Proving Unlinkability using ProVerif through Desynchronized Bi-Processes

David Baelde, Alexandre Debant, Stéphanie Delaune

Proving Unlinkability using ProVerif through Desynchronised Bi-Processes

David Baelde
Univ Rennes, CNRS, IRISA, France

Alexandre Debant
Université de Lorraine, CNRS, Inria, LORIA, F-54000 Nancy, France

Stéphanie Delaune
Univ Rennes, CNRS, IRISA, France

***Abstract*—Unlinkability is a privacy property of crucial importance for several systems such as mobile phones or RFID chips. Analysing this security property is very complex, and highly error-prone. Therefore, formal verification with machine support is desirable. Unfortunately, existing techniques are not sufficient to directly apply verification tools to automatically prove unlinkability.**

In this paper, we overcome this limitation by defining a simple transformation that will exploit some specific features of ProVerif. This transformation, together with some generic axioms, allows the tool to successfully conclude on several case studies. We have implemented our approach, effectively obtaining direct proofs of unlinkability on several protocols that were, until now, out of reach of automatic verification tools.

I. Introduction

RFID tags are popping up everywhere. They are present in credit cards enabling wireless payments, in wireless keys for opening cars, in e-passport for storing fingerprints and iris scans, in e-ticketing systems for public transport, amusement parks, etc. Citizens do not expect to be traceable just because they carry an e-passport or a credit card. Many applications blatantly ignore such concerns. As a famous and widespread example, contactless credit cards reveal card numbers in clear [3]. More interestingly, subtle tracing attacks may be found even when this security goal is explicitly taken into account. For instance, the BAC protocol for e-passports can be exploited to trace citizens [15]. The attack relies on the possibility for an attacker to trace a passport by observing which error is obtained when replaying a particular message.

According to the ISO/IEC 15408-2 standard, a protocol is said to provide unlinkability when "a user may make multiple uses of [the protocol] without others being able to link these uses together". This is often defined as the fact that an attacker should not be able to distinguish a scenario in which the same agent (i.e., the user) is involved in many sessions from one that involves different agents in each session. Following this intuition, slightly different definitions have been proposed, e.g. [27], [1], [11], [2]. A comparison between some of them may be found in [12]. In this work, we consider a variant of the definition originally proposed in [1], and revisited in [2] to make it more suitable to analyse RFID protocols manipulating a database. This definition follows the intuition that an attacker should not be able to distinguish between the real situation where multiple users (e.g. tags) are present and involved in possibly many sessions, and the ideal one in which each user is playing at most one session (in which case it is obviously impossible to link sessions).

Designing protocols which actually achieve the required security goals is well-known to be error-prone. A common good practice is to formally analyse protocols using symbolic techniques and tools, as has been done, e.g., for TLS 1.3 [18], [5]. Aiming at machine support is really relevant since manual proofs are error-prone, tedious and hardly verifiable. Moreover, new protocols are developed quite frequently and need to be verified quickly. These symbolic techniques are mature for reachability properties like confidentiality or authentication, and some tools exist, e.g. Tamarin [25] and ProVerif [8], to ease the verification process. This approach has been extended to deal with privacy properties that are usually expressed through equivalences [9], [4]. Nevertheless, verifying a security property in such a setting, and especially unlinkability as defined above, remains a difficult problem. Actually, ProVerif and Tamarin can only prove a restricted form of equivalence, namely *diff-equivalence*, which (in its standard form) is too limiting to prove unlinkability.

Various avenues have been explored to overcome this limitation and go beyond diff-equivalence, with varying degrees of success. To the best of our knowledge, none of these works is fully satisfactory. Some works rely on a notion of unlinkability expressed using two phases (a learning phase followed by a guessing phase) which is more amenable to automation, but may be weaker than expected. This is the approach followed in [20] where several RFID protocols using the XOR operator have been analysed. However, the security analysis in this work is performed in a restricted setting considering only a fixed number of agents and/or sessions (typically 1 or 2). Another line of works consists of establishing unlinkability using an indirect approach, e.g. [11], [22], [2]. In a nutshell, these works propose conditions that are more easily verified using existing tools, and show that these conditions imply the strongest notion of unlinkability. The latest paper in this line of work, [2], tackles protocols involving a database. It provides an improved notion of unlinkability for these protocols and sufficient conditions that have enabled several first-time unlinkability proofs (as well as the discovery of some attacks). In

This work received funding from the ANR Chair AI ASAP (ANR-20-CHIA-0024) and the France 2030 program managed by the French National Research Agency under grant agreement No. ANR-22-PECY-0006.

the present paper, we shall consider the same class of protocols and the same notion of unlinkability. We will come back to [2] in section V-C to compare its outcomes with the ones resulting from the new approach that we propose. Finally, a natural solution to overcome the limitations of diff-equivalence would be to use *restrictions*, that have been available in Tamarin for a long time. They allow the user to restrict the analysis to a subset of traces, those satisfying the formulas expressed by the restrictions. A rather simple idea would be to apply such a restriction on one side to express the fact that each user can play only once. Unfortunately, restrictions turned out to be incorrectly handled by Tamarin[1] which might introduce a lack of soundness. Recently, the use of restrictions has been formally justified, but for a very specific class (namely Type-0) only [26]. This type of restrictions allows one to enforce protocol phases (used e.g. in the analysis performed in [20]) but cannot capture e.g. equality restrictions needed to model conditional branching. Therefore, the analyses performed in [26] to establish unlinkability revisiting the models provided in [20] have been done considering simplified models (without conditional branching).

Our contributions. The general purpose of our work is to explore how the tool ProVerif can be used to verify unlinkability properties.

We first clarify how the introduction of restrictions in ProVerif [10] can be used to prove meaningful equivalences. This is non-trivial because ProVerif only proves diff-equivalence on bi-processes, and restrictions in that setting formally express properties of bi-executions, while equivalences are more naturally expressed between two independent processes. We identify a class of so-called diff-safe restrictions which, intuitively, have a consistent meaning for the bi-process and its projections. Although this new tool gives us more flexibility to encode security properties, it is still insufficient to model unlinkability.

We then turn to our main contribution. We design transformations of ProVerif models (i.e. bi-processes equipped with axioms and lemmas) which can turn an original model for which ProVerif fails into an equivalent model (in a sense that we make precise later) for which ProVerif succeeds in practice. These transformations are not specific to unlinkability, but we have designed them to tackle the key difficulties faced when trying to automatically verify that property. Our transformations make use of several features which are specific to ProVerif:

- Our first transformation relies on a recent extension of the notion of bi-process. Roughly, a bi-process combines two processes, which are declared equivalent if they can always evolve simultaneously, even for actions that are not observable by the attacker. We propose to dissociate the two processes that form the bi-process when executing some specific unobservable instructions on which the two processes may diverge. This is possible thanks to an extended kind of bi-process, first considered in [23] and officially introduced in ProVerif in 2020 with version 2.02.
- This transformation alone is not sufficient for the tool to conclude on our case studies because of internal over-approximations. However, adding some generic axioms, as authorised by the latest version of ProVerif presented in [10], improves the accuracy enough to successfully conclude on all our examples.

We have implemented our approach in a tool and applied it on several case studies, obtaining in this way, for the first time, direct proofs of unlinkability for several protocols.

[1]https://github.com/tamarin-prover/tamarin-prover/issues/324

II. Modelling protocols

We recall first how cryptographic messages are modelled as terms in ProVerif. Building on this, we shall see how protocols are modelled as processes, and how protocol indistinguishability is formally expressed through process equivalences.

A. Term algebra

As usual in the symbolic setting, messages are modelled through a term algebra. We assume an infinite set $\mathcal{N}$ of *names* used to represent atomic data such as keys, or nonces, as well as an infinite set Σ_0 of *constants* used to represent atomic data known by the attacker; and two infinite and disjoint sets of *variables* $\mathcal{X}$ and $\mathcal{W}$. Variables in $\mathcal{X}$ are used to refer e.g. to input messages, and variables in $\mathcal{W}$, called *handles*, are used as pointers to messages learned by the attacker. We assume a finite *signature* Σ of *function symbols* with their arity partitioned into two disjoint subsets Σ_c, and Σ_d representing respectively the *constructors*, and *destructors*, and we denote $\Sigma_c^+ = \Sigma_c \uplus \Sigma_0$, and $\Sigma^+ = \Sigma \uplus \Sigma_0$.

We note $\mathcal{T}(\mathcal{F}, D)$ the set of terms built from elements of the set of atomic data D by applying function symbols in the signature $\mathcal{F}$. We refer to elements of $\mathcal{T}(\Sigma^+, \mathcal{N} \cup \mathcal{X})$ as *expressions* and elements of $\mathcal{T}(\Sigma_c^+, \mathcal{N} \cup \mathcal{X})$ as (constructor) terms. We define $\mathsf{vars}(M)$ as the set of variables that occur in an expression M. A term (resp. an expression) is *ground* when it contains no variable. We finally define $\mathsf{names}(M)$ as the set of names occurring in M. We will sometimes use $\mathsf{names}(M)$ and $\mathsf{vars}(D)$ as vectors, meaning that we select an arbitrary ordering over these finite sets.

