) \mid V(B,v')$ and $V(A,v') \mid V(B,v)$ differ only by their terms. Such a pair of processes can be represented as a *biprocess* which has the same structure as each of the processes and captures the differences in terms using the construct $\mathsf{diff}[M,M']$, denoting the occurrence of a term M in the first process and a term M' in the second. For example, the pair of processes in our motivating example can be represented as the biprocess $P_1 \triangleq V(A, \mathsf{diff}[v,v']) \mid V(B, \mathsf{diff}[v',v])$. The two processes represented by a biprocess P are recovered by $\mathsf{fst}(P)$ and $\mathsf{snd}(P)$. Hence, $\mathsf{fst}(P_1) = V(A,v) \mid V(B,v')$ and $\mathsf{snd}(P_1) = V(A,v') \mid V(B,v)$.

Diff-equivalence implies observational equivalence. Hence, the equivalence (1) can be inferred from the diff-equivalence of the biprocess P_1. However, diff-equivalence is so strong that it does not hold for biprocesses modelling even trivial schemes, as the following example demonstrates.

Example 1. Consider an election scheme that instructs voters to publish their vote on an anonymous channel. The voter's role can be formalised as $V(A,v) = \overline{c}\langle v\rangle$. Thus, ballot secrecy can be analysed using the biprocess $P \triangleq \overline{c}\langle \mathsf{diff}[v,v']\rangle \mid \overline{c}\langle \mathsf{diff}[v',v]\rangle$. It is trivial to see that $\mathsf{fst}(P) = \overline{c}\langle v\rangle \mid \overline{c}\langle v'\rangle$ is indistinguishable from $\mathsf{snd}(P) = \overline{c}\langle v'\rangle \mid \overline{c}\langle v\rangle$, because any output by $\mathsf{fst}(P)$ can be matched by an output from $\mathsf{snd}(P)$, and vice-versa. However, the biprocess P does not satisfy diff-equivalence. Intuitively, this is because diff-equivalence requires that the subprocesses of the parallel composition, namely, $\overline{c}\langle \mathsf{diff}[v,v']\rangle$ and $\overline{c}\langle \mathsf{diff}[v',v]\rangle$, each satisfy diff-equivalence, which is false, because $\overline{c}\langle v\rangle$ is not equivalent to $\overline{c}\langle v'\rangle$ (nor is $\overline{c}\langle v'\rangle$ equivalent to $\overline{c}\langle v\rangle$).

Overcoming the difficulty encountered in Example 1 is straightforward: using the general property that $P \mid Q \approx Q \mid P$, we can instead prove

$$V(A,v) \mid V(B,v') \approx V(B,v) \mid V(A,v')$$

which, in the case of Example 1, is proved by noticing that the two sides of the equivalence are equal, i.e., by noticing that the biprocess $\hat{P} \triangleq \overline{c}\langle \mathsf{diff}[v,v]\rangle \mid \overline{c}\langle \mathsf{diff}[v',v']\rangle$ trivially satisfies diff-equivalence, since $\mathsf{fst}(\hat{P}) = \mathsf{snd}(\hat{P})$. However, this technique cannot be applied to more complex examples, as we show below.

Some security properties (e.g., privacy in elections [2], [29], vehicular ad-hoc networks [3], [30], and anonymity networks [1], [31], [32]) can only be realised if processes synchronise their actions in a specific manner.

Example 2. Building upon Example 1, suppose each voter sends their identity, then their vote, both on an anonymous channel, i.e., $V(A,v) = \overline{c}\langle A\rangle.\overline{c}\langle v\rangle$. This example does not satisfy ballot secrecy, because $V(A,v) \mid V(B,v')$ can output A, v, B, v' on channel c in that order, while $V(A,v') \mid V(B,v)$ cannot.

To modify this example so that it satisfies ballot secrecy, we use the notion of barrier synchronisation, which ensures that a process will block, when a barrier is encountered, until all other processes executing in parallel reach this barrier [33], [34], [35], [36].

Example 3. Let us modify the previous example so that voters publish their identity, synchronise with other voters, and publish their vote on an anonymous channel. The voter's role can be formalised as process $V(A,v) = \overline{c}\langle A\rangle.1{::}\,\overline{c}\langle v\rangle$, where $1{::}$ is a barrier synchronisation. Ballot secrecy can then be analysed using biprocess $P_{\mathsf{ex}} \triangleq \overline{c}\langle A\rangle.1{::}\,\overline{c}\langle \mathsf{diff}[v,v']\rangle \mid \overline{c}\langle B\rangle.1{::}\,\overline{c}\langle \mathsf{diff}[v',v]\rangle$. Synchronisation ensures the output of A and B, prior to v and v', in both $\mathsf{fst}(P_{\mathsf{ex}})$ and $\mathsf{snd}(P_{\mathsf{ex}})$, so that ballot secrecy holds, but diff-equivalence does not hold.

The technique used to overcome the difficulty in Example 1 cannot be applied here, because swapping the two voting processes leads to the biprocess $P'_{\mathsf{ex}} \triangleq \overline{c}\langle \mathsf{diff}[A,B]\rangle.1{::}\,\overline{c}\langle v\rangle \mid \overline{c}\langle \mathsf{diff}[B,A]\rangle.1{::}\,\overline{c}\langle v'\rangle$, which does not satisfy diff-equivalence. Intuitively, we need to swap at the barrier, not at the beginning (cf. P'_{ex}). In essence, by swapping data between the two voting processes at the barrier, it suffices to prove that the biprocess $P''_{\mathsf{ex}} \triangleq \overline{c}\langle A\rangle.1{::}\,\overline{c}\langle \mathsf{diff}[v,v]\rangle \mid \overline{c}\langle B\rangle.1{::}\,\overline{c}\langle \mathsf{diff}[v',v']\rangle$ satisfies diff-equivalence, which trivially holds since $\mathsf{fst}(P''_{\mathsf{ex}}) = \mathsf{snd}(P''_{\mathsf{ex}})$.

As illustrated in Examples 1 & 3, diff-equivalence is a sufficient condition for observational equivalence, but it is not necessary, and this precludes the analysis of interesting security properties. In this paper, we will partly overcome this limitation: we weaken the diff-equivalence requirement by allowing swapping of data between processes at barriers.

C. Contributions

First, we extend the process calculus by Blanchet, Abadi & Fournet [22] to capture barriers (Section II). Secondly, we formally define a compiler that encodes barriers and swapping using private channel communication (Section III). As a by-product, if we compile without swapping, we also obtain an encoding of barriers into the calculus without barriers, via private channel communication. Thirdly, we provide a detailed soundness proof for this compiler. (Details of the proof are in the long version of this paper [37].) Fourthly, we have implemented our compiler in ProVerif. Hence, we extend the class of equivalences that can be proved automatically. Finally, we analyse privacy in election schemes and in a vehicular ad-hoc network to showcase our results (Section IV).

D. Comparison with Delaune, Ryan & Smyth

The idea of swapping data at barriers was informally introduced by Delaune, Ryan & Smyth [38], [39]. Our contributions improve upon their work by providing a strong theoretical foundation to their idea. In particular, they do not provide a soundness proof, we do; they prohibit replication and place restrictions on control flow and parallel composition, we relax these conditions; and they did not implement their results, we implement ours. (Smyth presented a preliminary version of our compiler in his thesis [40, Chapter 5], and Klus, Smyth & Ryan implemented that compiler [41].)

II. Process calculus

We recall Blanchet, Abadi & Fournet's dialect [22] of the applied pi calculus [5], [42]. This dialect is particularly useful due to the automated support provided by ProVerif [43]. The semantics of the applied pi calculus [5] and the dialect of [22] were defined using structural equivalence. Those semantics have been simplified by semantics with configurations and without structural equivalence, first for trace properties [44], then for equivalences [25], [45], [46]. In this paper, we use the latter semantics. In addition, we extend the calculus to capture barrier synchronisation, by giving the syntax and formal semantics of barriers.

A. Syntax and semantics

The calculus assumes an infinite set of *names*, an infinite set of *variables*, and a finite set of *function symbols* (*constructors* and *destructors*), each with an associated arity. We write f for a constructor, g for a destructor, and h for a constructor or destructor; constructors are used to build terms, whereas destructors are used to manipulate terms in expressions. Thus, *terms* range over names, variables, and applications of constructors to terms, and *expressions* allow applications of function symbols to expressions (Fig. 1). We use metavariables u and w to range over both names and variables. *Substitutions* $\{^M/_x\}$ replace x with M. Arbitrarily large substitutions can be written as $\{^{M_1}/_{x_1}, \dots, ^{M_n}/_{x_n}\}$ and the letters σ and τ range over substitutions. We write $M\sigma$ for the result of applying σ to the variables of M. Similarly, *renamings* $\{^u/_w\}$ replace w with u, where u and w are both names or both variables.

$M, N ::=$	terms
$\quad a, b, c, \dots, k, \dots, m, n, \dots, s$	name
$\quad x, y, z$	variable
$\quad f(M_1, \dots, M_l)$	constructor application
$D ::=$	expressions
$\quad M$	term
$\quad h(D_1, \dots, D_l)$	function evaluation
$P, Q, R ::=$	processes
$\quad 0$	null process
$\quad P \mid Q$	parallel composition
$\quad !P$	replication
$\quad \nu\, a.P$	name restriction
$\quad M(x).P$	message input
$\quad \overline{M}\langle N\rangle.P$	message output
$\quad$ let $x = D$ in P else Q	expression evaluation
$\quad t{::}\, P$	barrier

Figure 1. Syntax for terms and processes

The semantics of a destructor g of arity l are given by a finite set $\mathsf{def}(g)$ of rewrite rules $g(M'_1, \dots, M'_l) \to M'$, where $M'_1, \dots, M'_l, M'$ are terms that contain only constructors and variables, the variables of M' must be bound in $M'_1, \dots, M'_l$, and variables are subject to renaming. The evaluation of expression $g(M_1, \dots, M_l)$ succeeds if there exists a rewrite rule $g(M'_1, \dots, M'_l) \to M'$ in $\mathsf{def}(g)$ and a substitution σ such that $M_i = M'_i\sigma$ for all $i \in \{1, \dots, l\}$, and in this case $g(M_1, \dots, M_l)$ evaluates to $M'\sigma$. In order to avoid distinguishing constructors and destructors in the semantics of expressions, we let $\mathsf{def}(f)$ be $\{f(x_1, \dots, x_l) \to f(x_1, \dots, x_l)\}$, where f is a constructor of arity l. In particular, we use n-ary constructors $(\!|M_1, \dots, M_n|\!)$ for tuples, and unary destructors $\pi_{i,n}$ for projections, with the rewrite rule $\pi_{i,n}((\!|x_1, \dots, x_n|\!)) \to x_i$ for all $i \in \{1, \dots, n\}$. ProVerif supports both rewrite rules and equations [22]; we omit equations in this paper for simplicity. It is straightforward to extend our proofs to equations, and our implementation supports them.

The grammar for *processes* is presented in Fig. 1. The process let $x = D$ in P else Q tries to evaluate D; if this succeeds, then x is bound to the result and P is executed, otherwise, Q is executed. We define the conditional if $M = N$ then P else Q as let $x = \mathsf{eq}(M, N)$ in P else Q, where x is a fresh variable, eq is a binary destructor, and $\mathrm{def}(\mathsf{eq}) = \{\mathsf{eq}(y, y) \to y\}$; we always include eq in our set of function symbols. The else branches may be omitted when Q is the null process. The rest of the syntax is standard (see [8], [22], [45]), except for barriers, which we explain below.

Our syntax allows processes to contain barriers $t{::}P$, where $t \in \mathbb{N}$. Intuitively, $t{::}P$ blocks P until all processes running in parallel are ready to synchronise at barrier t. In addition, barriers are ordered, so $t{::}P$ is also blocked if there are any barriers t' such that $t' < t$. Blanchet, Abadi & Fournet [22, Section 8] also introduced a notion of synchronisation, named *stages*. A stage synchronisation can occur at any point, by dropping processes that did not complete the previous stage. By comparison, a barrier synchronisation cannot drop processes. For example, in the process $\overline{c}\langle k\rangle.1{::}\overline{c}\langle m\rangle \mid 1{::}\overline{c}\langle n\rangle$, the barrier synchronisation cannot occur before the output of k. It follows that the process cannot output n without having previously output k. In contrast, with stage synchronisation, either k is output first, then the process moves to the next stage, then it may output m and n, or the process immediately moves to the next stage by dropping $\overline{c}\langle k\rangle.1{::}\overline{c}\langle m\rangle$, so it may output n without any other output. Our notion of barrier is essential for equivalence properties that require swapping data between two processes, because we must not drop one of these processes.