In order to provide a meaning to constructor symbols, we equip (constructor) terms with an equational theory. We assume a set E of equations over $\mathcal{T}(\Sigma_c, \mathcal{X})$ and we define $=_{\mathsf{E}}$ as the smallest congruence containing E that is closed under substitutions and under bijective renaming. In addition, the semantics of destructor symbols is given by a set R of *ordered rewriting rules* of the form $\mathsf{g}(M_1, \ldots, M_n) \to M_0$ with $M_0, M_1, \ldots, M_n \in \mathcal{T}(\Sigma_c, \mathcal{X})$. A ground expression D can be rewritten in D' if there is a position p in D, a rewrite rule $\mathsf{g}(M_1, \ldots, M_n) \to M_0$ and a substitution θ from variables to ground terms such that $D|_p =_{\mathsf{E}} \mathsf{g}(M_1\theta, \ldots, M_n\theta)$, and $D' =_{\mathsf{E}} D[M_0\theta]_p$, i.e. D in which the subterm at position p has been replaced by $M_0\theta$. In the case where more than

one rule may be applied at position p, only the first such rule can be effectively used. Given a ground expression D, it may be possible to rewrite it (in an arbitrary number of steps) into a ground (constructor) term M: in that case, this term is noted $D\Downarrow$, and we say that D *evaluates to* $D\Downarrow$. We write $D\Downarrow$ when no such term exists, and say that the *computation fails*.

For modelling purposes, we split the signature $\Sigma_c \uplus \Sigma_d$ into two parts, Σ_{pub} and Σ_{priv}. An attacker builds his own messages by applying public function symbols to terms he already knows, and which are available to him through variables in $\mathcal{W}$. Formally, a computation done by the attacker is a *recipe*, i.e. a term in $\mathcal{T}(\Sigma^+_{\mathsf{pub}}, \mathcal{W})$ where $\Sigma^+_{\mathsf{pub}} = \Sigma_{\mathsf{pub}} \uplus \Sigma_0$.

Example 1: We consider $\Sigma_c = \{\langle\ \rangle,\ \mathsf{proj}_1,\ \mathsf{proj}_2,\ \mathsf{h}\}$, $\Sigma_d = \emptyset$, and we assume that all the function symbols are public. The symbol $\langle\ \rangle$ of arity 2 is used to model the pairing operator, and we have projection symbols proj_1 and proj_2 (both of arity 1) to access the components of a pair. The symbol h of arity 2 represents a keyed hash function. Given $n_T, k \in \mathcal{N}$, we have that $\langle n_T, \mathsf{h}(n_T, k)\rangle$ is a term.

The standard equational theory $\mathsf{E}_{\mathsf{pair}}$ modelling the pairing operator contains the two following equations: $\mathsf{proj}_1(\langle x_1, x_2\rangle) = x_1$, and $\mathsf{proj}_2(\langle x_1, x_2\rangle) = x_2$.

The informed reader will have noticed that we have chosen here to represent the projections using constructors with an appropriate equational theory. Another choice could have been to use destructors and rewriting rules (instead of equations). Both representations are possible in ProVerif. We made this choice to be consistent with the assumptions of the transformation presented in Section IV-C.

Example 2: To illustrate the notion of destructor symbols and ordered rewriting systems, we consider the signature $\Sigma' = \Sigma'_c \cup \Sigma'_d$ such that $\Sigma'_c = \Sigma_c \cup \{\mathtt{true}, \mathtt{false}\}$ and $\Sigma'_d = \{\mathsf{eq}\}$ together with the rewriting rules $\mathsf{eq}(x, x) \to \mathtt{true}$, and $\mathsf{eq}(x, y) \to \mathtt{false}$, in this order (in addition to the equational theory $\mathsf{E}_{\mathsf{pair}}$ given in Example 1). We assume again that all these function symbols are public.

Let $\mathsf{w} \in \mathcal{W}$, and $\mathsf{k}_0 \in \Sigma_0$. We have that $R = \mathsf{eq}(\mathsf{h}(\mathsf{proj}_1(\mathsf{w}), \mathsf{k}_0), \mathsf{proj}_2(\mathsf{w}))$ is a recipe, and $D = R\{\mathsf{w} \mapsto \langle n_T, \mathsf{h}(n_T, \mathsf{k}_0)\rangle\}$, i.e. R in which the occurrences of w are replaced by $\langle n_T, \mathsf{h}(n_T, \mathsf{k}_0)\rangle$, is a ground expression such that $D\Downarrow = \mathtt{true}$.

B. Process algebra

We consider two disjoint sets $\mathcal{C}_{\mathsf{pub}}$ and $\mathcal{C}_{\mathsf{priv}}$ of *channel names*. The channels in $\mathcal{C}_{\mathsf{pub}}$ are assumed to be public, whereas those in $\mathcal{C}_{\mathsf{priv}}$ are not. We denote $\mathcal{C} = \mathcal{C}_{\mathsf{pub}} \uplus \mathcal{C}_{\mathsf{priv}}$. We also consider a set Σ_e of *event symbols* and a set Σ_t of *table symbols*. An event symbol evt of arity k may be applied to k terms to form an event $evt(M_1, \ldots, M_k)$. We assume that all table symbols are of arity 1. Events will be used to record particular points in executions, while tables are used to store data across sessions.

Protocols will be modelled as processes using the grammar given in Figure 1. We will not comment on the standard constructs of this grammar (i.e. null process, name and channel restrictions, input, output, parallel composition, replication). The process $\mathtt{insert}\ tbl(M); P$ inserts a record in the table tbl then runs P, and $\mathtt{get}\ tbl(x)\ \mathtt{st}.\ D\ \mathtt{in}\ P\ \mathtt{else}\ Q$ looks for a record M in the table tbl such that the expression D evaluates to $\mathtt{true}$ when x is bound to M. When such a record is found, it runs P in which x has been replaced by M. Otherwise, it runs Q. The construct $\mathtt{let}\ x = D\ \mathtt{in}\ P\ \mathtt{else}\ Q$ combines a computation with a conditional: it attempts to evaluate D, executes P with x bound to the resulting message upon success, and Q otherwise. The construct $i : P$ models that the process P must be executed in phase i only. Finally, the last construct simply records the event e before executing P.

$$
\begin{array}{rcl}
P, Q & := & 0 \\
& | & \mathtt{new}\ n; P \\
& | & \mathsf{out}(c, M); P \\
& | & \mathtt{new}\ c_{\mathsf{priv}}; P \\
& | & \mathsf{in}(c, x); P \\
& | & \mathtt{let}\ x = D\ \mathtt{in}\ P\ \mathtt{else}\ Q \\
& | & \mathtt{event}(e); P \\
& | & \mathtt{insert}\ tbl(M); P \\
& | & !P \\
& | & \mathtt{get}\ tbl(x)\ \mathtt{st}.\ D\ \mathtt{in}\ P\ \mathtt{else}\ Q \\
& | & (P \mid Q) \\
& | & i : P
\end{array}
$$

In this figure, $c \in \mathcal{C}$, $M \in \mathcal{T}(\Sigma^+_c, \mathcal{N} \cup \mathcal{X})$ is a term, $x \in \mathcal{X}$, $n \in \mathcal{N}$, $c_{\mathsf{priv}} \in \mathcal{C}_{\mathsf{priv}}$, $tbl \in \Sigma_t$, $D \in \mathcal{T}(\Sigma^+, \mathcal{N} \cup \mathcal{X})$ is an expression and e is an event.

Fig. 1. Grammar for processes

The constructs in and $\mathtt{let}$ bind their variables x in their sub-processes P. The get construct binds x in D and P. The free variables $\mathsf{fv}(P)$ of a process P are defined accordingly. A process P is *closed* when $\mathsf{fv}(P) = \emptyset$.

The operational semantics of processes is given by a labelled transition system over configurations (denoted by K) representing the current state of the process.

Definition 1: A *configuration* is a tuple $(\mathcal{P}; \Phi; \mathcal{S}; i)$ with $i \in \mathbb{N}$ and such that:

- $\mathcal{P}$ is a multiset of closed processes;
- $\Phi = \{\mathsf{w}_1 \mapsto M_1, \ldots, \mathsf{w}_n \mapsto M_n\}$ is a *frame* representing the knowledge of the adversary, i.e. a substitution where $\mathsf{w}_1, \ldots, \mathsf{w}_n$ are handles and $M_1, \ldots, M_n$ are ground terms;
- $\mathcal{S}$ is a set of elements of the form $tbl(M)$ with $tbl \in \Sigma_t$, and M a ground term. This set $\mathcal{S}$ represents the content of all the tables.

We may write P for the configuration $(\{0 : P\}; \emptyset; \emptyset; 0)$, where we explicitly indicate that the process is intended to execute in phase 0.

The labelled transitions are defined using usual rules (see e.g. [17], [10]). Some rules are recalled in Figure 2. An adversary can send any message of their knowledge on a public channel through the IN rule, while communications are

IN $(\{i : \mathsf{in}(c,x); P\} \uplus \mathcal{P}; \Phi; \mathcal{S}; i) \xrightarrow{\mathsf{in}(c,R)} (\{i : P\{x \mapsto M\}\} \uplus \mathcal{P}; \Phi; \mathcal{S}; i)$
if $c \in \mathcal{C}_{\mathsf{pub}}$ and there exists a recipe R such that $R\Phi\Downarrow =_{\mathsf{E}} M$

IO $(\{i : \mathsf{in}(c,x); P\} \uplus \{i : \mathsf{out}(c,M); Q\} \uplus \mathcal{P}; \Phi; \mathcal{S}; i)$
$\xrightarrow{\tau} (\{i : P\{x \mapsto M\}\} \uplus \{i : Q\} \uplus \mathcal{P}; \Phi; \mathcal{S}; i)$ if $c \in \mathcal{C}_{\mathsf{priv}}$

INSERT $(\{i : \mathtt{insert}\ tbl(M); P\} \uplus \mathcal{P}\ \Phi\ \mathcal{S}; i) \xrightarrow{\tau} (\{i : P\} \uplus \mathcal{P}; \Phi; \mathcal{S} \cup \{tbl(M)\}; i)$

GET-THEN $(\{i : \mathtt{get}\ tbl(x)\ \mathtt{st.}\ D\ \mathtt{in}\ P\ \mathtt{else}\ Q\} \uplus \mathcal{P}; \Phi; \mathcal{S}; i) \xrightarrow{\tau} (\{i : P\{x \mapsto M\}\} \uplus \mathcal{P}; \Phi; \mathcal{S}; i)$
if there exists M such that $tbl(M) \in \mathcal{S}$, and $D\{x \mapsto M\}$ evaluates to true

GET-ELSE $(\{i : \mathtt{get}\ tbl(x)\ \mathtt{st.}\ D\ \mathtt{in}\ P\ \mathtt{else}\ Q\} \uplus \mathcal{P}; \Phi, \mathcal{S}; i) \xrightarrow{\tau} (\{i : Q\} \uplus \mathcal{P}; \Phi; \mathcal{S}; i)$
if for any M such that $tbl(M) \in \mathcal{S}$, $D\{x \mapsto M\}$ does not evaluate to true

MOVE $(\mathcal{P}; \Phi; \mathcal{S}; i) \xrightarrow{\mathsf{phase}\ (i+1)} (\mathcal{P}; \Phi; \mathcal{S}; i+1)$

PHASE $(\{i : i' : P\} \cup \mathcal{P}; \Phi; \mathcal{S}; i) \xrightarrow{\tau} (\{i' : P\} \cup \mathcal{P}; \Phi; \mathcal{S}; i)$

Fig. 2. Semantics for processes (selected rules)

synchronous on private channels (rule IO). Regarding tables, the semantics is given by the rules INSERT, GET-THEN, and GET-ELSE. We also have rules for the other constructs, and some rules to deal with phases, e.g. MOVE and PHASE. An *execution trace* is a finite sequence:

$$K_0 \xrightarrow{\ell_1} K_1 \xrightarrow{\ell_2} \ldots \xrightarrow{\ell_n} K_n.$$

The associated sequence of observables of such a trace is the sequence $\ell_1, \ldots, \ell_n$ from which τ actions have been removed. Given a closed process P, we denote $\mathtt{traces}(P)$ the set of execution traces from P. Given an integer j such that $0 \leq j \leq n$, we denote $T[j]$ the configuration K_j, i.e the configuration at step j in trace T.

Example 3: For illustration purposes, we consider the Basic Hash protocol as described in [11]. Each tag stores a secret key that is never updated, and the readers have access to a database containing all these keys. We have that:

$$T \to R : \langle n_T, \mathsf{h}(n_T, k)\rangle$$

where n_T is a fresh nonce and k is the secret key. When receiving a message, the reader checks that it is a pair whose second element is a hash of the first element of the pair with one of the keys from the database. The protocol can be modelled in our syntax as follows:

$$P_{\mathsf{BH}} = (0 : !\ \mathtt{new}\ k;\ \mathtt{insert}\ keys(k); 1 : !\ P_{\mathsf{T}})\ \mid\ (1 : !\ P_{\mathsf{R}})$$

where $k, n_T \in \mathcal{N}$, $y, z \in \mathcal{X}$, $\mathsf{ok}, \mathsf{error} \in \Sigma_0$, $keys \in \Sigma_t$, and $c_R, c_T \in \mathcal{C}_{\mathsf{pub}}$. Moreover, we have that:

- $P_{\mathsf{R}} = \mathsf{in}(c_R, x)$;
  $\mathtt{get}\ keys(y)\ \mathtt{st.}\ D\ \mathtt{in}\ \mathsf{out}(c_R, \mathsf{ok})\ \mathtt{else}\ \mathsf{out}(c_R, \mathsf{error})$;
- $D = \mathsf{eq}(\mathsf{proj}_2(x), \mathsf{h}(\mathsf{proj}_1(x), y))$; and
- $P_{\mathsf{T}} = \mathtt{new}\ n_T$; $\mathsf{out}(c_T, \langle n_T, \mathsf{h}(n_T, k)\rangle)$.

C. Trace equivalence

We now define the notion of *trace equivalence* on which our definition of unlinkability is based. Intuitively, two processes are trace equivalent if for each trace of one process, there is an indistinguishable trace of the other process. To define this formally, we first introduce *static equivalence* between frames. Intuitively, an attacker can distinguish two frames Φ and Φ' if there exists a test that fails in Φ and succeeds in Φ' (or the contrary).

Definition 2: Two frames Φ and Φ' are in *static equivalence*, written $\Phi \sim_s \Phi'$, when $\mathrm{dom}(\Phi) = \mathrm{dom}(\Phi')$ and for any recipes R_1 and R_2, we have that:
$R_1\Phi\Downarrow =_{\mathsf{E}} R_2\Phi\Downarrow$, if, and only if, $R_1\Phi'\Downarrow =_{\mathsf{E}} R_2\Phi'\Downarrow$.

Example 4: Continuing with the Basic Hash protocol, we consider the two following frames ($n_T, n'_T, k, k' \in \mathcal{N}$):

- $\Phi_{\mathsf{diff}} = \{\mathsf{w}_1 \mapsto \langle n_T, \mathsf{h}(n_T, k)\rangle;\ \mathsf{w}_2 \mapsto \langle n'_T, \mathsf{h}(n'_T, k')\rangle\}$; and
- $\Phi_{\mathsf{same}} = \{\mathsf{w}_1 \mapsto \langle n_T, \mathsf{h}(n_T, k)\rangle;\ \mathsf{w}_2 \mapsto \langle n'_T, \mathsf{h}(n'_T, k)\rangle\}$.

We have that $\Phi_{\mathsf{diff}} \sim_s \Phi_{\mathsf{same}}$. Intuitively, an attacker cannot distinguish two outputs from the same tag from two outputs performed by different tags.

We then extend the notion of equivalence to traces as follows: given two execution traces T and T' leading respectively to configurations $K = (\mathcal{P}; \Phi; \mathcal{S}; i)$ and $K' = (\mathcal{P}'; \Phi'; \mathcal{S}'; i')$, we write $T \cong T'$ when $i = i'$, T and T' have the same sequence of observables, and $\Phi \sim_s \Phi'$.

Finally, we define the notion of *trace equivalence* of two closed processes by considering all their execution traces.

Definition 3: Let P and P' be two closed processes. We say that P and P' are in *trace equivalence*, written $P \approx_t P'$, when, for any execution trace $T \in \mathtt{traces}(P)$, there exists $T' \in \mathtt{traces}(P')$ such that $T \cong T'$, and conversely.

We rely on this notion of trace equivalence to model unlinkability following the definition originally proposed in [1] and used in [21], [24] relying on a notion of equivalence stronger than the notion of trace equivalence. Actually, the notion of unlinkability we consider here is exactly the one studied in [2]. The purpose of this work is not to provide a discussion on how to model unlinkability but to show how to overcome the

limitations of the existing tools such ProVerif and Tamarin, and to show how to automate proofs of unlinkability in ProVerif relying on some recent advanced features added into the tool. Therefore, we simply illustrate the definition we will use here on the Basic Hash example.

Example 5: Going back to our running example, unlinkability will be modelled as the following equivalence:

$$P_{\mathsf{BH}} \overset{?}{\approx}_t (0 : !\,\mathtt{new}\ k;\ \mathtt{insert}\ keys(k); 1 :\ P_{\mathsf{T}})\ \mid\ (1 : !\,P_{\mathsf{R}})$$

This equivalence states that an attacker cannot distinguish the situation where many tags play multiple sessions from the situation where each tag can play only once. Note that the replication ! in front of process P_{T} has been removed on the right hand side of the equivalence.

Tools like Tamarin or ProVerif cannot currently prove this equivalence, as observed in [2]. We explain how ProVerif works and why it fails on our example in Section III. We then describe our approach to solve this issue in Section IV.

III. Proving equivalences using ProVerif

ProVerif can prove an equivalence between two processes P and Q that differ only by the terms and expressions they contain. In Section III-A, we explain this notion of bi-processes, and the notion of diff-equivalence that ProVerif considers. Then, we describe some features that have been recently introduced in ProVerif [10] and on which our approach will rely.

A. Bi-processes and diff-equivalence

Two processes that only differ by terms and expressions can be given by a single process in which the special function symbol $\mathtt{diff}[\cdot,\cdot]$ is used. Intuitively, the first argument corresponds to the term used in the first process, whereas the second one is the term used in the second process. More formally, the grammar we consider for bi-processes is the same as the one introduced previously with the addition of $\mathtt{diff}[M, M']$ for terms and the systematic use of $\mathtt{diff}[x, y]$ for variable bindings in let, get, and input commands. For instance an input bi-process is of the form $\mathsf{in}(c, \mathtt{diff}[x, y]); P$. The latter addition means that we do not consider ProVerif's standard bi-processes but their extension allowed by the `allowDiffPatterns` option.

When P is a bi-process we define $\mathsf{fst}(P)$ as the process obtained by replacing any subterm $\mathtt{diff}[M, M']$ by M in P. We define similarly $\mathsf{fst}(M)$ when M is a bi-term, and define symmetrically $\mathsf{snd}(P)$ and $\mathsf{snd}(M)$. ProVerif's standard bi-processes, without diff operators on variable bindings, can equivalently be characterised in our setting as *separated* bi-processes. To formally define this notion, we assume given a partition $\mathcal{X} = \mathcal{X}^{\mathsf{L}} \uplus \mathcal{X}^{\mathsf{R}}$ for variables.