Given a process P, the multiset $\mathsf{barriers}(P)$ collects all barriers that occur in P. Thus, $\mathsf{barriers}(t{::}Q) = \{t\} \cup \mathsf{barriers}(Q)$ and in all other cases, $\mathsf{barriers}(P)$ is the multiset union of the barriers of the immediate subprocesses of P. We naturally extend the function barriers to multisets $\mathcal{P}$ of processes by $\mathsf{barriers}(\mathcal{P}) = \bigcup_{P\in\mathcal{P}} \mathsf{barriers}(P)$. For each barrier t, the number of processes that must synchronise is equal to the number of elements t in $\mathsf{barriers}(P)$. It follows that the number of barriers which must be reached is defined in advance of execution, and thus branching behaviour may cause blocking. For example, the process $c(x).\text{if } x = k \text{ then } 1{::}\overline{c}\langle m\rangle \text{ else } \overline{c}\langle n\rangle \mid 1{::}\overline{c}\langle s\rangle$ contains two barriers that must synchronise. However, when the term bound to x is not k, the else branch is taken and one of the barriers is dropped, so only one barrier remains. In this case, barrier synchronisation blocks forever, and the process never outputs s. The occurrence of barriers under replication is explicitly forbidden, because with barriers under replication, the number of barriers that we need to synchronise is ill-defined. We partly overcome this limitation in Section III-E1.

The *scope* of names and variables is delimited by binders $\nu\, n$, $M(x)$, and $\text{let } x = D \text{ in}$. The set of free names $\mathsf{fn}(P)$ contains every name n in P which is not under the scope of the binder $\nu\, n$. The set of free variables $\mathsf{fv}(P)$ contains every variable x in P which is not under the scope of a message input $M(x)$ or an expression evaluation $\text{let } x = D \text{ in}$. Using similar notation, the set of names in a term M is denoted $\mathsf{fn}(M)$ and the set of variables in a term M is denoted $\mathsf{fv}(M)$. We naturally extend these functions to multisets $\mathcal{P}$ of processes by $\mathsf{fn}(\mathcal{P}) = \bigcup_{P\in\mathcal{P}} \mathsf{fn}(P)$ and $\mathsf{fv}(\mathcal{P}) = \bigcup_{P\in\mathcal{P}} \mathsf{fv}(P)$. A term M is ground if $\mathsf{fv}(M) = \emptyset$, a substitution $\{^{M}/_{x}\}$ is ground if M is ground, and a process P is closed if $\mathsf{fv}(P) = \emptyset$. Processes are considered equal modulo renaming of bound names and variables. As usual, substitutions avoid name and variable capture, by first renaming bound names and variables to fresh names and variables, respectively.

$M \Downarrow M$ (M is a term, so it does not contain destructors)

$h(D_1, \dots, D_l) \Downarrow N\sigma$ if
there exist $h(N_1, \dots, N_l) \to N \in \mathsf{def}(h)$ and σ such that for all $i \in \{1, \dots, l\}$ we have $D_i \Downarrow M_i$ and $M_i = N_i\sigma$

$B, E, \mathcal{P} \cup \{0\} \to B, E, \mathcal{P}$ (RED NIL)

$B, E, \mathcal{P} \cup \{P \mid Q\} \to B, E, \mathcal{P} \cup \{P, Q\}$ (RED PAR)

$B, E, \mathcal{P} \cup \{!P\} \to B, E, \mathcal{P} \cup \{P, !P\}$ (RED REPL)

$B, E, \mathcal{P} \cup \{\nu\, n.P\} \to B, E \cup \{n'\}, \mathcal{P} \cup \{P\{^{n'}/_{n}\}\}$
for some name n' such that $n' \notin E \cup \mathsf{fn}(\mathcal{P} \cup \{\nu\, n.P\})$ (RED RES)

$B, E, \mathcal{P} \cup \{\overline{N}\langle M\rangle.P, N(x).Q\} \to B, E, \mathcal{P} \cup \{P, Q\{^{M}/_{x}\}\}$ (RED I/O)

$B, E, \mathcal{P} \cup \{\text{let } x = D \text{ in } P \text{ else } Q\} \to B, E, \mathcal{P} \cup \{P\{^{M}/_{x}\}\}$ (RED DESTR 1)
if $D \Downarrow M$

$B, E, \mathcal{P} \cup \{\text{let } x = D \text{ in } P \text{ else } Q\} \to B, E, \mathcal{P} \cup \{Q\}$
if there is no M such that $D \Downarrow M$ (RED DESTR 2)

$B, E, \mathcal{P} \cup \{t{::}P_1, \dots, t{::}P_n\} \to B\backslash\{t^n\}, E, \mathcal{P} \cup \{P_1, \dots, P_n\}$
if $n \geq 1$ and for all t' such that $t' \leq t$, we have $t' \notin B\backslash\{t^n\}$, where t^n denotes n copies of t. (RED BAR)

Figure 2. Operational semantics

The operational semantics is defined by reduction ($\to$) on *configurations*. A configuration $\mathcal{C}$ is a triple $B, E, \mathcal{P}$, where B is a finite multiset of integers, E is a finite set of names, and $\mathcal{P}$ is a finite multiset of closed processes. The multiset B contains the barriers that control the synchronisation of processes in $\mathcal{P}$. The set E is initially empty and is extended to include any names introduced during reduction, namely, those names introduced by (RED RES). When $E = \{\tilde{a}\}$ and $\mathcal{P} = \{P_1, \dots, P_n\}$, the configuration $B, E, \mathcal{P}$ intuitively stands for $\nu\, \tilde{a}.(P_1 \mid \cdots \mid P_n)$. We consider configurations as equal modulo any renaming of the names in $E, \mathcal{P}$ that leaves $\mathsf{fn}(\mathcal{P}) \setminus E$ unchanged. The initial configuration for a closed process P is $\mathcal{C}_{\text{init}}(P) = \mathsf{barriers}(P), \emptyset, \{P\}$. Fig. 2 defines reduction rules for each construct of the language. The rule (RED REPL) creates a new copy of the replicated process P. The rule (RED RES) reduces $\nu\, n$ by creating a fresh name n', adding it to E, and substituting it for n. The rule (RED I/O) performs communication: the term M sent by $\overline{N}\langle M\rangle.P$ is received by $N(x).Q$, and substituted for x. The rules (RED DESTR 1) and (RED DESTR 2) treat expression evaluations. They first evaluate D, using the relation $D \Downarrow M$, which means that the expression D evaluates to the term M, and is also defined in Fig. 2. When this evaluation succeeds, (RED DESTR 1) substitutes the result M for x and runs P. When it fails, (RED DESTR 2) runs Q. Finally, the new rule (RED BAR) performs barrier synchronisation: it synchronises on the lowest barrier t in B. If t occurs n times in B, it requires n processes $t{::}\, P_1, \dots, t{::}\, P_n$ to be ready to synchronise, and in this case, it removes barrier t both from B and from these processes, which can then further reduce. A configuration $B, E, \mathcal{P}$ is *valid* when $\mathsf{barriers}(\mathcal{P}) \subseteq B$. It is easy to check that the initial configuration is valid and that validity is preserved by reduction. We shall only manipulate valid configurations.

Example 4. Let us consider the parallel composition of processes $P \triangleq \overline{c}\langle k\rangle.1{::}\, c(x)$, $Q \triangleq \nu\, n.1{::}\,\overline{c}\langle n\rangle$, and $R \triangleq c(x)$, which yields the initial configuration $\mathcal{C} = \{1^2\},\ \emptyset,\ \{P \mid Q \mid R\}$, since the process $P \mid Q \mid R$ contains two barriers 1. We have

$$\begin{aligned}
\mathcal{C} &= \{1^2\},\ \emptyset,\ \{P \mid Q \mid R\} \\
&\to \{1^2\}, \emptyset, \{P,\ Q \mid R\} && \text{by (RED PAR)}\\
&\to \{1^2\}, \emptyset, \{P,\ Q,\ R\} && \text{by (RED PAR)}\\
&\to \{1^2\}, \emptyset, \{1{::}\, c(x),\ Q,\ 0\} && \text{by (RED I/O)}\\
&\to \{1^2\}, \emptyset, \{1{::}\, c(x),\ Q\} && \text{by (RED NIL)}\\
&\to \{1^2\}, \{n'\}, \{1{::}\, c(x),\ 1{::}\,\overline{c}\langle n'\rangle\} && \text{by (RED RES)}\\
&\to \emptyset, \{n'\}, \{c(x),\ \overline{c}\langle n'\rangle\} && \text{by (RED BAR)}\\
&\to \emptyset, \{n'\}, \{0,\ 0\} && \text{by (RED I/O)}\\
&\to \emptyset, \{n'\}, \{0\} && \text{by (RED NIL)}\\
&\to \emptyset, \{n'\}, \emptyset && \text{by (RED NIL)}
\end{aligned}$$

B. Observational equivalence

Intuitively, configurations $\mathcal{C}$ and $\mathcal{C}'$ are observationally equivalent if they can output on the same channels in the presence of any adversary. Formally, we adapt the definition of observational equivalence by Arapinis *et al.* [46] to consider barriers rather than mutable state. We define a *context* $C[\_]$ to be a process with a hole. We obtain $C[P]$ as the result of filling $C[\_]$'s hole with process P. We define *adversarial contexts* as contexts $\nu\,\tilde{n}.(\_ \mid Q)$ with $\mathsf{fv}(Q) = \emptyset$ and $\mathsf{barriers}(Q) = \emptyset$. When $\mathcal{C} = B, E, \mathcal{P}$ and $C[\_] = \nu\,\tilde{n}.(\_ \mid Q)$ is an adversarial context, we define $C[\mathcal{C}] = B, E \cup \{\tilde{n}\}, \mathcal{P} \cup \{Q\}$, after renaming the names in $E, \mathcal{P}$ so that $E \cap \mathsf{fn}(Q) = \emptyset$. A configuration $\mathcal{C} = B, E, \mathcal{P}$ can output on a channel N, denoted, $\mathcal{C} \downarrow_N$, if there exists $\overline{N}\langle M\rangle.P \in \mathcal{P}$ with $\mathsf{fn}(N) \cap E = \emptyset$, for some term M and process P.

Definition 1 (Observational equivalence). Observational equivalence between configurations $\approx$ is the largest symmetric relation $\mathcal{R}$ between valid configurations such that $\mathcal{C}\ \mathcal{R}\ \mathcal{C}'$ implies:

1) if $\mathcal{C} \downarrow_N$, then $\mathcal{C}' \to^* \downarrow_N$, for all N;
2) if $\mathcal{C} \to \mathcal{C}_1$, then $\mathcal{C}' \to^* \mathcal{C}_1'$ and $\mathcal{C}_1\ \mathcal{R}\ \mathcal{C}_1'$, for some $\mathcal{C}_1'$.
3) $C[\mathcal{C}]\ \mathcal{R}\ C[\mathcal{C}']$ for all adversarial contexts $C[\_]$.

Closed processes P and P' are *observationally equivalent*, denoted $P \approx P'$, if $\mathcal{C}_{\text{init}}(P) \approx \mathcal{C}_{\text{init}}(P')$.

The definition first formulates observational equivalence on semantic configurations. Item 1 guarantees that, if a configuration $\mathcal{C}$ outputs on a public channel, then so does $\mathcal{C}'$. Item 2 guarantees that this property is preserved by reduction, and Item 3 guarantees that it is preserved in the presence of any adversary. Finally, observational equivalence is formulated on closed processes.