Definition 4: A bi-process B is *separated* when its variable bindings are of the form $\mathtt{diff}[x^{\mathsf{L}}, x^{\mathsf{R}}]$ with $x^{\mathsf{L}} \in \mathcal{X}^{\mathsf{L}}$ and $x^{\mathsf{R}} \in \mathcal{X}^{\mathsf{R}}$, and all (bi)terms M occurring in B are such that $\mathsf{vars}(\mathsf{fst}(M)) \subseteq \mathcal{X}^{\mathsf{L}}$ and $\mathsf{vars}(\mathsf{snd}(M)) \subseteq \mathcal{X}^{\mathsf{R}}$.

When a closed bi-process P is separated, $\mathsf{fst}(P)$ and $\mathsf{snd}(P)$ are closed processes too.

Example 6: Let $B = \mathsf{in}(c, \mathtt{diff}[x^{\mathsf{L}}, x^{\mathsf{R}}]); \mathsf{out}(c, \langle x^{\mathsf{L}}, x^{\mathsf{R}}\rangle)$. We have that B is a closed bi-process since the variables x^{L} and x^{R} are bound by the input construct, but it is not separated. Indeed, we have that $M = \langle x^{\mathsf{L}}, x^{\mathsf{R}}\rangle$ occurs in B, and $\mathsf{vars}(\mathsf{fst}(M)) = \mathsf{vars}(\mathsf{snd}(M)) = \{x^{\mathsf{L}}, x^{\mathsf{R}}\}$. Thus, we have neither $\mathsf{vars}(\mathsf{fst}(M)) \subseteq \mathcal{X}^{\mathsf{L}}$ nor $\mathsf{vars}(\mathsf{snd}(M)) \subseteq \mathcal{X}^{\mathsf{R}}$. Moreover, $\mathsf{fst}(B) = \mathsf{in}(c, x^{\mathsf{L}}); \mathsf{out}(c, \langle x^{\mathsf{L}}, x^{\mathsf{R}}\rangle)$ is not closed. Note that $\mathsf{snd}(B)$ is not closed as well.

The semantics of bi-processes is defined by a labelled transition system over bi-configurations of the form $(\mathcal{P}; \Phi; \mathcal{S}; i)$ where:

- $\mathcal{P}$ is a multiset of closed bi-processes,
- Φ is a substitution mapping variables to bi-terms called a *bi-frame*,
- $\mathcal{S}$ is a *bi-store*, i.e. a set of items of the form $tbl(M)$ where tbl is a table identifier and M is a bi-term, and
- i is an integer.

This transition system over bi-configurations is actually an adaptation of the transition system for regular configurations. Some rules are given in Figure 3. When Φ is a bi-frame, we write Φ_{fst} for the frame of domain $\mathrm{dom}(\Phi)$ such that $\Phi_{\mathsf{fst}}(\mathsf{w}) = \mathsf{fst}(\Phi(\mathsf{w}))$. For a bi-store $\mathcal{S}$, we define $\mathcal{S}_{\mathsf{fst}} = \{tbl(\mathsf{fst}(M)) \mid tbl(M) \in \mathcal{S}\}$. We define similarly Φ_{snd} and $\mathcal{S}_{\mathsf{snd}}$. Note that, in this semantics, both sides of the bi-process have to progress in the same way. For instance, when executing a let instruction, the left and right processes have to progress in the same branch. When executing a get instruction, the left and right processes have to execute the same branch when they access to the same element.

For separated bi-processes, $(\mathcal{P}; \Phi; \mathcal{S}; i) \rightarrow_b (\mathcal{P}'; \Phi'; \mathcal{S}'; i)$ implies $(\mathsf{fst}(\mathcal{P}); \Phi_{\mathsf{fst}}; \mathcal{S}_{\mathsf{fst}}; i) \rightarrow (\mathsf{fst}(\mathcal{P}'); \Phi'_{\mathsf{fst}}; \mathcal{S}'_{\mathsf{fst}}; i)$ and similarly for the second projection. This does not hold in general and, as illustrated by Example 6, it may not even make sense as the projections are not necessarily closed processes.

An execution trace of a bi-process is called a *bi-execution trace*. Given a bi-execution

$$T = K_0 \xrightarrow{\ell_1}_b K_1 \xrightarrow{\ell_2}_b \ldots \xrightarrow{\ell_n}_b K_n$$

and an integer $0 \leq j \leq n$, we denote by $T[j]$ the bi-configuration at step j. As for processes, we may simply write B for the bi-configuration $(\{0 : B\}; \emptyset; \emptyset; 0)$.

The notion of convergence defined next captures the fact that the left and right processes always agree along a trace, in the sense that one of the above rules is always applicable unless both sides of the bi-process are blocked.

Definition 5: A bi-execution trace T *converges*, denoted $T{\downarrow}{\uparrow}$, when for all steps j, $T[j] = K = (\mathcal{P}; \Phi; \mathcal{S}; i)$ implies:

1) if $(i : \mathtt{get}\ tbl(\mathtt{diff}[x^{\mathsf{L}}, x^{\mathsf{R}}])\ \mathtt{st}.\ D\ \mathtt{in}\ P\ \mathtt{else}\ Q) \in \mathcal{P}$ then for all $tbl(M) \in \mathcal{S}$, we have that:

$$\mathsf{fst}(D)\{x^{\mathsf{L}} \mapsto \mathsf{fst}(M), x^{\mathsf{R}} \mapsto \mathsf{snd}(M)\}{\Downarrow} =_{\mathsf{E}} \mathtt{true}$$

if, and only if,

$$\text{IN}_b \quad (\{i : \mathsf{in}(c, \mathtt{diff}[x^{\mathsf{L}}, x^{\mathsf{R}}]); P\} \uplus \mathcal{P}; \Phi; \mathcal{S}; i) \xrightarrow{\mathsf{in}(c,R)} (\{i : P\{x^{\mathsf{L}} \mapsto M, x^{\mathsf{R}} \mapsto M'\}\} \uplus \mathcal{P}; \Phi; \mathcal{S}; i)$$
$$\text{if } c \in \mathcal{C}_{\mathsf{pub}} \text{ and } R\Phi_{\mathsf{fst}}\Downarrow =_{\mathsf{E}} M \text{ and } R\Phi_{\mathsf{snd}}\Downarrow =_{\mathsf{E}} M'$$

$$\text{IO}_b \quad (\{i : \mathsf{in}(c, \mathtt{diff}[x^{\mathsf{L}}, x^{\mathsf{R}}]); P\} \uplus \{i : \mathsf{out}(c, M); Q\} \uplus \mathcal{P}; \Phi; \mathcal{S}; i) \xrightarrow{\tau} (\{i : P\{x^{\mathsf{L}} \mapsto \mathsf{fst}(M), x^{\mathsf{R}} \mapsto \mathsf{snd}(M)\}\} \uplus \{i : Q\} \uplus \mathcal{P}; \Phi; \mathcal{S}; i) \text{ if } c \in \mathcal{C}_{\mathsf{priv}}$$

$$\text{LET-THEN}_b \quad (\{i : \mathtt{let}\ \mathtt{diff}[x^{\mathsf{L}}, x^{\mathsf{R}}] = D\ \mathtt{in}\ P\ \mathtt{else}\ Q\} \uplus \mathcal{P}; \Phi; \mathcal{S}; i) \xrightarrow{\tau}_b (\{i : P\{x^{\mathsf{L}} \mapsto \mathsf{fst}(D)\Downarrow, x^{\mathsf{R}} \mapsto \mathsf{snd}(D)\Downarrow\}\} \uplus \mathcal{P}; \Phi; \mathcal{S}; i)$$

$$\text{LET-ELSE}_b \quad (\{i : \mathtt{let}\ \mathtt{diff}[x^{\mathsf{L}}, x^{\mathsf{R}}] = D\ \mathtt{in}\ P\ \mathtt{else}\ Q\} \uplus \mathcal{P}; \Phi; \mathcal{S}; i) \xrightarrow{\tau}_b (\{i : Q\} \uplus \mathcal{P}; \Phi; \mathcal{S}; i) \text{ if } \mathsf{fst}(D)\not\Downarrow \text{ and } \mathsf{snd}(D)\not\Downarrow$$

$$\text{GET-THEN}_b \quad (\{i : \mathtt{get}\ tbl(\mathtt{diff}[x^{\mathsf{L}}, x^{\mathsf{R}}])\ \mathtt{st}.\ D\ \mathtt{in}\ P\ \mathtt{else}\ Q\} \uplus \mathcal{P}; \Phi; \mathcal{S}; i) \xrightarrow{\tau}_b (\{i : P\{x^{\mathsf{L}} \mapsto \mathsf{fst}(M), x^{\mathsf{R}} \mapsto \mathsf{snd}(M)\}\} \uplus \mathcal{P}; \Phi; \mathcal{S}; i)$$
$$\text{if there exists } tbl(M) \in \mathcal{S} \text{ such that } \mathsf{fst}(D)\{x^{\mathsf{L}} \mapsto \mathsf{fst}(M), x^{\mathsf{R}} \mapsto \mathsf{snd}(M)\}\Downarrow =_{\mathsf{E}} \mathsf{true} \text{ and } \mathsf{snd}(D)\{x^{\mathsf{L}} \mapsto \mathsf{fst}(M), x^{\mathsf{R}} \mapsto \mathsf{snd}(M)\}\Downarrow =_{\mathsf{E}} \mathsf{true}$$

$$\text{GET-ELSE}_b \quad (\{i : \mathtt{get}\ tbl(\mathtt{diff}[x^{\mathsf{L}}, x^{\mathsf{R}}])\ \mathtt{st}.\ D\ \mathtt{in}\ P\ \mathtt{else}\ Q\} \uplus \mathcal{P}; \Phi; \mathcal{S}; i) \xrightarrow{\tau}_b (\{i : Q\} \uplus \mathcal{P}; \Phi; \mathcal{S}; i)$$
$$\text{if for all } tbl(M) \in \mathcal{S},\ \mathsf{fst}(D)\{x^{\mathsf{L}} \mapsto \mathsf{fst}(M), x^{\mathsf{R}} \mapsto \mathsf{snd}(M)\} \text{ does not evaluate to } \mathsf{true} \text{ and } \mathsf{snd}(D)\{x^{\mathsf{L}} \mapsto \mathsf{fst}(M), x^{\mathsf{R}} \mapsto \mathsf{snd}(M)\} \text{ does not evaluate to } \mathsf{true}$$