C. Biprocesses

The calculus defines syntax to model pairs of processes that have the same structure and differ only by the terms that they contain. We call such a pair of processes a *biprocess*. The grammar for biprocesses is an extension of Fig. 1, with additional cases so that $\mathsf{diff}[M, M']$ is a term and $\mathsf{diff}[D, D']$ is an expression. (We occasionally refer to processes and biprocesses as processes when it is clear from the context.) Given a biprocess P, we define processes $\mathsf{fst}(P)$ and $\mathsf{snd}(P)$ as follows: $\mathsf{fst}(P)$ is obtained by replacing all occurrences of $\mathsf{diff}[M, M']$ with M and $\mathsf{snd}(P)$ is obtained by replacing $\mathsf{diff}[M, M']$ with M'. We define $\mathsf{fst}(D)$, $\mathsf{fst}(M)$, $\mathsf{snd}(D)$, and $\mathsf{snd}(M)$ similarly, and naturally extend these functions to multisets of biprocesses by $\mathsf{fst}(\mathcal{P}) = \{\mathsf{fst}(P) \mid P \in \mathcal{P}\}$ and $\mathsf{snd}(\mathcal{P}) = \{\mathsf{snd}(P) \mid P \in \mathcal{P}\}$, and to configurations by $\mathsf{fst}(B, E, \mathcal{P}) = B, E, \mathsf{fst}(\mathcal{P})$ and $\mathsf{snd}(B, E, \mathcal{P}) = B, E, \mathsf{snd}(\mathcal{P})$. The standard definitions of barriers, free names, and free variables apply to biprocesses as well. Observational equivalence can be formalised as a property of biprocesses:

Definition 2. A closed biprocess P satisfies *observational equivalence* if $\mathsf{fst}(P) \approx \mathsf{snd}(P)$.

The semantics for biprocesses includes the rules in Fig. 2, except for (RED I/O), (RED DESTR 1), and (RED DESTR 2) which are revised in Fig. 3. It follows from this semantics that, if $\mathcal{C} \to \mathcal{C}'$, then $\mathsf{fst}(\mathcal{C}) \to \mathsf{fst}(\mathcal{C}')$ and $\mathsf{snd}(\mathcal{C}) \to \mathsf{snd}(\mathcal{C}')$. In other words, a biprocess reduces when the two underlying processes reduce in the same way. However, reductions in $\mathsf{fst}(\mathcal{C})$ or $\mathsf{snd}(\mathcal{C})$ do not necessarily imply reductions in $\mathcal{C}$, that is, there exist configurations $\mathcal{C}$ such that $\mathsf{fst}(\mathcal{C}) \to \mathsf{fst}(\mathcal{C}')$, but there is no such reduction $\mathcal{C} \to \mathcal{C}'$, and symmetrically for $\mathsf{snd}(\mathcal{C})$. For example, given the configuration $\mathcal{C} = \emptyset, \emptyset, \{\overline{\mathsf{diff}[a, c]}\langle n\rangle.0, a(x).0\}$, we have $\mathsf{fst}(\mathcal{C}) \to \emptyset, \emptyset, \{0, 0\}$, but there is no reduction $\mathcal{C} \to$

$$B, E, \mathcal{P} \cup \{\overline{N}\langle M\rangle.P, N'(x).Q\} \;\rightarrow$$
$$B, E, \mathcal{P} \cup \{P, Q\{^{M}/_{x}\}\} \qquad \text{(RED I/O)}$$
if $\mathsf{fst}(N) = \mathsf{fst}(N')$ and $\mathsf{snd}(N) = \mathsf{snd}(N')$

$$B, E, \mathcal{P} \cup \{\text{let } x = D \text{ in } P \text{ else } Q\} \;\rightarrow$$
$$B, E, \mathcal{P} \cup \{P\{^{\mathsf{diff}[M, M']}/_{x}\}\} \quad \text{(RED DESTR 1)}$$
if $\mathsf{fst}(D) \Downarrow M$ and $\mathsf{snd}(D) \Downarrow M'$

$$B, E, \mathcal{P} \cup \{\text{let } x = D \text{ in } P \text{ else } Q\} \;\rightarrow\; B, E, \mathcal{P} \cup \{Q\}$$
if there is no M such that $\mathsf{fst}(D) \Downarrow M$
and no M' such that $\mathsf{snd}(D) \Downarrow M'$
(RED DESTR 2)

Figure 3. Generalised semantics for biprocesses

$$B, E, \mathcal{P} \cup \{\overline{N}\langle M\rangle.P, N'(x).Q\} \;\uparrow$$
if $(\mathsf{fst}(N) = \mathsf{fst}(N')) \not\Leftrightarrow (\mathsf{snd}(N) = \mathsf{snd}(N'))$ (DIV I/O)

$$B, E, \mathcal{P} \cup \{\text{let } x = D \text{ in } P \text{ else } Q\} \;\uparrow$$
if $(\exists M.\mathsf{fst}(D) \Downarrow M) \not\Leftrightarrow (\exists M'.\mathsf{snd}(D) \Downarrow M')$
(DIV DESTR)

Figure 4. Semantics for divergence

$\emptyset, \emptyset, \{0, 0\}$. Formally, this behaviour can be captured using the *divergence* relation ($\uparrow$) for configurations (Fig. 4) [25]. Divergence can occur because either: i) one process can perform a communication and the other cannot, by rule (DIV I/O); or ii) the evaluation of an expression succeeds in one process and fails in the other, by rule (DIV DESTR). Using the notion of *diff-equivalence* (Definition 3), Theorem 1 shows that a biprocess P satisfies observational equivalence when reductions in $C[\mathcal{C}_{\mathsf{init}}(\mathsf{fst}(P))]$ or $C[\mathcal{C}_{\mathsf{init}}(\mathsf{snd}(P))]$ imply reductions in $C[\mathcal{C}_{\mathsf{init}}(P)]$ for all adversarial contexts $C[\_]$, that is, configurations obtained from $C[\mathcal{C}_{\mathsf{init}}(P)]$ never diverge.

Definition 3 (Diff-equivalence). A closed biprocess P satisfies *diff-equivalence* if for all adversarial contexts $C[\_]$, there is no configuration $\mathcal{C}$ such that $C[\mathcal{C}_{\mathsf{init}}(P)] \rightarrow^* \mathcal{C}$ and $\mathcal{C} \uparrow$.

Theorem 1. Let P be a closed biprocess without barriers. If P satisfies diff-equivalence, then P satisfies observational equivalence.

Theorem 1 can be proved by adapting the proof of Blanchet, Abadi & Fournet [22, Theorem 1], which presents a similar result using a semantics based on structural equivalence and reduction instead of reduction on configurations.

III. AUTOMATED REASONING

To prove equivalence, we define a compiler from a biprocess (containing barriers) to a set of biprocesses without barriers. The biprocesses in that set permit various swapping strategies. We show that if one of these biprocesses satisfies diff-equivalence, then the original biprocess satisfies observational equivalence. The compiler works in two steps:

1) Function annotate annotates barriers with the data to be swapped and channels for sending and receiving such data.
2) Function elim-and-swap translates the biprocess with annotated barriers into biprocesses without barriers, which encode barriers using communication (inputs and outputs). We exploit this communication to allow swapping, by sending back data to a different barrier.

We introduce annotated barriers (Section III-A) and define these two steps (Sections III-B and III-C) below. By combining these two steps we obtain our compiler (Section III-D), which we have implemented in ProVerif 1.94 (http://proverif.inria.fr/). The proof of soundness shows that these two steps preserve the observational behaviour of the biprocesses, so that if a compiled biprocess satisfies observational equivalence, then so does the initial biprocess.

A. Process calculus with annotated barriers

We introduce an *annotated barrier* construct $t[a, c, \varsigma]{::}\, P$, which is not present in the syntax introduced in Section II, but is used by our compiler. In this construct, a and c are distinct channel names: channel a will be used for sending swappable data, and channel c for receiving swapped data. Moreover, the *ordered substitution* $\varsigma = (^{M_1}/_{x_1}, \ldots, ^{M_n}/_{x_n})$ collects swappable data $M_1, \ldots, M_n$ and associates these terms with variables $x_1, \ldots, x_n$; the process P uses these variables instead of the terms $M_1, \ldots, M_n$. The ordered substitution ς is similar to a substitution, except that the elements $^{M_1}/_{x_1}, \ldots, ^{M_n}/_{x_n}$ are ordered. (We indicate ordering using parentheses instead of braces.) The ordering is used to designate each variable in the domain unambiguously. We define $\mathsf{dom}(\varsigma) = \{x_1, \ldots, x_n\}$ and $\mathsf{range}(\varsigma) = \{M_1, \ldots, M_n\}$. The annotated barrier $t[a, c, \varsigma]{::}\, P$ binds the variables in the domain of ς in P, so we extend the functions fn and fv to annotated barriers as follows:

$$\mathsf{fn}(t[a, c, \varsigma]{::}\, P) = \{a, c\} \cup \mathsf{fn}(\mathsf{range}(\varsigma)) \cup \mathsf{fn}(P)$$
$$\mathsf{fv}(t[a, c, \varsigma]{::}\, P) = \mathsf{fv}(\mathsf{range}(\varsigma)) \cup (\mathsf{fv}(P) \setminus \mathsf{dom}(\varsigma))$$

We define the *ordered domain* of ς, $\mathsf{ordom}(\varsigma) = (x_1, \ldots, x_n)$, as the tuple containing the variables in the domain of ς, in the same order as in the definition of ς.

We also introduce a *domain-barrier* construct $t[a, c, \tilde{x}]{::}\, P$, which is similar to an annotated barrier except that the ordered substitution ς is replaced with a tuple of variables $\tilde{x} = (x_1, \ldots, x_n)$ corresponding to the ordered domain of ς. Domain-barriers occur in barriers(P), but not in processes. We extend function barriers to annotated barriers as follows:

$$\mathsf{barriers}(t[a, c, \varsigma]{::}\, P) = \{t[a, c, \mathsf{ordom}(\varsigma)]{::}\, P\} \cup \mathsf{barriers}(P)$$

Hence, function barriers maps processes to multisets of domain-barriers and integers, and domain-barriers include the process that follows the barrier itself. In addition, we extend fst and snd for configurations as follows: $\mathsf{fst}(t[a,c,\tilde{x}]\!::P) = t[a,c,\tilde{x}]\!::\mathsf{fst}(P)$ and $\mathsf{fst}(B,E,\mathcal{P}) = \mathsf{fst}(B),E,\mathsf{fst}(\mathcal{P})$, and similarly for snd.

The operational semantics for processes with both standard and annotated barriers extends the semantics for processes with only standard barriers, with the following rule:

$$\begin{aligned}&B,E,\mathcal{P}\cup\{t\!::P_1,\dots,t\!::P_m,t[a_{m+1},c_{m+1},\varsigma_{m+1}]\!::P_{m+1},\\&\qquad\dots,t[a_n,c_n,\varsigma_n]\!::P_n\}\\&\to B',E,\mathcal{P}\cup\{P_1,\dots,P_m,P_{m+1}\varsigma_{m+1},\dots,P_n\varsigma_n\}\end{aligned} \quad \text{(RED BAR')}$$

where $0 \le m \le n$, $1 \le n$, $B = \{t^m, t[a_{m+1},c_{m+1},\mathsf{ordom}(\varsigma_{m+1})]\!::P_{m+1},\dots,t[a_n,c_n,\mathsf{ordom}(\varsigma_n)]\!::P_n\}\cup B'$, and for all t' such that $t' \le t$, t' does not appear in B', i.e., $t' \notin B'$ and $t'[\_]\!::\_ \notin B'$. When all barriers are standard, this rule reduces to (RED BAR).

We introduce the function $\mathsf{channels}(B) = \{a \mid t[a,c,\tilde{x}]\!::P \in B\} \cup \{c \mid t[a,c,\tilde{x}]\!::P \in B\}$ to recover the multiset of names used by the domain-barriers in B. We also define the function fn-nobc, which returns the free names excluding the channels of barriers, by $\mathsf{fn\text{-}nobc}(t[a,c,\varsigma]\!::P) = \mathsf{fn}(\mathsf{range}(\varsigma)) \cup \mathsf{fn\text{-}nobc}(P)$ and, for all other processes, $\mathsf{fn\text{-}nobc}(P)$ is defined inductively like $\mathsf{fn}(P)$. (The acronym "nobc" stands for "no barrier channels".) The initial configuration for a closed process P with annotated barriers is $\mathcal{C}_{\mathsf{init}}(P) = \mathsf{barriers}(P), \mathsf{channels}(\mathsf{barriers}(P)), \{P\}$.

We introduce the following validity condition to ensure that channels of annotated barriers are not mixed with other names: they are fresh names when they are introduced by barrier annotation (Section III-B); they should remain pairwise distinct and distinct from other names. Their scope is global, but they are private, that is, the adversary does not have access to them.

Definition 4 (Validity). A process P is *valid* if it is closed, the elements of $\mathsf{channels}(\mathsf{barriers}(P))$ are pairwise distinct, $\mathsf{channels}(\mathsf{barriers}(P)) \cap \mathsf{fn\text{-}nobc}(P) = \emptyset$, and for all annotated barriers in P such that $P = C[t[a,c,\varsigma]\!::Q]$, we have $\mathsf{fv}(Q) \subseteq \mathsf{dom}(\varsigma)$ and $C[\_]$ does not bind a, c, nor the names in $\mathsf{fn}(Q)$ above the hole.

A configuration $B,E,\mathcal{P}$ is *valid* if $\mathsf{barriers}(\mathcal{P}) \subseteq B$, $\mathsf{channels}(B) \subseteq E$, all processes in $\mathcal{P}$ are valid, the elements of $\mathsf{channels}(B)$ are pairwise distinct, and $\mathsf{channels}(B) \cap \mathsf{fn\text{-}nobc}(\mathcal{P}) = \emptyset$.

Validity guarantees that channels used in annotated barriers are pairwise distinct (the elements of $\mathsf{channels}(\mathsf{barriers}(P))$ are pairwise distinct; the elements of $\mathsf{channels}(B)$ are pairwise distinct), distinct from other names ($\mathsf{channels}(\mathsf{barriers}(P)) \cap \mathsf{fn\text{-}nobc}(P) = \emptyset$; $\mathsf{channels}(B) \cap \mathsf{fn\text{-}nobc}(\mathcal{P}) = \emptyset$), and free in the processes (for all annotated barriers in P such that $P = C[t[a,c,\varsigma]\!::Q]$, $C[\_]$ does not bind a nor c above the hole). These channels must be in E ($\mathsf{channels}(B) \subseteq E$), which corresponds to the intuition that they are global but private. Furthermore, for each annotated barrier $t[a,c,\varsigma]\!::Q$, we require that $\mathsf{fv}(Q) \subseteq \mathsf{dom}(\varsigma)$ and the names in $\mathsf{fn}(Q)$ are not bound above the barrier, that is, they are global. This requirement ensures that the local state of the process $t[a,c,\varsigma]\!::Q$ is contained in the ordered substitution ς. The process Q refers to this state using variables in $\mathsf{dom}(\varsigma)$.

The next lemma allows us to show that all considered configurations are valid.

Lemma 2. If P is a valid process, then $\mathcal{C}_{\mathsf{init}}(P)$ is valid. Validity is preserved by reduction, by application of an adversarial context, and by application of fst and snd.

The proof of Lemma 2 and all other proofs are detailed in the long version of this paper [37].

We refer to processes in which all barriers are annotated as *annotated processes*, and processes in which all barriers are standard as *standard processes*.

B. Barrier annotation

Next, we define the first step of our compiler, which annotates barriers with additional information.

Definition 5. We define function annotate, from standard processes to annotated processes, as follows: annotate transforms $C[t\!::Q]$ into $C[t[a,c,\varsigma]\!::Q']$, where $C[\_]$ is any context without replication above the hole, a and c are distinct fresh names, and $(Q',\varsigma) = \mathsf{split}(Q)$, where the function split is defined below. The transformations are performed until all barriers are annotated, in a top-down order, so that in the transformation above, all barriers above $t\!::Q$ are already annotated and barriers inside Q are standard.

The function split is defined by $\mathsf{split}(Q) = (Q',\varsigma)$ where Q' is a process and $\varsigma = ({}^{M_1}/_{x_1},\dots,{}^{M_n}/_{x_n})$ is an ordered substitution such that terms $M_1,\dots,M_n$ are the largest subterms of Q that do not contain names or variables previously bound in Q, variables $x_1,\dots,x_n$ are fresh, and process Q' is obtained from Q by replacing each M_i with x_i, so that $Q = Q'\varsigma$. Moreover, the variables $x_1,\dots,x_n$ occur in this order in Q' when read from left to right.

Intuitively, the function split separates a process Q into its "skeleton" Q' (a process with variables as placeholders for data) and associated data in the ordered substitution ς. Such data can be swapped with another process that has the same skeleton. The ordering of $x_1,\dots,x_n$ chosen in the definition of split guarantees that the ordering of variables in the domain of ς is consistent among the various subprocesses. This ordering of variables and the fact that $M_1,\dots,M_n$ are the largest possible subterms allows the checks in the definition of our compiler (see definition of function swapper in Section III-C) to succeed more often, and hence increases opportunities for swapping.

Example 5. We have

$$\mathsf{split}(\overline{c}\langle\mathsf{diff}[v,v']\rangle) = (\overline{x}\langle y\rangle, ({}^{c}/_{x}, {}^{\mathsf{diff}[v,v']}/_{y}))$$
$$\mathsf{split}(\overline{c}\langle\mathsf{diff}[v',v]\rangle) = (\overline{x'}\langle y'\rangle, ({}^{c}/_{x'}, {}^{\mathsf{diff}[v',v]}/_{y'}))$$

The process $\overline{c}\langle\mathsf{diff}[v,v']\rangle$ is separated into its skeleton $Q' = \overline{x}\langle y\rangle$ and the ordered substitution $\varsigma = ({}^{c}/_{x}, {}^{\mathsf{diff}[v,v']}/_{y})$, which defines the values of the variables x and y such that $\overline{c}\langle\mathsf{diff}[v,v']\rangle = Q'\varsigma$. The process $\overline{c}\langle\mathsf{diff}[v',v]\rangle$ is separated similarly.

Using these results, $\mathsf{annotate}(P_{\mathsf{ex}})$ is defined as

$$\overline{c}\langle A\rangle.1[a,b,({}^{c}/_{x}, {}^{\mathsf{diff}[v,v']}/_{y})]{::}\,\overline{x}\langle y\rangle \mid$$
$$\overline{c}\langle B\rangle.1[a',b',({}^{c}/_{x'}, {}^{\mathsf{diff}[v',v]}/_{y'})]{::}\,\overline{x'}\langle y'\rangle$$

where a, a', b, b' are fresh names. That is, $\mathsf{annotate}(P_{\mathsf{ex}})$ is derived by annotating the two barriers in P_{ex}. (Process P_{ex} is given in Example 3.)

For soundness of the transformation (Proposition 4), it is sufficient that:

Lemma 3. If $(Q', \varsigma) = \mathsf{split}(Q)$, then $Q = Q'\varsigma$, $\mathsf{fv}(Q') = \mathsf{dom}(\varsigma)$, and $\mathsf{fn}(Q') = \emptyset$.

Intuitively, when reducing the annotated barrier by (RED BAR'), we reduce $t[a,c,\varsigma]{::}\,Q'$ to $Q'\varsigma$, which is equal to Q by Lemma 3, so we recover the process Q we had before annotation. The conditions that $\mathsf{fv}(Q') = \mathsf{dom}(\varsigma)$ and $\mathsf{fn}(Q') = \emptyset$ show that no names and variables are free in Q' and bound above the barrier, thus substitution ς contains the whole state of the process $Q = Q'\varsigma$.

The following proposition shows that annotation does not alter the semantics of processes:

Proposition 4. If P_0 is a closed standard biprocess and $P'_0 = \mathsf{annotate}(P_0)$, then P'_0 is valid, $\mathsf{fst}(P'_0) \approx \mathsf{fst}(P_0)$, and $\mathsf{snd}(P'_0) \approx \mathsf{snd}(P_0)$.

Proof sketch: The main step of the proof consists in showing that, when $C[t{::}\,P\varsigma]$ and $C[t[a,c,\varsigma]{::}\,P]$ are valid processes, we have

$$C[t{::}\,P\varsigma] \approx C[t[a,c,\varsigma]{::}\,P] \tag{2}$$

This proof is performed by defining a relation $\mathcal{R}$ that satisfies the conditions of Definition 1. By Lemma 3, from the annotated biprocess P'_0, we can rebuild the initial process P_0 by replacing each occurrence of an annotated barrier $t[a,c,\varsigma]{::}\,Q$ with $Q\varsigma$, so the same replacement also transform $\mathsf{fst}(P'_0)$ into $\mathsf{fst}(P_0)$ and $\mathsf{snd}(P'_0)$ into $\mathsf{snd}(P_0)$. By (2), this replacement preserves the observational behaviour of the processes. ■

C. Barrier elimination and swapping

Next, we define the second step of our compiler, which translates an annotated biprocess into biprocesses without barriers. Each annotated barrier $t[a,c,\varsigma]$ is eliminated by replacing it with an output on channel a of swappable data, followed by an input on channel c that receives swapped data. A swapping process is added in parallel, which receives the swappable data on channels a for all barriers t, before sending swapped data on channels c. Therefore, all inputs on channels a must be received before the outputs on channels c are sent and the processes that follow the barriers can proceed, thus the synchronisation between the barriers is guaranteed. Moreover, the swapping process may permute data, sending on channel c data that comes from channel a' with $a' \neq a$, thus implementing swapping. This swapping is allowed only when the processes that follow the barriers are identical (up to renaming of some channel names and variables), so that swapping preserves the observational behaviour of the processes. We detail this construction below.

1) Barrier elimination: First, we eliminate barriers.

Definition 6. The function $\mathsf{bar\text{-}elim}$ removes annotated barriers, by transforming each annotated barrier $t[a, c, ({}^{M_1}/_{z_1}, \dots, {}^{M_n}/_{z_n})]{::}\,Q$ into $\overline{a}\langle(\!|M_1,\dots,M_n|\!)\rangle.c(z).$ let $z_1 = \pi_{1,n}(z)$ in $\cdots$ let $z_n = \pi_{n,n}(z)$ in Q, where z is a fresh variable.

The definition of function $\mathsf{bar\text{-}elim}$ ensures that, if the message $(\!|M_1,\dots,M_n|\!)$ on the private channel a is simply forwarded to the private channel c, then the process derived by application of $\mathsf{bar\text{-}elim}$ binds z_i to M_i for each $i \in \{1,\dots,n\}$, like the annotated barrier, so the original process and the process derived by application $\mathsf{bar\text{-}elim}$ are observationally equivalent. Intuitively, the private channel communication provides an opportunity to swap data.

Example 6. Using the results of Example 5, eliminating barriers from $\mathsf{annotate}(P_{\mathsf{ex}})$ results in $\mathsf{bar\text{-}elim}(\mathsf{annotate}(P_{\mathsf{ex}})) = P_{\mathsf{comp}} \mid P'_{\mathsf{comp}}$, where

$$P_{\mathsf{comp}} \triangleq \overline{c}\langle A\rangle.\overline{a}\langle(\!|c,\mathsf{diff}[v,v']|\!)\rangle.b(z).$$
$$\text{let } x = \pi_{1,2}(z) \text{ in let } y = \pi_{2,2}(z) \text{ in } \overline{x}\langle y\rangle$$
$$P'_{\mathsf{comp}} \triangleq \overline{c}\langle B\rangle.\overline{a'}\langle(\!|c,\mathsf{diff}[v',v]|\!)\rangle.b'(z').$$
$$\text{let } x' = \pi_{1,2}(z') \text{ in let } y' = \pi_{2,2}(z') \text{ in } \overline{x'}\langle y'\rangle$$

for some fresh variables z and z'.

2) Swapping: Next, we define swapping strategies.