Fig. 3. Semantics for bi-processes

$\mathsf{snd}(D)\{x^{\mathsf{L}} \mapsto \mathsf{fst}(M), x^{\mathsf{R}} \mapsto \mathsf{snd}(M)\}\Downarrow =_{\mathsf{E}} \mathtt{true}$;

2) $(i : \mathtt{let}\ \mathtt{diff}[x^{\mathsf{L}}, x^{\mathsf{R}}] = D\ \mathtt{in}\ P\ \mathtt{else}\ Q) \in \mathcal{P}$ then $\mathsf{fst}(D)$ evaluates if, and only if, $\mathsf{snd}(D)$ evaluates;
3) $\mathsf{fst}(\Phi) \sim_s \mathsf{snd}(\Phi)$.

We naturally extend this notion to sets of bi-traces: $\mathcal{T}\downarrow\uparrow$ if, and only if, $T\downarrow\uparrow$ for all $T \in \mathcal{T}$. We finally say that diff-equivalence holds for a bi-process B when $\mathtt{traces}(B)\downarrow\uparrow$.

In [9], Blanchet *et al.* proved an important result for separated bi-processes: if $\mathtt{traces}(B)$ converges, i.e. $\mathtt{traces}(B)\downarrow\uparrow$, then $\mathsf{fst}(B)$ is observationally equivalent to $\mathsf{snd}(B)$. In short, and because observational equivalence implies trace equivalence [13], we have:

if B is separated and $\mathtt{traces}(B)\downarrow\uparrow$ then $\mathsf{fst}(B) \approx_t \mathsf{snd}(B)$

Example 7: Going back to our running example, we form the bi-process $B_{\mathsf{BH}} =$

$$(0 : !\,\mathtt{new}\ k^{\mathsf{L}}; !\,\mathtt{new}\ k^{\mathsf{R}}; \mathtt{insert}\ keys(\mathtt{diff}[k^{\mathsf{L}}, k^{\mathsf{R}}]); 1 : B_{\mathsf{T}}) \mid (1 : !\,B_{\mathsf{R}})$$

where $B_{\mathsf{T}} = P_{\mathsf{T}}\{k \mapsto \mathtt{diff}[k^{\mathsf{L}}, k^{\mathsf{R}}]\}$, and B_{R} is obtained from P_{R} by changing x (resp. y) into $\mathtt{diff}[x^{\mathsf{L}}, x^{\mathsf{R}}]$ (resp. $\mathtt{diff}[y^{\mathsf{L}}, y^{\mathsf{R}}]$). This is a separated bi-process.

We can recognise the two processes compared in Example 5 as the projections of our bi-process. Indeed, when considering the first projection $\mathsf{fst}(B_{\mathsf{BH}})$, and after having removed the name k^{R} that is not used in the projected process, we get

$$(0 : !\,\mathtt{new}\ k^{\mathsf{L}}; !\mathtt{insert}\ keys(k^{\mathsf{L}}); 1 : \mathsf{fst}(B_{\mathsf{T}})) \mid (1 : !\,\mathsf{fst}(B_{\mathsf{R}})).$$

which corresponds to P_{BH} in Example 3 (note that $\mathsf{fst}(B_{\mathsf{T}}) = P_{\mathsf{T}}\{k \mapsto k^{\mathsf{L}}\}$ and $\mathsf{fst}(B_{\mathsf{R}}) = P_{\mathsf{R}}\{k \mapsto k^{\mathsf{R}}\}$). The unique difference is that the replication command is executed before the table insertion instead of before the process P_{T}. This has no significant impact; there will just be many instances of the same key k^{L} in the table. Similarly, when considering the second projection $\mathsf{snd}(B_{\mathsf{BH}})$, and after having removed the unused name k^{L}, we get

$$(0 : !\,!\,\mathtt{new}\ k^{\mathsf{R}}; \mathtt{insert}\ keys(k^{\mathsf{R}}); 1 : \mathsf{snd}(B_{\mathsf{T}})) \mid (1 : !\,\mathsf{snd}(B_{\mathsf{R}}))$$

which corresponds to the right-hand side process in Example 5. Semantically, the two consecutive replication operators are indeed equivalent to a single one.

In conclusion we have that $\mathsf{fst}(B_{\mathsf{BH}})$ represents the situation with multiple tags playing multiple sessions, while $\mathsf{snd}(B_{\mathsf{BH}})$ represents tags that play only once.

Unfortunately, the set $\mathtt{traces}(B_{\mathsf{BH}})$ of bi-execution traces is not convergent. Indeed, as explained in [2], starting from B_{BH}, we can reach a bi-configuration $(\mathcal{P}; \Phi; \mathcal{S}; i)$ such that $\mathcal{P}$ contains the bi-process

$$1 : \mathtt{get}\ keys(\mathtt{diff}[y^{\mathsf{L}}, y^{\mathsf{R}}])\ \mathtt{st}.\ D\sigma\ \mathtt{in}\ \ldots\ \mathtt{else}\ \ldots$$

where:

- $\sigma = \{x^{\mathsf{L}} \mapsto \mathsf{fst}(M), x^{\mathsf{R}} \mapsto \mathsf{snd}(M)\}$,
- $M = \langle n_T, \mathsf{h}(n_T, \mathtt{diff}[k^{\mathsf{L}}, k_1^{\mathsf{R}}])\rangle$, and
- $keys(\mathtt{diff}[k^{\mathsf{L}}, k_1^{\mathsf{R}}])$, $keys(\mathtt{diff}[k^{\mathsf{L}}, k_2^{\mathsf{R}}]) \in \mathcal{S}$.

Considering the element $keys(\mathtt{diff}[k^{\mathsf{L}}, k_2^{\mathsf{R}}]) \in \mathcal{S}$, we can see that item 1 of Definition 5 is not satisfied.

Although Tamarin relies on a different approach to establish equivalences, a similar encoding adapted to the syntax of Tamarin will lead to the exact same issue (see file BasicHash-Tamarin.spthy in our repository).

B. Analysing bi-processes with restrictions, lemmas and axioms

An extension of ProVerif with restrictions, lemmas, and axioms has recently been introduced [10]. We will see that these features (especially axioms) are actually useful to establish unlinkability.

We consider *formulas* and *correspondence queries* given by the grammar below, where $e, e_1, \ldots, e_n$ are events and M, N are (constructor) terms without `diff` operators:

$$\begin{array}{rcl} \psi, \psi' & := & \mathsf{true} \mid \mathsf{false} \mid \mathtt{event}(e) \mid M = N \mid M \neq N \\ & & \mid \psi \wedge \psi' \mid \psi \vee \psi' \\ \rho & := & \mathtt{event}(e_1) \wedge \ldots \wedge \mathtt{event}(e_n) \Rightarrow \psi \end{array}$$

Intuitively, a trace T satisfies the correspondence query ρ (noted $T \vdash \rho$) if whenever T contains instances of $\mathtt{event}(e_i)$ at τ_i for each $1 \leq i \leq n$, then T also satisfies ψ (for some instantiation of the variables that occur in ψ but not in the e_i) where each event in ψ is found in T at some step τ that occurs before some τ_i. A closed process P satisfies ρ when $T \vdash \rho$ for any $T \in \mathtt{traces}(P)$.

This notion of correspondence queries is extended straightforwardly to bi-execution traces by considering that events contain bi-terms. The formal definition of satisfaction is actually the same as the one given above. We want to emphasise that this formal definition is different from the more natural one that would consist in checking satisfaction of each projection of the bi-restriction by the corresponding projection of the bi-trace. We first illustrate this on an example, and we will come back to this point in Section III-C.

Example 8: Consider the following bi-process B:

$$!\,\mathtt{new}\ k^{\mathsf{L}};\ \mathtt{new}\ k^{\mathsf{R}};$$
$$!\,\mathtt{new}\ sid;\ \mathtt{event}(e(sid, \mathtt{diff}[k^{\mathsf{L}}, k^{\mathsf{R}}]));\mathtt{out}(c, \mathtt{diff}[k^{\mathsf{L}}, k^{\mathsf{R}}]).$$

Let ρ_2 be the following correspondence query:

$$\begin{array}{r} \mathtt{event}(e(\mathtt{diff}[x_1, x_2], \mathtt{diff}[y_1, y_2])) \\ \wedge\, \mathtt{event}(e(\mathtt{diff}[x_1', x_2'], \mathtt{diff}[y_1', y_2])) \end{array} \Rightarrow\ x_2 = x_2'$$

We have that $\mathtt{traces}(B) \not\vdash \rho_2$. Indeed, consider

$$T = \left\{ \begin{array}{l} \mathtt{event}(e(sid_1, \mathtt{diff}[k^{\mathsf{L}}, k^{\mathsf{R}}])); \\ \mathtt{event}(e(sid_2, \mathtt{diff}[k^{\mathsf{L}}, k^{\mathsf{R}}])). \end{array} \right.$$

We have that $T \in \mathtt{traces}(B)$ and $T \not\vdash \rho_2$ as $sid_1 \neq sid_2$.