Definition 7. The function $\mathsf{swapper}$ is defined as follows:

$$\begin{aligned}
&\mathsf{swapper}(\emptyset) = \{0\}\\
&\mathsf{swapper}(B) =\\
&\quad \big\{a_1(x_1).\cdots.a_n(x_n).\\
&\quad\ \ \overline{c_1}\langle\mathsf{diff}[x_1, x_{f(1)}]\rangle.\cdots.\overline{c_n}\langle\mathsf{diff}[x_n, x_{f(n)}]\rangle.R\\
&\quad\ \ \big| B = \{t[a_1,c_1,\tilde{z}_1]{::}\,Q_1,\dots,t[a_n,c_n,\tilde{z}_n]{::}\,Q_n\} \cup B'\\
&\quad\ \ \text{where, for all } t'[a,c,\tilde{z}]{::}\,Q \in B', \text{ we have } t' > t;\\
&\quad\ \ f \text{ is a permutation of } \{1,\dots,n\} \text{ such that,}\\
&\quad\ \ \text{for all } 1 \le l \le n, \text{ we have } Q_l/\tilde{z}_l =_{\mathsf{ch}} Q_{f(l)}/\tilde{z}_{f(l)};\\
&\quad\ \ R \in \mathsf{swapper}(B');\\
&\quad\ \ \text{and } x_1,\dots,x_n \text{ are fresh variables}\big\}\\
&\quad \text{if } B \neq \emptyset
\end{aligned}$$

where $=_{\mathsf{ch}}$ is defined as follows:

- $Q =_{\mathsf{ch}} Q'$ means that Q equals Q' modulo renaming of channels of annotated barriers and
- $Q/\tilde{z} =_{\mathsf{ch}} Q'/\tilde{z}'$ means that $\tilde{z} = (z_1, \dots, z_k)$ and $\tilde{z}' = (z'_1, \dots, z'_k)$ for some integer k, and $Q\{^{y_1}/_{z_1}, \dots, ^{y_k}/_{z_k}\} =_{\mathsf{ch}} Q'\{^{y_1}/_{z'_1}, \dots, ^{y_k}/_{z'_k}\}$ for some fresh variables $y_1, \dots, y_k$.

The function swapper builds a set of processes from a multiset of domain-barriers B as follows. We identify integer $t \in \mathbb{N}$ and domain-barriers $t[a_1, c_1, \tilde{z}_1] :: Q_1, \dots, t[a_n, c_n, \tilde{z}_n] :: Q_n$ in B such that no other barriers with $t' \leq t$ appear in B, so that these barriers are reduced before other barriers in B. Among these barriers, we consider barriers $t[a_i, c_i, \tilde{z}_i] :: Q_i$ and $t[a_j, c_j, \tilde{z}_j] :: Q_j$ such that $Q_i/\tilde{z}_i =_{\mathsf{ch}} Q_j/\tilde{z}_j$, that is, the processes Q_i and Q_j are equal modulo renaming of channels of annotated barriers, after renaming the variables in $\tilde{z}_i$ and $\tilde{z}_j$ to the same variables, and we allow swapping data between such barriers using the permutation f. We then construct a set of processes which enable swapping, by receiving data to be swapped on channels $a_1, \dots, a_n$, and sending it back on channels $c_1, \dots, c_n$, in the same order in the first component of diff and permuted by f in the second component of diff. The function swapper does not specify an ordering on the pairs of channels $(a_1, c_1), \dots, (a_n, c_n)$, since any ordering is correct.

Example 7. We have $\mathsf{barriers}(\mathsf{annotate}(P_{\mathsf{ex}})) = \{1[a, b, (x, y)] :: \overline{x}\langle y\rangle, 1[a', b', (x', y')] :: \overline{x'}\langle y'\rangle\}$. Moreover, we trivially have $\overline{x}\langle y\rangle/(x, y) =_{\mathsf{ch}} \overline{x}\langle y\rangle/(x, y)$ and $\overline{x'}\langle y'\rangle/(x', y') =_{\mathsf{ch}} \overline{x'}\langle y'\rangle/(x', y')$, because $Q/\tilde{z} =_{\mathsf{ch}} Q/\tilde{z}$ for all Q and $\tilde{z}$. We also have $\overline{x}\langle y\rangle/(x, y) =_{\mathsf{ch}} \overline{x'}\langle y'\rangle/(x', y')$, because

$$\overline{x}\langle y\rangle\{^{x''}/_{x}, ^{y''}/_{y}\} = \overline{x''}\langle y''\rangle = \overline{x'}\langle y'\rangle\{^{x''}/_{x'}, ^{y''}/_{y'}\}$$

It follows that $\mathsf{swapper}(\mathsf{barriers}(\mathsf{annotate}(P_{\mathsf{ex}}))) = \{P_{\mathsf{same}}, P_{\mathsf{swap}}\}$, where

$$P_{\mathsf{same}} \triangleq a(z).a'(z').\overline{b}\langle \mathsf{diff}[z, z]\rangle.\overline{b'}\langle \mathsf{diff}[z', z']\rangle$$
$$P_{\mathsf{swap}} \triangleq a(z).a'(z').\overline{b}\langle \mathsf{diff}[z, z']\rangle.\overline{b'}\langle \mathsf{diff}[z', z]\rangle$$

for some fresh variables z and z'. (Note that $\mathsf{diff}[z, z]$ could be simplified into z.) This set considers the two possible swapping strategies: the strategy that does not swap any data and the strategy that swaps data between the two processes at the barrier.

3) Combining barrier elimination and swapping: Finally, we derive a set of processes by parallel composition of the process output by bar-elim and the processes output by swapper, under the scope of name restrictions on the fresh channels introduced by annotate.

$$\mathsf{elim\text{-}and\text{-}swap}(P) = \left\{ \begin{array}{l} \nu\, \tilde{a}.(\mathsf{bar\text{-}elim}(P) \mid R) \text{ where } B = \mathsf{barriers}(P), \\ \{\tilde{a}\} = \mathsf{channels}(B), \text{ and } R \in \mathsf{swapper}(B) \end{array} \right\}$$

Intuitively, function elim-and-swap encodes barrier synchronisation and swapping using private channel communication, thereby preserving the observational behaviour of processes.

Example 8. Using the results of Examples 6 & 7, applying elim-and-swap to the process $\mathsf{annotate}(P_{\mathsf{ex}})$ generates two processes

$$P_1 \triangleq \nu\, a, a', b, b'.(P_{\mathsf{comp}} \mid P'_{\mathsf{comp}} \mid P_{\mathsf{same}})$$
$$P_2 \triangleq \nu\, a, a', b, b'.(P_{\mathsf{comp}} \mid P'_{\mathsf{comp}} \mid P_{\mathsf{swap}})$$

In the process P_1, no data is swapped, so it behaves exactly like P_{ex}: $(\!| c, \mathsf{diff}[v, v'] |\!)$ is sent on a, sent back on b by P_{same} as $\mathsf{diff}[(\!| c, \mathsf{diff}[v, v'] |\!), (\!| c, \mathsf{diff}[v, v'] |\!)]$ which simplifies into $(\!| c, \mathsf{diff}[v, v'] |\!)$, and after evaluating the projections, P_{comp} reduces into $\overline{c}\langle \mathsf{diff}[v, v']\rangle$, which is the output present in the process P_{ex}. Similarly, P'_{comp} reduces into $\overline{c}\langle \mathsf{diff}[v', v]\rangle$, present in P_{ex}.

By contrast, in process P_2, data is swapped: $(\!| c, \mathsf{diff}[v, v'] |\!)$ is sent on a and $(\!| c, \mathsf{diff}[v', v] |\!)$ is sent on a', and P_{swap} sends back $\mathsf{diff}[(\!| c, \mathsf{diff}[v, v'] |\!), (\!| c, \mathsf{diff}[v', v] |\!)]$ on b. The first component of this term is $(\!| c, v |\!)$ (obtained by taking the first component of each diff), and similarly its second component is also $(\!| c, v |\!)$, so this term simplifies into $(\!| c, v |\!)$. After evaluating the projections, P_{comp} reduces into $\overline{c}\langle v\rangle$. Similarly, P'_{comp} reduces into $\overline{c}\langle v'\rangle$. Hence P_2 behaves like $\overline{c}\langle A\rangle.1 :: \overline{c}\langle v\rangle \mid \overline{c}\langle B\rangle.1 :: \overline{c}\langle v'\rangle$. In particular, P_2 outputs A and B before barrier synchronisation and v and v' after synchronisation just like P_{ex}. But P_2 satisfies diff-equivalence while P_{ex} does not.

The next proposition formalises this preservation of observable behaviour.

Proposition 5. Let P be a valid, annotated biprocess. If $P' \in \mathsf{elim\text{-}and\text{-}swap}(P)$, then $\mathsf{fst}(P) \approx \mathsf{fst}(P')$ and $\mathsf{snd}(P) \approx \mathsf{snd}(P')$.

Proof sketch: This proof is performed by defining a relation $\mathcal{R}$ that satisfies the conditions of Definition 1. The proof is fairly long and delicate, and relies on preliminary lemmas that show that barrier elimination commutes with renaming and substitution, and that it preserves reduction when barriers are not reduced. ∎

D. Our compiler

We combine the annotation (Section III-B) and removal of barrier (Section III-C) steps to define our compiler as

$$\mathsf{compiler}(P) = \mathsf{elim\text{-}and\text{-}swap}(\mathsf{annotate}(P))$$

We have implemented the compiler in ProVerif, which is available from: http://proverif.inria.fr/.

By combining Propositions 4 and 5, we immediately obtain:

Proposition 6. Let P be a closed standard biprocess. If $P' \in \mathsf{compiler}(P)$, then $\mathsf{fst}(P) \approx \mathsf{fst}(P')$ and $\mathsf{snd}(P) \approx \mathsf{snd}(P')$.

This proposition shows that compilation preserves the observational behaviour of processes. The following theorem is an immediate consequence of this proposition:

Theorem 7. Let P be a closed biprocess. If a biprocess in $\mathsf{compiler}(P)$ satisfies observational equivalence, then P satisfies observational equivalence.

This theorem allows us to prove observational equivalence using swapping: we prove that a biprocess in $\mathsf{compiler}(P)$ satisfies observational equivalence using ProVerif (by Theorem 1), and conclude that P satisfies observational equivalence as well. For instance, ProVerif can show that the process $P_2 \in \mathsf{compiler}(P_{\mathsf{ex}})$ of Example 8 satisfies observational equivalence, thus P_{ex} satisfies observational equivalence too.

Our compiler could be implemented in other tools that prove diff-equivalence (e.g., Maude-NPA [23] and Tamarin [24]), by adapting the input language. It could also be applied to other methods of proving equivalence. However, it may be less useful in these cases, since it might not permit the proof of more equivalences in such cases.

E. Extensions

1) Replicated barriers: While our calculus does not allow barriers under replication, we can still prove equivalence with barriers under bounded replication, for any bound. We define bounded replication by $!^n P \triangleq P \mid \cdots \mid P$ with n copies of the process P. We have the following results:

Proposition 8. Let $C[!Q]$ be a closed standard biprocess, such that the context $C[\_]$ does not contain any barrier above the hole. If a biprocess in $\mathsf{compiler}(C[!Q])$ satisfies diff-equivalence, then for all n, a biprocess in $\mathsf{compiler}(C[!^n Q])$ satisfies diff-equivalence.

Proposition 8 shows that, if our approach proves equivalence with unbounded replication, then it also proves equivalence with bounded replication.

Proposition 9. Let $C[Q]$ be a closed standard biprocess, such that the context $C[\_]$ does not contain any replication above the hole. If a biprocess in $\mathsf{compiler}(C[Q])$ satisfies diff-equivalence, then a biprocess in $\mathsf{compiler}(C[t{::}\,Q])$ satisfies diff-equivalence.

Proposition 9 shows that, if our approach proves equivalence after removing a barrier, then it also proves equivalence with the barrier. By combining these two results, we obtain:

Corollary 10. Let Q_{nobar} be obtained from Q by removing all barriers. Let $C[\_]$ be a context that does not contain any replication or barrier above the hole. If a biprocess in $\mathsf{compiler}(C[!Q_{\text{nobar}}])$ satisfies diff-equivalence, then for all n, process $C[!^n Q]$ satisfies observational equivalence.

Hence, we can apply our compiler to prove observational equivalence for biprocesses with bounded replication, for any value of the bound. In the case of election schemes, this result allows us to prove privacy for an unbounded number of voters, for instance in the protocol by Lee *et al.* (Section IV-B).

2) Local synchronisation: Our results could be extended to systems in which several groups of participants synchronise locally inside each group, but do not synchronise with other groups. In this case, we would need several swapping processes similar to those generated by swapper, one for each group.

3) Trace properties: ProVerif also supports the proof of trace properties (reachability and correspondence properties of the form "if some event has been executed, then some other events must have been executed", which serve for formalising authentication) [47]. Our implementation extends this support to processes with barriers, by compiling them to processes without barriers, and applying ProVerif to the compiled processes. In this case, swapping does not help, so our compiler does not swap. We do not detail the proof of trace properties with barriers further, since it is easier and less important than observational equivalence.

IV. Privacy in elections

Elections enable voters to choose representatives. Choices should be made freely, and this has led to the emergence of ballot secrecy as a *de facto* standard privacy requirement of elections. Stronger formulations of privacy, such as receipt-freeness, are also possible.

- Ballot secrecy: a voter's vote is not revealed to anyone.
- Receipt-freeness: a voter cannot prove how she voted.

We demonstrate the suitability of our approach for analysing privacy requirements of election schemes by Fujioka, Okamoto & Ohta, commonly referred to as FOO, and Lee *et al.*, along with some of its variants. Our ProVerif scripts are included in ProVerif's documentation package (http://proverif.inria.fr/). The runtime of these scripts (including compilation of barriers and proof of diff-equivalence by ProVerif) ranges from 0.14 seconds for FOO to 90 seconds for the most complex variant of the Lee *et al.* protocol, on an Intel Xeon 3.6 GHz under Linux.

A. Case study: FOO

1) Cryptographic primitives: FOO uses commitments and blind signatures. We model commitment with a binary constructor commit, and the corresponding destructor open for opening the commitment, with the following rewrite rule:

$$\mathsf{open}(x_{\mathsf{k}}, \mathsf{commit}(x_{\mathsf{k}}, x_{\mathsf{plain}})) \to x_{\mathsf{plain}}$$

Using constructors sign, blind, and pk, we model blind signatures as follows: $\mathsf{sign}(x_{\mathsf{sk}}, x_{\mathsf{msg}})$ is the signature of message x_{msg} under secret key x_{sk}, $\mathsf{blind}(x_{\mathsf{k}}, x_{\mathsf{msg}})$ is the blinding of message x_{msg} with coins x_{k}, and $\mathsf{pk}(x_{\mathsf{sk}})$ is the public key corresponding to the secret key x_{sk}. We also use three destructors: checksign to verify signatures, getmsg to model that an adversary may recover the message from the signature, even without the public key, and unblind for unblinding, defined by the following rewrite rules:

$$\begin{aligned}
\mathsf{checksign}(\mathsf{pk}(x_{\mathsf{sk}}), \mathsf{sign}(x_{\mathsf{sk}}, x_{\mathsf{msg}})) &\to x_{\mathsf{msg}} \\
\mathsf{getmsg}(\mathsf{sign}(x_{\mathsf{sk}}, x_{\mathsf{msg}})) &\to x_{\mathsf{msg}} \\
\mathsf{unblind}(x_{\mathsf{k}}, \mathsf{sign}(x_{\mathsf{sk}}, \mathsf{blind}(x_{\mathsf{k}}, x_{\mathsf{msg}}))) &\to \mathsf{sign}(x_{\mathsf{sk}}, x_{\mathsf{msg}}) \\
\mathsf{unblind}(x_{\mathsf{k}}, \mathsf{blind}(x_{\mathsf{k}}, x_{\mathsf{plain}})) &\to x_{\mathsf{plain}}
\end{aligned}$$

With blind signatures, a signer may sign a blinded message without learning the plaintext message, and the signature on the plaintext message can be recovered by unblinding, as shown by the third rewrite rule.

2) Protocol description: The protocol uses two authorities, a *registrar* and a *tallier*, and it is divided into four phases, *setup*, *preparation*, *commitment*, and *tallying*. The setup phase proceeds as follows.

1) The registrar creates a signing key pair sk_R and $\mathsf{pk}(sk_R)$, and publishes the public part $\mathsf{pk}(sk_R)$. In addition, each voter is assumed to have a signing key pair sk_V and $\mathsf{pk}(sk_V)$, where the public part $\mathsf{pk}(sk_V)$ has been published.

The preparation phase then proceeds as follows.

2) The voter chooses coins k and k', computes the commitment to her vote $M = \mathsf{commit}(k, v)$ and the signed blinded commitment $\mathsf{sign}(sk_V, \mathsf{blind}(k', M))$, and sends the signature, paired with her public key, to the registrar.
3) The registrar checks that the signature belongs to an eligible voter and returns the blinded commitment signed by the registrar $\mathsf{sign}(sk_R, \mathsf{blind}(k', M))$.
4) The voter verifies the registrar's signature and unblinds the message to recover $\hat{M} = \mathsf{sign}(sk_R, M)$, that is, her commitment signed by the registrar.

After a deadline, the protocol enters the commitment phase.

5) The voter posts her ballot $\hat{M}$ to the bulletin board.

Similarly, the tallying phase begins after a deadline.

6) The tallier checks validity of all signatures on the bulletin board and prepends an identifier ℓ to each valid entry.
7) The voter checks the bulletin board for her entry, the pair $\ell, \hat{M}$, and appends the commitment factor k.
8) Finally, using k, the tallier opens all of the ballots and announces the election outcome.

The distinction between phases is essential to uphold the protocol's security properties. In particular, voters must synchronise before the commitment phase to ensure ballot secrecy (observe that without synchronisation, traffic analysis may allow the voter's signature to be linked with the commitment to her vote – this is trivially possible when a voter completes the commitment phase before any other voter starts the preparation phase, for instance – which can then be linked to her vote) and before the tallying phase to avoid publishing partial results, that is, to ensure *fairness* (see Cortier & Smyth [48] for further discussion on fairness).

3) Model: To analyse ballot secrecy, it suffices to model the participants that must be honest (i.e., must follow the protocol description) for ballot secrecy to be satisfied. All the remaining participants are controlled by the adversary. The FOO protocol assures ballot secrecy in the presence of dishonest authorities if the voter is honest. Hence, it suffices to model the voter's part of FOO as a process.

Definition 8. The process $P_{\mathsf{foo}}(x_{\mathsf{sk}}, x_{\mathsf{vote}})$ modelling a voter in FOO, with signing key x_{sk} and vote x_{vote}, is defined as follows

$$\begin{array}{ll}
\nu\, k.\nu\, k'. & \%\ \text{Step 2} \\
\text{let } M = \mathsf{commit}(k, x_{\mathsf{vote}}) \text{ in} & \\
\text{let } M' = \mathsf{blind}(k', M) \text{ in} & \\
\overline{c}\langle(\!|\mathsf{pk}(x_{\mathsf{sk}}), \mathsf{sign}(x_{\mathsf{sk}}, M')|\!)\rangle. & \\
c(y). & \%\ \text{Step 4} \\
\text{let } y' = \mathsf{checksign}(\mathsf{pk}(sk_R), y) \text{ in} & \\
\text{if } y' = M' \text{ then} & \\
\text{let } \hat{M} = \mathsf{unblind}(k', y) \text{ in} & \\
1{::}\, \overline{c}\langle\hat{M}\rangle. & \%\ \text{Step 5} \\
2{::}\, c(z).\text{let } z_2 = \pi_{2,2}(z) \text{ in} & \%\ \text{Step 7} \\
\text{if } z_2 = \hat{M} \text{ then } \overline{c}\langle(\!|z, k|\!)\rangle &
\end{array}$$

The process $P_{\mathsf{foo}}(sk_1, v_1) \mid \cdots \mid P_{\mathsf{foo}}(sk_n, v_n)$ models an election with n voters casting votes $v_1, \ldots, v_n$ and encodes the separation of phases using barriers.

4) Analysis: ballot secrecy: Based upon [2], [49] and as outlined in Section I, we formalise ballot secrecy for two voters A and B with the assertion that an adversary cannot distinguish between a situation in which voter A votes for candidate v and voter B votes for candidate v', from another one in which A votes v' and B votes v. We use the biprocess $P_{\mathsf{foo}}(sk_A, \mathsf{diff}[v, v'])$ to model A and the biprocess $P_{\mathsf{foo}}(sk_B, \mathsf{diff}[v', v])$ to model B, and formally express ballot secrecy as an equivalence which can be checked using Theorem 7. Voters' keys are modelled as free names, since ballot secrecy can be achieved without confidentiality of these keys. (Voters' keys *must* be secret for other properties.)

Definition 9 (Ballot secrecy). FOO preserves *ballot secrecy* if the biprocess $Q_{\mathsf{foo}} \triangleq P_{\mathsf{foo}}(sk_A, \mathsf{diff}[v, v']) \mid P_{\mathsf{foo}}(sk_B, \mathsf{diff}[v', v])$ satisfies observational equivalence.

To provide further insight into how our compiler works, let us consider how to informally prove this equivalence: that $\mathsf{fst}(Q_{\mathsf{foo}})$ is indistinguishable from $\mathsf{snd}(Q_{\mathsf{foo}})$. Before the first barrier, A outputs

$$(\!|\mathsf{pk}(sk_A),\ \mathsf{sign}(sk_A, \mathsf{blind}(k'_a, \mathsf{commit}(k_a, v)))|\!)$$

in $\mathsf{fst}(Q_{\mathsf{foo}})$ and

$$(\!|\mathsf{pk}(sk_A),\ \mathsf{sign}(sk_A, \mathsf{blind}(k'_a, \mathsf{commit}(k_a, v')))|\!)$$

in $\mathsf{snd}(Q_{\mathsf{foo}})$, where the name k'_a remains secret. By the equational theory for blinding, N can only be recovered from $\mathsf{blind}(M, N)$ if M is known, so these two messages are indistinguishable. The situation is similar for B. Therefore, before the first barrier, A moves in $\mathsf{fst}(Q_{\mathsf{foo}})$ are mimicked by A moves in $\mathsf{snd}(Q_{\mathsf{foo}})$ and B moves in $\mathsf{fst}(Q_{\mathsf{foo}})$ are mimicked by B moves in $\mathsf{snd}(Q_{\mathsf{foo}})$.

Let us define $\mathsf{sc}(k,v) \triangleq \mathsf{sign}(sk_R, \mathsf{commit}(k,v))$. After the first barrier, A outputs

$$\mathsf{sc}(k_a,v) \text{ and } (\!(\!(\ell_1,\ \mathsf{sc}(k_a,v))\!),\ k_a)\!) \text{ in } \mathsf{fst}(Q_{\mathsf{foo}})$$
$$\mathsf{sc}(k_a,v') \text{ and } (\!(\!(\ell_1,\ \mathsf{sc}(k_a,v'))\!),\ k_a)\!) \text{ in } \mathsf{snd}(Q_{\mathsf{foo}})$$

where ℓ_1 is chosen by the adversary. It follows that A reveals her vote v in $\mathsf{fst}(Q_{\mathsf{foo}})$ and her vote v' in $\mathsf{snd}(Q_{\mathsf{foo}})$, so these messages are distinguishable. However, B outputs

$$\mathsf{sc}(k_b,v') \text{ and } (\!(\!(\ell_2,\ \mathsf{sc}(k_b,v'))\!)\ k_b)\!) \text{ in } \mathsf{fst}(Q_{\mathsf{foo}})$$
$$\mathsf{sc}(k_b,v) \text{ and } (\!(\!(\ell_2,\ \mathsf{sc}(k_b,v))\!)\ k_b)\!) \text{ in } \mathsf{snd}(Q_{\mathsf{foo}})$$

where ℓ_2 is similarly chosen by the adversary. Hence, B's messages in $\mathsf{snd}(Q_{\mathsf{foo}})$ are indistinguishable from A's messages in $\mathsf{fst}(Q_{\mathsf{foo}})$. Therefore, after the first barrier, A moves in $\mathsf{fst}(Q_{\mathsf{foo}})$ are mimicked by B moves in $\mathsf{snd}(Q_{\mathsf{foo}})$ and symmetrically, B moves in $\mathsf{fst}(Q_{\mathsf{foo}})$ are mimicked by A moves in $\mathsf{snd}(Q_{\mathsf{foo}})$, that is, the roles are swapped at the first barrier. Our compiler encodes the swapping, hence we can show that FOO satisfies ballot secrecy using Theorem 7. Moreover, ProVerif proves this result automatically. This proof is done for two honest voters, but it generalises immediately to any number of possibly dishonest voters, since other voters can be part of the adversary.

Showing that FOO satisfies ballot secrecy is not new: Delaune, Kremer & Ryan [2], [49] present a manual proof of ballot secrecy, Chothia *et al.* [50] provide an automated analysis in the presence of a passive adversary, and Delaune, Ryan & Smyth [38], Klus, Smyth & Ryan [41], and Chadha, Ciobâcă & Kremer [19], [20] provide automated analysis in the presence of an active adversary. Nevertheless, our analysis is useful to demonstrate our approach.