Correspondence queries may be used in several ways in the verification of a bi-process. We call *restriction* a correspondence query that is going to be used to restrict the set of bi-execution traces that we want to consider. We call *axiom* a correspondence query that holds for all bi-execution traces, and we call it a *lemma* when this fact should be verified by ProVerif instead of just being assumed. The next theorem sums up what it means to verify a bi-process in presence of axioms, lemmas and restrictions using ProVerif's semi-decision procedure prove' for equivalence queries — axioms and lemmas are similarly handled in this theorem, but in practice axioms would be trusted while lemmas would be verified.

Given a closed bi-process B and a set of restrictions $\mathcal{R}$, we note $\mathtt{traces}^{\mathcal{R}}(B) = \mathtt{traces}(B) \cap \{T \mid T \vdash \rho, \forall \rho \in \mathcal{R}\}$. Moreover, we note $\mathsf{fst}(\mathcal{R}) = \{\mathsf{fst}(\rho) \mid \rho \in \mathcal{R}\}$ and $\mathsf{snd}(\mathcal{R}) = \{\mathsf{snd}(\rho) \mid \rho \in \mathcal{R}\}$.

Example 9: Going back to Example 8, we have that:

$$\mathsf{fst}(\rho_2) = \mathtt{event}(e(x_1, y_1)) \wedge \mathtt{event}(e(x_1', y_1')) \Rightarrow x_2 = x_2'$$
$$\mathsf{snd}(\rho_2) = \mathtt{event}(e(x_2, y_2)) \wedge \mathtt{event}(e(x_2', y_2)) \Rightarrow x_2 = x_2'$$

It is important to note that we have $\mathsf{snd}(T) \not\vdash \mathsf{snd}(\rho_2)$ as expected but $\mathsf{fst}(T) \vdash \mathsf{fst}(\rho_2)$. Indeed, according to ProVerif semantics, variables that appear on the conclusion of a query (or lemma and restriction) and that are not introduced in its premises, are existentially quantified. This means that $\mathsf{fst}(\rho_2)$ is satisfied by any trace. We wanted to point out this (maybe unexpected) semantics feature of ProVerif to ease the understanding of some examples.

Theorem 1 ([10]): Let B be a closed bi-process, $\mathcal{A}x$, $\mathcal{L}$, $\mathcal{R}$ be respectively sets of axioms, lemmas, and restrictions. If

- for all $\varrho \in \mathcal{A}x \cup \mathcal{L}$, we have that $\mathtt{traces}^{\mathcal{R}}(B) \vdash \varrho$; and
- $\mathsf{prove}'(B, \mathcal{A}x, \mathcal{L}, \mathcal{R})$ returns true

then $\mathtt{traces}^{\mathcal{R}}(B)\downarrow\uparrow$.

This result has been stated and proved in [10, Theorem 1] for separated bi-processes only. We are confident that it remains true for the more general bi-processes that we obtain after the transformations presented in this paper. To justify this claim, it is not feasible to adapt the proofs of the technical report associated to [10], which rely on a complex technical condition (called well-origination) that cannot be adapted without changing large proofs involving the inner workings of ProVerif's saturation procedure. However, the authors of [10] have indicated in personal communication that they hope to lift these conditions completely in order to obtain results that would be easier to reuse in future developments, but also to justify some recent features of ProVerif that would, in turn, make it easy to justify the soundness of the verification procedure on our extended bi-processes. More details are available in appendix A.

Without restrictions and when B is separated, the conclusion of Theorem 1 could be $\mathsf{fst}(B) \approx_t \mathsf{snd}(B)$. However, as already noted in [10], with restrictions, it does not directly show trace equivalence (or observational equivalence) for traces satisfying their respective restrictions (obtained through projections). We illustrate this observation below through an example.

Example 10: Going back to our running example, a natural way to encode unlinkability would be to consider the (bi)process P_{BH} as described in Example 3, and use a restriction to discard some traces on the right-hand side, enforcing that each tag executes only once. Applying this idea on the toy example developed in Example 8, we will consider the restriction ρ_2 which says that if two sessions of the same tag occur in a trace (on the right side of the bi-process), then this should be the same session. This is ensured by requiring that the session ids are the same, i.e. $x_2 = x_2'$.

When analysing an equivalence property, a restriction in ProVerif has to be seen as a bi-restriction, and will discard

bi-traces. Therefore, such a restriction will also discard some behaviours on the left-hand side. Thus, our example restriction ρ_2 actually does not encode the unlinkability property we want: Considering the process B given in Example 8, it is relatively easy to see that $\mathtt{traces}(B)\downarrow\uparrow$ (both sides of the bi-process are equal up to a renaming). Thus, we also have convergence for the restricted set of bi-traces, i.e.

$$\mathtt{traces}^{\rho_2}(B)\downarrow\uparrow$$

while our process clearly does not satisfy unlinkability.

Regarding Tamarin, restrictions and lemmas are something that have been available for quite a long time. However, the soundness of the tool in presence of restrictions for equivalence properties has only been established recently [26], and only for a specific type of restrictions which are actually not sufficient to write the restriction ρ_2 mentioned above.

C. Trace equivalence in presence of restrictions

In presence of restrictions, and when B is separated, we would like to be able to establish that $\mathtt{traces}^{\mathcal{R}}(B)\downarrow\uparrow$ implies trace equivalence between $\mathsf{fst}(B)$ and $\mathsf{snd}(B)$ for traces satisfying their respective restrictions, i.e.

$$\mathtt{traces}^{\mathsf{fst}(\mathcal{R})}(\mathsf{fst}(B)) \approx_t \mathtt{traces}^{\mathsf{snd}(\mathcal{R})}(\mathsf{snd}(B)).$$

We first illustrate through examples why such a result does not hold in general.

Example 11: Consider the following bi-process

$$B = \mathtt{event}(e(\mathtt{diff}[\mathsf{ok}^{\mathsf{L}}, \mathsf{ok}^{\mathsf{R}}])); \mathtt{out}(c, \mathtt{diff}[\mathsf{bad}^{\mathsf{L}}, \mathsf{bad}^{\mathsf{R}}])$$

and the restriction

$$\rho = \mathtt{event}(e(\mathtt{diff}[x^{\mathsf{L}}, x^{\mathsf{R}}])) \Rightarrow x^{\mathsf{L}} = x^{\mathsf{R}}.$$

We have that $\mathtt{traces}^{\rho}(B)$ contains only the empty bi-trace. However, $\mathsf{fst}(\rho) \stackrel{\mathsf{def}}{=} \big(\mathtt{event}(e(x^{\mathsf{L}})) \Rightarrow x^{\mathsf{L}} = x^{\mathsf{R}}\big)$ will not discard any traces on $\mathsf{fst}(B)$: recall that x^{R} is quantified existentially because it only occurs in the conclusion of the clause. The same goes for $\mathsf{snd}(\rho)$ and $\mathsf{snd}(B)$. Therefore, we have that:

$$\mathtt{traces}^{\mathsf{fst}(\rho)}(\mathsf{fst}(B)) \not\approx_t \mathtt{traces}^{\mathsf{snd}(\rho)}(\mathsf{snd}(B))$$

Note that the restriction ρ applies in a symmetric way on $\mathsf{fst}(B)$ and $\mathsf{snd}(B)$ (a variable occurring free on the right, as x^{R}, being quantified existentially). The issue here comes from the fact that this restriction discards more traces when considering the bi-process than its two projections individually. Concretely, a non-empty trace tr_1 of $\mathsf{fst}(B)$ trivially satisfies the restriction $\mathsf{fst}(\rho)$ but cannot be lifted to a bi-trace of B that satisfies ρ, i.e. there is no bi-trace $\mathsf{tr} \in \mathtt{traces}^{\rho}(B)$ such that $\mathsf{fst}(\mathsf{tr}) = \mathsf{tr}_1$. The same issue holds for traces of $\mathsf{snd}(B)$.

Example 12: Consider the bi-process

$$B = \mathtt{event}(e(\mathtt{diff}[\mathsf{ok}^{\mathsf{L}}, \mathsf{ok}^{\mathsf{R}}], \mathsf{ok})); \mathtt{out}(c, \mathsf{ok})$$

and the following restriction:

$$\mathtt{event}(e(\mathtt{diff}[\mathsf{ok}^{\mathsf{R}}, \mathsf{ok}^{\mathsf{R}}], \mathtt{diff}[x, x])) \Rightarrow x \neq \mathsf{ok}$$

This restriction ρ will not discard any trace from $\mathtt{traces}(B)$ as any event e occurring in $\mathtt{traces}(B)$ will not have $\mathtt{diff}[\mathsf{ok}^{\mathsf{R}}, \mathsf{ok}^{\mathsf{R}}]$ as first argument. Thus, we have that $\mathtt{traces}^{\rho}(B)\downarrow\uparrow$, and it is possible to lift any trace possible on the left to a bi-trace T satisfying the bi-restrictions.