FOO does not satisfy receipt-freeness, because each voter knows the coins used to construct their ballot and these coins can be used as a witness to demonstrate how they voted. In an effort to achieve receipt-freeness, the protocol by Lee *et al.* [51] uses a hardware device to introduce coins into the ballot that the voter does not know.

B. Case study: Lee et al.

1) Protocol description: The protocol uses a registrar and some talliers, and it is divided into three phases, *setup*, *voting*, and *tallying*. For simplicity, we assume there is a single tallier. The setup phase proceeds as follows.

1) The tallier generates a key pair and publishes the public key.
2) Each voter is assumed to have a signing key pair and an offline tamper-resistant hardware device. The registrar is assumed to know the public keys of voters and devices. The registrar publishes those public keys.

The voting phase proceeds as follows.

3) The voter encrypts her vote and inputs the resulting ciphertext into her tamper-resistant hardware device.
4) The hardware device re-encrypts the voter's ciphertext, signs the re-encryption, computes a Designated Verifier Proof that the re-encryption was performed correctly, and outputs these values to the voter.
5) If the signature and proof are valid, then the voter outputs the re-encryption and signature, along with her signature of these elements.

The hardware device re-encrypts the voter's encrypted choice to ensure that the voter's coins cannot be used as a witness demonstrating how the voter voted. Moreover, the device is offline, thus communication between the voter and the device is assumed to be untappable, hence, the only meaningful relation between the ciphertexts input and output by the hardware device is due to the Designated Verifier Proof, which can only be verified by the voter.

Finally, the tallying phase proceeds as follows.

6) Valid ballots (that is, ciphertexts associated with valid signatures) are input to a mixnet and the mixnet's output is published. We model the mixnet as a collection of parallel processes that each input a ballot, verify the signatures, synchronise with the other processes, and finally output the ciphertext on an anonymous channel.
7) The tallier decrypts each ciphertext and announces the election outcome.

2) Analysis: ballot secrecy: In this protocol, the authorities and hardware devices must be honest for ballot secrecy to be satisfied, so we need to explicitly model them. Therefore, building upon (1), we formalise ballot secrecy by the equivalence

$$C[V(A,v) \mid V(B,v')] \approx C[V(A,v') \mid V(B,v)] \qquad (3)$$

where the process $V(A,v)$ models a voter with identity A (including its private key, its device public key, and its private channel to the device) voting v, and the context C models all other participants: authorities and hardware devices. (Other voters are included in C for privacy results concerning more than two voters.) With two voters, we prove ballot secrecy by swapping data at the synchronisation in the mixnet. With an unbounded number of honest voters, we prove ballot secrecy using Corollary 10 to model an unbounded number of voters by a replicated process. As far as we know, this is the first proof of this result.

With an additional dishonest voter, the proof of ballot secrecy fails. This failure does not come from a limitation of our approach, but from a ballot copying attack, already mentioned in the original paper [51, Section 6] and formalised in [52]: the dishonest voter can copy A's vote, as follows. The adversary observes A's encrypted vote on the bulletin board (since it is accompanied by the voter's signature), inputs the ciphertext to the adversary's tamper-resistant hardware device, uses the output to derive a related ballot, and derives A's vote from the election outcome, which contains two copies of A's vote.

3) Analysis: receipt-freeness: Following [2], receipt-freeness can be formalised as follows: there exists a process V' such that

$$V'^{\backslash chc} \approx V(A, v) \tag{4}$$

$$C[V(A, v')^{chc} \mid V(B, v)] \approx C[V' \mid V(B, v')] \tag{5}$$

where the context $C[\_]$ appears in (3), chc is a public channel, $V'^{\backslash chc} = \nu\, chc.(V \mid !chc(x))$, which is intuitively equivalent to removing all outputs on channel chc from V', and $V(A, v')^{chc}$ is obtained by modifying $V(A, v')$ as follows: we output on channel chc the private key of A, its device public key, all restricted names created by V, and messages received by V. Intuitively, the voter A tries to prove to the adversary how she voted, by giving the adversary all its secrets, as modelled by $V(A, v')^{chc}$. The process V' simulates a voter A that votes v, as shown by (4), but outputs messages on channel chc that aim to make the adversary think that it voted v'. The equivalence (5) shows that the adversary cannot distinguish voter A voting v' and trying to prove it to the adversary and voter B voting v, from V' and voter B voting v', so V' successfully votes v and deceives the adversary in thinking that it voted v'.

In the case of the Lee *et al*. protocol, V' is derived from $V(A, v)^{chc}$ by outputting on chc a fake Designated Verifier Proof that simulates a proof of re-encryption of a vote for v', instead of the Designated Verifier Proof that it receives from the device. Intuitively, the adversary cannot distinguish a fake proof from a real one, because only the voter can verify the proof.

The equivalence (4) holds by construction of V', because after removing outputs on chc, V' is exactly the same as $V(A, v)$. We prove (5) using our approach, for an unbounded number of honest voters. Hence, this protocol satisfies receipt-freeness for an unbounded number of honest voters. As far as we know, this is the first proof of this result. Obviously, receipt-freeness does not hold with dishonest voters, because it implies ballot secrecy.

4) Variant by Dreier, Lafourcade & Lakhnech: Dreier, Lafourcade & Lakhnech [52] introduced a variant of this protocol in which, in step 3, the voter additionally signs the ciphertext containing her vote, and in step 4, the hardware device verifies this signature. We have also analysed this variant using our approach. It is sufficiently similar to the original protocol that we obtain the same results for both.

5) Variant by Delaune, Kremer, & Ryan:

Protocol description: Delaune, Kremer, & Ryan [2] introduced a variant of this protocol in which the hardware devices are replaced with a single administrator, and the voting phase becomes:

3) The voter encrypts her vote, signs the ciphertext, and sends the ciphertext and signature to the administrator on a private channel.
4) The administrator verifies the signature, re-encrypts the voter's ciphertext, signs the re-encryption, computes a Designated Verifier Proof of re-encryption, and outputs these values to the voter.
5) If the signature and proof are valid, then the voter outputs her ballot, consisting of the signed re-encryption (via an anonymous channel).

The mixnet is replaced with the anonymous channel, and the tallying phase becomes:

6) The collector checks that the ballots are pairwise distinct, checks the administrator's signature on each of the ballots, and, if valid, decrypts the ballots and announces the election outcome.

Analysis: ballot secrecy: We have shown that this variant preserves ballot secrecy, with two honest voters, using our approach. In this proof, all keys are public and the collector is not trusted, so it is included in the adversary. Since the keys are public, any number of dishonest voters can also be included in the adversary, so the proof with two honest voters suffices to imply ballot secrecy for any number of possibly dishonest voters. Hence, this variant avoids the ballot copying attack and satisfies a stronger ballot secrecy property than the original protocol. Thus, we automate the proof made manually in [2]. For this variant, the swapping occurs at the beginning of the voting process, so we can actually prove the equivalence by proving diff-equivalence after applying the general property that $C[P \mid Q] \approx C[Q \mid P]$, much like for Example 1. Furthermore, an extension of ProVerif [53] takes advantage of this property to merge processes into biprocesses in order to prove observational equivalence. The approach outlined in that paper also succeeds in proving ballot secrecy for this variant. It takes 13 minutes 22 seconds, while our implementation with swapping takes 34 seconds. It spends most of the time computing the merged biprocesses; this is the reason why it is slower.

Analysis: receipt-freeness: We prove receipt-freeness for two honest voters. The administrator and voter keys do need to be secret, and all authorities need to be explicitly modelled. The process V' is built similarly to the one for the original protocol by Lee *et al*. Equivalence (4) again holds by construction of V'. To prove (5), much like in [2], we model the collector as parallel processes that each input one ballot, check the signature, decrypt, synchronise together, and output the decrypted vote:

$$\begin{aligned}&c(b); \text{let } ev = \mathsf{checksign}(pk_A, b) \text{ in}\\&\text{let } v = \mathsf{dec}(sk_C, ev) \text{ in } 2{::}\,\overline{c}\langle v\rangle\end{aligned}$$

There are as many such processes as there are voters, two in our case. However, such a collector does not check that the ballots are pairwise distinct: each of the two parallel processes has access to a single ballot, so each process individually cannot check that the two ballots are distinct. We implemented this necessary check by manually modifying the code generated by our compiler, by adding a check that the ballots are distinct in the process that swaps data. An excerpt of the obtained code follows:

$$
\begin{array}{ll}
 & (c(b); \mathsf{let}\ ev = \mathsf{checksign}(pk_A, b)\ \mathsf{in} \\
 & \quad \mathsf{let}\ v = \mathsf{dec}(sk_C, ev)\ \mathsf{in}\ \overline{a_1}\langle(\!|b, v|\!)\rangle; c_1(v'); \overline{c}\langle v'\rangle) \\
 & \mid (c(b); \mathsf{let}\ ev = \mathsf{checksign}(pk_A, b)\ \mathsf{in} \\
 & \quad \mathsf{let}\ v = \mathsf{dec}(sk_C, ev)\ \mathsf{in}\ \overline{a_2}\langle(\!|b, v|\!)\rangle; c_2(v'); \overline{c}\langle v'\rangle) \\
 & \mid (a_1((\!|b_1, v_1|\!)); a_2((\!|b_2, v_2|\!)); \\
(*) & \quad \mathsf{if}\ b_1 = b_2\ \mathsf{then}\ 0\ \mathsf{else} \\
 & \quad \overline{c_1}\langle \mathsf{diff}[v_1, v_2]\rangle; \overline{c_2}\langle \mathsf{diff}[v_2, v_1]\rangle)
\end{array}
$$

This code shows the two collectors and the process that swaps data. We use $a((\!|b, v|\!))$ as an abbreviation for $a(x)$; let $b = \pi_{1,2}(x)$ in let $v = \pi_{2,2}(x)$ in. The ballots are sent on channels a_1 and a_2 in addition to the decrypted votes, and we check that the two ballots are distinct at line $(*)$. With this code, ProVerif proves the diff-equivalence, so we have shown receipt-freeness for two honest voters. This proof is difficult to generalise to more voters in ProVerif, because in this case the collector should swap two ballots among the ones it has received (the two coming to the voters that swap their voters), but it has no means to detect which ones.

C. Other examples

The idea of swapping for proving equivalences has been applied by Dahl, Delaune & Steel [3] to prove privacy in a vehicular ad-hoc network [54]. They manually encode swapping based upon the informal idea of [38]. We have repeated their analysis using our approach. Thus, we automate the encoding of swapping in [3], and obtain stronger confidence in the results thanks to our soundness proof.

Backes, Hriţcu & Maffei [29] also applied the idea of swapping, together with other encoding tricks, to prove a privacy notion stronger than receipt-freeness, namely *coercion resistance*, of the protocol by Juels, Catalano & Jakobsson [55]. We did not try to repeat their analysis using our approach.

V. Conclusion

We extend the applied pi calculus to include barrier synchronisation and define a compiler to the calculus without barriers. Our compiler enables swapping data between processes at barriers, which simplifies proofs of observational equivalence. We have proven the soundness of our compiler and have implemented it in ProVerif, thereby extending the class of equivalences that can be automatically verified. The applicability of the results is demonstrated by analysing ballot secrecy and receipt-freeness in election schemes, as well as privacy in a vehicular ad-hoc network. The idea of swapping data at barriers was introduced in [38], without proving its soundness, and similar ideas have been used by several researchers [3], [29], so we believe that it is important to provide a strong theoretical foundation to this technique.

Acknowledgements: We are particularly grateful to Tom Chothia, Véronique Cortier, Andy Gordon, Mark Ryan, and the anonymous CSF reviewers, for their careful reading of preliminary drafts which led to this paper; their comments provided useful guidance. Birmingham's *Formal Verification and Security Group* provided excellent discussion and we are particularly grateful to: Myrto Arapinis, Sergiu Bursuc, Dan Ghica, and Eike Ritter, as well as, Mark and Tom, whom we have already mentioned. Part of the work was conducted while the authors were at École Normale Supérieure, Paris, France and while Smyth was at Inria, Paris, France and the University of Birmingham, Birmingham, UK.