However, it is easy to see that the bi-trace

$$T = \mathtt{event}(e(\mathtt{diff}[\mathsf{ok}^{\mathsf{L}}, \mathsf{ok}^{\mathsf{R}}], \mathsf{ok})); \mathtt{out}(c, \mathsf{ok})$$

is such that $\mathsf{snd}(T) \not\vdash \mathsf{snd}(\rho)$, whereas $\mathsf{fst}(T) \vdash \mathsf{fst}(\rho)$, thus we have that:

$$\mathtt{traces}^{\mathsf{fst}(\rho)}(\mathsf{fst}(B)) \not\approx_t \mathtt{traces}^{\mathsf{snd}(\rho)}(\mathsf{snd}(B))$$

In the remaining of this section, we establish the expected result (see Proposition 1) for the class of restrictions that are *diff-safe*. First, as shown in Example 11, it is important to be able to lift a trace from the left-hand side and satisfying $\mathsf{fst}(\mathcal{R})$ to a bi-trace in $\mathtt{traces}^{\mathcal{R}}(B)$. Second, as shown in Example 12, we have also to ensure that this bi-trace will satisfy the restrictions on the right. This leads us to consider the following definition.

Definition 6: Let $\mathcal{R}$ be a set of bi-restrictions, and B a separated closed bi-process. We say that $\mathcal{R}$ is *diff-safe* w.r.t. B if for all $T_{\mathsf{fst}} \in \mathtt{traces}^{\mathsf{fst}(\mathcal{R})}(\mathsf{fst}(B))$, there exists $T \in \mathtt{traces}^{\mathcal{R}}(B)$ such that $\mathsf{fst}(T) \cong T_{\mathsf{fst}}$, and $\mathsf{snd}(T) \vdash \mathsf{snd}(\rho)$ for all $\rho \in \mathcal{R}$, and conversely for traces in $\mathtt{traces}^{\mathsf{snd}(\mathcal{R})}(\mathsf{snd}(B))$.

We establish the following result.

Proposition 1: Let B be a separated closed bi-process, and $\mathcal{R}$ be a set of restrictions that is diff-safe. If $\mathtt{traces}^{\mathcal{R}}(B)\downarrow\uparrow$ then

$$\mathtt{traces}^{\mathsf{fst}(\mathcal{R})}(\mathsf{fst}(B)) \approx_t \mathtt{traces}^{\mathsf{snd}(\mathcal{R})}(\mathsf{snd}(B)).$$

Proof. We establish the result for one direction. The other one can be obtained in a similar way. Let $T_{\mathsf{fst}} \in \mathtt{traces}^{\mathsf{fst}(\mathcal{R})}(\mathsf{fst}(B))$. We have to establish the existence of $T_{\mathsf{snd}} \in \mathtt{traces}^{\mathsf{snd}(\mathcal{R})}(\mathsf{snd}(B))$ such that $T_{\mathsf{fst}} \cong T_{\mathsf{snd}}$.

By definition of *diff-safe*, we know that there exits $T \in \mathtt{traces}^{\mathcal{R}}(B)$ such that $\mathsf{fst}(T) = T_{\mathsf{fst}}$ and $\mathsf{snd}(T) \vdash \mathsf{snd}(\rho)$ for all $\rho \in \mathcal{R}$. Relying on the fact that $\mathtt{traces}^{\mathcal{R}}(B)\downarrow\uparrow$, we deduce that $\mathsf{fst}(T) \cong \mathsf{snd}(T)$. As $T \in \mathtt{traces}^{\mathcal{R}}(B)$, we know that $\mathsf{snd}(T) \in \mathtt{traces}(\mathsf{snd}(B))$, and since $\mathsf{snd}(T) \vdash \mathsf{snd}(\rho)$ for all $\rho \in \mathcal{R}$, we deduce that $\mathsf{snd}(T) \in \mathtt{traces}^{\mathsf{snd}(\mathcal{R})}(\mathsf{snd}(B))$. □

Deciding whether a set of restrictions is diff-safe is not an easy task. As a stepping stone towards a syntactic sufficient condition, we propose in the next lemma a useful characterisation that is used later on to identify some classes of bi-restrictions which are diff-safe.

Lemma 1: Let $\mathcal{R}$ be a set of bi-restrictions, and B be a separated closed bi-process such that $\mathtt{traces}^{\mathcal{R}}(B)\downarrow\uparrow$, and such that for any $\rho \in \mathcal{R}$ and $T \in \mathtt{traces}(B)$, we have:

$$T \vdash \rho \;\Leftrightarrow\; \mathsf{fst}(T) \vdash \mathsf{fst}(\rho) \;\Leftrightarrow\; \mathsf{snd}(T) \vdash \mathsf{snd}(\rho)$$

We have that $\mathcal{R}$ is diff-safe w.r.t. B.

Proof. Let $T_{\mathsf{fst}} \in \mathtt{traces}^{\mathsf{fst}(\mathcal{R})}(\mathsf{fst}(B))$. We have to establish that there exists $T \in \mathtt{traces}^{\mathcal{R}}(B)$ such that $\mathsf{fst}(T) \cong$

T_{fst}, and $\mathsf{snd}(T) \vdash \mathsf{snd}(\rho)$ for all $\rho \in \mathcal{R}$. We show this result by induction on the length of T_{fst}. First, the result trivially holds for a trace of length 0. We now assume that $T_{\mathsf{fst}} = T'_{\mathsf{fst}} \cdot \alpha$. Because the restriction satisfaction relation is prefix closed, i.e. $T_{\mathsf{fst}} \vdash \rho$ implies $T'_{\mathsf{fst}} \vdash \rho$, we have that $T'_{\mathsf{fst}} \in \mathtt{traces}^{\mathsf{fst}(\mathcal{R})}(\mathsf{fst}(B))$. Applying the induction hypothesis, we deduce that there exists $T' \in \mathtt{traces}^{\mathcal{R}}(B)$ such that $\mathsf{fst}(T') \cong T'_{\mathsf{fst}}$, and $\mathsf{snd}(T') \vdash \mathsf{snd}(\rho)$ for all $\rho \in \mathcal{R}$. We know that $T' \in \mathtt{traces}(B)$, and we distinguish two cases:

- We have that there exists β such that $T = T'.\beta \in \mathtt{traces}(B)$ with $\mathsf{fst}(\beta) = \alpha$. In such a case, relying on our hypothesis, and since $\mathsf{fst}(T) \vdash \mathsf{fst}(\rho)$ for all $\rho \in \mathcal{R}$, we deduce that $T \vdash \rho$ for all $\rho \in \mathcal{R}$, and therefore $T \in \mathtt{traces}^{\mathcal{R}}(B)$. We have also that $\mathsf{fst}(T) \cong T_{\mathsf{fst}}$, and $\mathsf{snd}(T) \vdash \mathsf{snd}(\rho)$ relying on our hypothesis. This allows us to conclude.
- We have that there does not exist β such that $T = T'.\beta \in \mathtt{traces}(B)$ and $\mathsf{fst}(\beta) = \alpha$ meaning that $T' \in \mathtt{traces}^{\mathcal{R}}(B)$ diverges which is not possible by hypothesis.

This concludes the proof of the first direction, and the other direction can be done in a similar way. □

Example 13: Going back to Example 8, the restriction ρ_2 does not satisfy the criterion given in Lemma 1. It is indeed easy to see that for some $T \in \mathtt{traces}(B)$, $\mathsf{fst}(T)$ will satisfy $\mathsf{fst}(\rho_2)$ whereas $\mathsf{snd}(T)$ will not satisfied $\mathsf{snd}(\rho_2)$. Actually, ρ_2 is not diff-safe. Consider the following bi-trace:

$$T' = \left\{ \begin{array}{l} \mathtt{event}(e(sid_1, \mathtt{diff}[k^{\mathsf{L}}, k^{\mathsf{R}}])); \mathtt{out}(c, \mathtt{diff}[k^{\mathsf{L}}, k^{\mathsf{R}}]); \\ \mathtt{event}(e(sid_2, \mathtt{diff}[k^{\mathsf{L}}, k^{\mathsf{R}}])); \mathtt{out}(c, \mathtt{diff}[k^{\mathsf{L}}, k^{\mathsf{R}}]) \end{array} \right.$$

We have that $T_{\mathsf{fst}} \in \mathtt{traces}^{\mathsf{fst}(\rho_2)}(B)$. The only bi-trace T_0 such that $T_0 \in \mathtt{traces}^{\rho_2}(B)$ with $\mathsf{fst}(T_0) \cong T_{\mathsf{fst}}$ is actually T'. However, it happens that $\mathsf{snd}(T') \not\vdash \rho_2$.

In ProVerif, restrictions are actually bi-restrictions, and as we have seen in Proposition 1, a link between bi-restrictions and their projections has to be established to get the expected equivalence, i.e.

$$\mathtt{traces}^{\mathsf{fst}(\mathcal{R})}(\mathsf{fst}(B)) \approx_t \mathtt{traces}^{\mathsf{snd}(\mathcal{R})}(\mathsf{snd}(B)).$$

This way of considering restrictions (i.e. referring to their projections) is more in line with what is done in Tamarin. Even if restrictions have been available for quite a long time in Tamarin, the soundness of the procedure in presence of restrictions has only been established recently, and for Type-0 restrictions only [26]. Intuitively, a Type-0 restriction is a Tamarin restriction that can be written relying solely on facts of arity 0. The idea behind this condition is to ensure that a restriction will apply in a symmetric way on both sides (on the dependency graph and also on its mirror). It may seem quite restrictive. However, it is important to note that, in Tamarin, any fact has a timestamp embedded in it. Therefore, Type-0 restrictions are expressive enough to enforce protocol phases or to model the fact that a certain action occurs only once:

$$\mathsf{OnlyOnce}()@\#i \;\&\; \mathsf{OnlyOnce}()@\#j \Rightarrow \#i = \#j$$

Restrictions having a similar effect can also be written in ProVerif but restrictions with events of arity 0 are not sufficient for that.