References

[1] A. Pfitzmann and M. Köhntopp, "Anonymity, Unobservability, and Pseudonymity – A Proposal for Terminology," in *International Workshop on Design Issues in Anonymity and Unobservability*, ser. LNCS, vol. 2009. Springer, 2001, pp. 1–9, extended versions available at http://dud.inf.tu-dresden.de/Anon_Terminology.shtml.

[2] S. Delaune, S. Kremer, and M. D. Ryan, "Verifying privacy-type properties of electronic voting protocols," *Journal of Computer Security*, vol. 17, no. 4, pp. 435–487, 2009.

[3] M. Dahl, S. Delaune, and G. Steel, "Formal Analysis of Privacy for Vehicular Mix-Zones," in *ESORICS'10: 15th European Symposium on Research in Computer Security*, ser. LNCS, vol. 6345. Springer, 2010, pp. 55–70.

[4] M. Abadi and A. D. Gordon, "A Calculus for Cryptographic Protocols: The Spi Calculus," in *CCS'97: 4th ACM Conference on Computer and Communications Security*. ACM Press, 1997, pp. 36–47.

[5] M. Abadi and C. Fournet, "Mobile values, new names, and secure communication," in *POPL'01: 28th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. ACM Press, 2001, pp. 104–115.

[6] S. Delaune, S. Kremer, and O. Pereira, "Simulation based security in the applied pi calculus," in *FSTTCS: IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science*, ser. Leibniz International Proceedings in Informatics, vol. 4. Leibniz-Zentrum für Informatik, 2009, pp. 169–180.

[7] M. Abadi, "Security Protocols and their Properties," in *Foundations of Secure Computation*, ser. NATO Science Series. IOS Press, 2000, pp. 39–60.

[8] B. Blanchet, "Automatic Proof of Strong Secrecy for Security Protocols," in *S&P'04: 25th IEEE Symposium on Security and Privacy*. IEEE Computer Society, 2004, pp. 86–100.

[9] V. Cortier, M. Rusinowitch, and E. Zălinescu, "Relating two standard notions of secrecy," *Logical Methods in Computer Science*, vol. 3, no. 3, 2007.

[10] M. Abadi and A. D. Gordon, "A Bisimulation Method for Cryptographic Protocols," *Nordic Journal of Computing*, vol. 5, no. 4, pp. 267–303, 1998.

[11] J. Borgström and U. Nestmann, "On bisimulations for the spi calculus," *Mathematical Structures in Computer Science*, vol. 15, no. 3, pp. 487–552, 2005.

[12] J. Borgström, S. Briais, and U. Nestmann, "Symbolic Bisimulation in the Spi Calculus," in *CONCUR'04: 15th International Conference on Concurrency Theory*, ser. LNCS, vol. 3170. Springer, 2004, pp. 161–176.

[13] S. Delaune, S. Kremer, and M. D. Ryan, "Symbolic Bisimulation for the Applied Pi Calculus," *Journal of Computer Security*, vol. 18, no. 2, pp. 317–377, 2010.

[14] H. Hüttel, "Deciding Framed Bisimilarity," *Electronic Notes in Theoretical Computer Science*, vol. 68, no. 6, pp. 1–20, 2003, special issue Infinity'02: 4th International Workshop on Verification of Infinite-State Systems.

[15] L. Durante, R. Sisto, and A. Valenzano, "Automatic Testing Equivalence Verification of Spi Calculus Specifications," *ACM Transactions on Software Engineering and Methodology*, vol. 12, no. 2, pp. 222–284, 2003.

[16] V. Cortier and S. Delaune, "A method for proving observational equivalence," in *CSF'09: 22nd IEEE Computer Security Foundations Symposium*. IEEE Computer Society, 2009, pp. 266–276.

[17] A. Tiu and J. Dawson, "Automating open bisimulation checking for the spi calculus," in *CSF'10: 23rd IEEE Computer Security Foundations Symposium*. IEEE, 2010, pp. 307–321.

[18] V. Cheval, H. Comon-Lundh, and S. Delaune, "Trace equivalence decision: Negative tests and non-determinism," in *CCS'11: 18th ACM Conference on Computer and Communications Security*. ACM Press, 2011, pp. 321–330.

[19] R. Chadha, S. Ciobâca, and S. Kremer, "Automated Verification of Equivalence Properties of Cryptographic Protocols," in *ESOP'12: 21st European Symposium on Programming*, ser. LNCS, vol. 7211. Springer, 2012, pp. 108–127.

[20] S. Ciobâca, "Verification and Composition of Security Protocols with Applications to Electronic Voting," Ph.D. dissertation, LSV, ENS Cachan & CNRS & INRIA, 2011.

[21] M. Abadi and V. Cortier, "Deciding knowledge in security protocols under equational theories," *Theoretical Computer Science*, vol. 367, no. 1–2, pp. 2–32, 2006.

[22] B. Blanchet, M. Abadi, and C. Fournet, "Automated verification of selected equivalences for security protocols," *Journal of Logic and Algebraic Programming*, vol. 75, no. 1, pp. 3–51, 2008.

[23] S. Santiago, S. Escobar, C. Meadows, and J. Meseguer, "A formal definition of protocol indistinguishability and its verification using Maude-NPA," in *STM'14: Security and Trust Management*, ser. LNCS, vol. 8743. Springer, 2014, pp. 162–177.

[24] D. Basin, J. Dreier, and R. Casse, "Automated symbolic proofs of observational equivalence," in *CCS'15: 22nd ACM Conference on Computer and Communications Security*. ACM, 2015, pp. 1144–1155.

[25] M. Baudet, "Sécurité des protocoles cryptographiques : aspects logiques et calculatoires," Ph.D. dissertation, Laboratoire Spécification et Vérification, ENS Cachan, France, 2007.

[26] ——, "Deciding security of protocols against off-line guessing attacks," in *CCS'05: 12th ACM Conference on Computer and Communications Security*. ACM Press, 2005, pp. 16–25.

[27] R. Chrétien, V. Cortier, and S. Delaune, "Decidability of trace equivalence for protocols with nonces," in *CSF'15: 28th IEEE Computer Security Foundations Symposium*. IEEE Computer Society, 2015, pp. 170–184.

[28] ——, "From security protocols to pushdown automata," *ACM Transactions on Computational Logic*, vol. 17, no. 1:3, 2015.

[29] M. Backes, C. Hriţcu, and M. Maffei, "Automated Verification of Remote Electronic Voting Protocols in the Applied Pi-calculus," in *CSF'08: 21st IEEE Computer Security Foundations Symposium*. IEEE Computer Society, 2008, pp. 195–209.

[30] M. Dahl, S. Delaune, and G. Steel, "Formal Analysis of Privacy for Anonymous Location Based Services," in *TOSCA'11: Workshop on Theory of Security and Applications*, ser. LNCS, vol. 6993. Springer, 2011, pp. 98–112.

[31] M. K. Reiter and A. D. Rubin, "Crowds: Anonymity for web transactions," *ACM Transactions on Information and System Security*, vol. 1, no. 1, pp. 66–92, 1998.

[32] T. Chothia, "Analysing the MUTE Anonymous File-Sharing System Using the Pi-Calculus," in *FORTE'06: 26th International Conference on Formal Techniques for Networked and Distributed Systems*, ser. LNCS, vol. 4229. Springer, 2006, pp. 115–130.

[33] E. D. Brooks, III, "The Butterfly Barrier," *International Journal of Parallel Programming*, vol. 15, no. 4, pp. 295–307, 1986.

[34] D. Hensgen, R. Finkel, and U. Manber, "Two Algorithms for Barrier Synchronization," *International Journal of Parallel Programming*, vol. 17, no. 1, pp. 1–17, 1988.

[35] N. S. Arenstorf and H. F. Jordan, "Comparing barrier algorithms," *International Journal of Parallel Computing*, vol. 12, no. 2, pp. 157–170, 1989.

[36] B. D. Lubachevsky, "Synchronization Barrier and Related Tools for Shared Memory Parallel Programming," *International Journal of Parallel Programming*, vol. 19, no. 3, pp. 225–250, 1990.

[37] B. Blanchet and B. Smyth, "Automated reasoning for equivalences in the applied pi calculus with barriers," Inria, Research report RR-8906, 2016, available at https://hal.inria.fr/hal-01306440.

[38] S. Delaune, M. D. Ryan, and B. Smyth, "Automatic verification of privacy properties in the applied pi-calculus," in *IFIPTM'08: 2nd Joint iTrust and PST Conferences on Privacy, Trust Management and Security*, ser. International Federation for Information Processing, vol. 263. Springer, 2008, pp. 263–278.

[39] B. Smyth, "Automatic verification of privacy properties in the applied pi calculus," in *Formal Protocol Verification Applied: Abstracts Collection*, ser. Dagstuhl Seminar Proceedings, 2007, no. 07421.

[40] ——, "Formal verification of cryptographic protocols with automated reasoning," Ph.D. dissertation, School of Computer Science, University of Birmingham, 2011.

[41] P. Klus, B. Smyth, and M. D. Ryan, "ProSwapper: Improved equivalence verifier for ProVerif," http://www.bensmyth.com/proswapper.php, 2010.

[42] M. D. Ryan and B. Smyth, "Applied pi calculus," in *Formal Models and Techniques for Analyzing Security Protocols*. IOS Press, 2011, ch. 6.

[43] B. Blanchet, B. Smyth, and V. Cheval, "ProVerif 1.93: Automatic cryptographic protocol verifier, user manual and tutorial," http://proverif.inria.fr, 2016.

[44] M. Abadi and B. Blanchet, "Computer-assisted verification of a protocol for certified email," *Science of Computer Programming*, vol. 58, no. 1–2, pp. 3–27, 2005, special issue SAS'03.

[45] B. Blanchet, "Vérification automatique de protocoles cryptographiques: modèle formel et modèle calculatoire." Habilitation à diriger des recherches, Université Paris-Dauphine, 2008.

[46] M. Arapinis, J. Liu, E. Ritter, and M. Ryan, "Stateful applied pi calculus," in *POST'14: 3rd Conference on Principles of Security and Trust*, ser. LNCS, vol. 8414. Springer, 2014, pp. 22–41.

[47] B. Blanchet, "Automatic Verification of Correspondences for Security Protocols," *Journal of Computer Security*, vol. 17, no. 4, pp. 363–434, 2009.

[48] V. Cortier and B. Smyth, "Attacking and fixing Helios: An analysis of ballot secrecy," *Journal of Computer Security*, vol. 21, no. 1, pp. 89–148, 2013.

[49] S. Kremer and M. D. Ryan, "Analysis of an Electronic Voting Protocol in the Applied Pi Calculus," in *ESOP'05: 14th European Symposium on Programming*, ser. LNCS, vol. 3444. Springer, 2005, pp. 186–200.

[50] T. Chothia, S. Orzan, J. Pang, and M. T. Dashti, "A Framework for Automatically Checking Anonymity with μCRL," in *TGC'06: 2nd Symposium on Trustworthy Global Computing*, ser. LNCS, vol. 4661. Springer, 2007, pp. 301–318.

[51] B. Lee, C. Boyd, E. Dawson, K. Kim, J. Yang, and S. Yoo, "Providing Receipt-Freeness in Mixnet-Based Voting Protocols," in *ICISC'03: 6th International Conference on Information Security and Cryptology*, ser. LNCS, vol. 2971. Springer, 2004, pp. 245–258.

[52] J. Dreier, P. Lafourcade, and Y. Lakhnech, "Vote-independence: A powerful privacy notion for voting protocols," in *FPS'11: 4th Workshop on Foundations & Practice of Security*, ser. LNCS, vol. 6888. Springer, 2011, pp. 164–180.

[53] V. Cheval and B. Blanchet, "Proving more observational equivalences with ProVerif," in *POST'13: 2nd Conference on Principles of Security and Trust*, ser. LNCS, vol. 7796. Springer, 2013, pp. 226–246.

[54] J. Freudiger, M. Raya, M. Félegyházi, P. Papadimitratos, and J.-P. Hubaux, "Mix-zones for location privacy in vehicular networks," in *WiN-ITS'07: 1st International Workshop on Wireless Networking for Intelligent Transportation Systems*, 2007.

[55] A. Juels, D. Catalano, and M. Jakobsson, "Coercion-resistant electronic elections," in *Towards Trustworthy Elections: New Directions in Electronic Voting*, ser. LNCS. Springer, 2010, vol. 6000, pp. 37–63.