Example 14: Let B be a bi-process (with no replication) containing several actions, e.g. $\mathtt{out}(c_1, u_1), \ldots, \mathtt{out}(c_n, u_n)$, and assume that we want to consider traces where at most one of these outputs is executed. The idea is to add $\mathtt{new}\ sid; \mathtt{event}(\mathsf{OnlyOnce}(sid))$ before each output, and to consider the following restriction:

$$\begin{array}{l} \mathtt{event}(\mathsf{OnlyOnce}(\mathtt{diff}[x, x])) \wedge \\ \mathtt{event}(\mathsf{OnlyOnce}(\mathtt{diff}[y, y])) \Rightarrow x = y \end{array}$$

It is easy to see that such a restriction ρ satisfies the hypothesis given in Lemma 1, and is thus diff-safe, allowing one to derive the expected result, i.e. $\mathtt{traces}^{\rho}(B)\downarrow\uparrow$ implies that $\mathtt{traces}^{\mathsf{fst}(\rho)}(\mathsf{fst}(B)) \approx_t \mathtt{traces}^{\mathsf{snd}(\rho)}(\mathsf{snd}(B))$. In other words, we get the soundness of ProVerif considering a more natural semantics for restrictions (we consider their respective projections).

Relying on diff-safe restrictions (or even simply the characterisation given in Lemma 1), we can also encode protocol phases (which is already a built-in mechanism in ProVerif). We believe that the criterion we provide with Lemma 1 to characterise diff-safe restrictions can be seen as the ProVerif counterpart of Type-0 restriction in the Tamarin setting. Thus, as done in [26] for Tamarin, we have also established the soundness of ProVerif w.r.t. diff-safe restrictions considering a more natural semantics for restrictions. Indeed, for this class of bi-restrictions, we can reason on their projections avoiding the issues mentioned in Examples 11 and 12.

However, the class of diff-safe restrictions is too restrictive to encode unlinkability as done e.g. in Example 10 (see Example 13). We thus develop another approach in the following section using the `allowDiffPatterns` option and axioms of ProVerif.

IV. Our approach

As we have seen in Example 7, ProVerif is not able to conclude on the bi-process modelling our unlinkability property on the Basic Hash protocol, and diff-safe restrictions are too restrictive to encode this property. We thus propose another approach. We propose to transform such bi-processes so that diff-equivalence may be achieved (and verified with ProVerif) on the transformed bi-processes, and that this diff-equivalence implies trace equivalence for the original process. We proceed in two steps:

1) We duplicate the get instructions to dissociate the process of the left and the one on the right.
2) We add some axioms to help ProVerif to reason on our bi-process. These axioms are used to improve the accuracy of the verification procedure by avoiding over-approximations.

Before formally defining these two steps in Sections IV-B and IV-C, we illustrate the approach on our running example.

A. Going back to our running example

On our running example, our transformations will only change the bi-process modelling the reader since it is the only one that features a get instruction. Instead of performing a get instruction to access one (bi)record in the keys table, it will perform two get instructions in a row to access two records in the $keys$ table. This will allow one to choose two different records: one for the left and one for the right. Of course, the test performed remains the same and has to be satisfied by the element chosen on the left, and the one chosen on the right. We show the transformed bi-process in Figure 4, where the symbol _ represents irrelevant names for variables or public channels, and bad_L and bad_R are distinct public constants. The reader can ignore the added events at this stage.

This transformation alone is not enough to allow ProVerif to conclude. We thus add some axioms that emphasise some properties that are necessarily satisfied, to make sure that these properties are not lost in the over-approximations performed in ProVerif's verification procedure. To this end, we make use of the three events introduced in the transformed bi-process: Inserted, FailL, and FailR. The first one indicates when an element is inserted in the table $keys$, whereas the two others will be used to track the outcome of lookups in that table: FailL indicates a failure of the get instruction regarding the left hand side of the process, whereas FailR indicates a failure of the get instruction w.r.t. the right process. Relying on these events, we state two axioms given in Figure 4.

Let us focus on the first one. In case the FailL event is executed with a value M, it means that there is no entry $\mathtt{diff}[N, N']$ in the table allowing one to evaluate the expression $\mathsf{eq}(\mathsf{proj}_2(M), \mathsf{h}(\mathsf{proj}_1(M), N))$ to true. Otherwise, we should have chosen this entry to execute the then branch. Thus, it is safe to say that for any entry $\mathtt{diff}[N, N']$ in the table $keys$, we have $\mathsf{proj}_2(M) \neq \mathsf{h}(\mathsf{proj}_1(M), N)$. The same applies on the right.

We may note that such an axiom is indeed satisfied since we know that insertions in the table can be done at the very beginning of the execution traces. This means that the element allowing one to perform the reasoning above can be assumed to be present before the execution of the FailL or FailR event. The validity of the axioms added by our transformation will be a consequence of Proposition 2.

B. Desynchronising the two parts of the bi-process

We now define the first step of the transformation informally described above.

Definition 7: Let B be a separated bi-process. We define $\mathcal{T}_1(B)$ as follows:

- $\mathcal{T}_1(\mathtt{get}\ tbl(\mathtt{diff}[x^\mathsf{L}, x^\mathsf{R}])\ \mathtt{st}.\ D\ \mathtt{in}\ B_\mathsf{then}\ \mathtt{else}\ B_\mathsf{else})$ is the following bi-process, where _ represents irrelevant names for variables or public channels, and $\mathsf{bad} = \mathtt{diff}[\mathsf{bad}_\mathsf{L}, \mathsf{bad}_\mathsf{R}]$ for two arbitrary distinct public constants bad_L and bad_R:

$$\begin{array}{l}\mathtt{get}\ tbl(\mathtt{diff}[x^\mathsf{L}, \_])\ \mathtt{st}.\ \mathsf{fst}(D)\ \mathtt{in}\\ \quad \mathtt{get}\ tbl(\mathtt{diff}[\_, x^\mathsf{R}])\ \mathtt{st}.\ \mathsf{snd}(D)\ \mathtt{in}\ \mathcal{T}_1(B_\mathsf{then})\\ \qquad\qquad\qquad\qquad\qquad\qquad \mathtt{else}\ \mathsf{out}(\_, \mathsf{bad})\\ \mathtt{else}\\ \quad \mathtt{get}\ tbl(\mathtt{diff}[\_, x^\mathsf{R}])\ \mathtt{st}.\ \mathsf{snd}(D)\ \mathtt{in}\ \mathsf{out}(\_, \mathsf{bad})\\ \qquad\qquad\qquad\qquad\qquad\qquad \mathtt{else}\ \mathcal{T}_1(B_\mathsf{else})\end{array}$$

- The transformation is homomorphic in all other cases: for instance, we define:

$-\mathcal{T}_1(\mathtt{out}(c, M); B') = \mathtt{out}(c, M); \mathcal{T}_1(B')$, and
$-\mathcal{T}_1(\mathtt{in}(c, \mathtt{diff}[x^\mathsf{L}, x^\mathsf{R}]); B') = \mathtt{in}(c, \mathtt{diff}[x^\mathsf{L}, x^\mathsf{R}]); \mathcal{T}_1(B')$.

Note that, because B is separated, $\mathsf{fst}(D)$ cannot have x^R as a free variable, which would render the above transformation semantically dubious. In the above definition, when $\mathsf{fst}(D)$ and $\mathsf{snd}(D)$ disagree, we use the bi-process $\mathsf{out}(\_, \mathtt{diff}[\mathsf{bad}_\mathsf{L}, \mathsf{bad}_\mathsf{R}])$ to signal that we want diff-equivalence to fail. To conclude in more cases, a variant could slightly relax this, under the assumption that B_then and B_else have the same structure. We do not encounter this situation in our case studies, and we left this extension as future work.

We establish the following result so that the "equivalence is true" returned by ProVerif on $\mathcal{T}_1(B_0)$ will allow one to conclude that trace equivalence holds between the two projections of B_0, and thus our unlinkability property holds.

Theorem 2: Let B_0 be a separated closed bi-process such that $\mathcal{T}_1(B_0)\downarrow\uparrow$. We have that $\mathsf{fst}(B_0) \approx_t \mathsf{snd}(B_0)$.

This theorem will be proved relying on two lemmas that we now introduce. In these lemmas, when T is a bi-execution trace of $\mathcal{T}_1(B_0)$, we allow ourselves to write $\mathsf{fst}(T)$ meaning that the first projection has been applied on each configuration of T. Note that this does not imply that $\mathsf{fst}(T)$ is an execution of $\mathsf{fst}(\mathcal{T}_1(B_0))$ as $\mathcal{T}_1(B_0)$ is not separated. Actually, we only use $\mathsf{fst}(T)$ to assess its indistinguishability w.r,t. another trace (e.g., in $T_\mathsf{fst} \cong \mathsf{fst}(T)$): we thus only care about the observables, and the first projection of the last frame of T. Since $\mathcal{T}_1(B_0)$ duplicates the get instruction, to establish the connection between $\mathsf{fst}(B_0)$ and $\mathcal{T}_1(B_0)$, it is easier to consider a notion of execution trace where the two get instructions corresponding to the same get instruction in the original bi-process B_0 are triggered in a row. We denote $\overline{\mathtt{traces}}$ this notion of bi-execution trace.

Lemma 2: Let B_0 be a separated closed bi-process such that $\mathcal{T}_1(B_0)\downarrow\uparrow$, and $T_\mathsf{fst} \in \mathtt{traces}(\mathsf{fst}(B_0))$. There exists $T \in \overline{\mathtt{traces}}(\mathcal{T}_1(B_0))$ such that $T_\mathsf{fst} \cong \mathsf{fst}(T)$. A similar result holds for snd.

Proof. (Sketch) We will find T such that $\mathsf{fst}(T)$ and T_fst differ only on the duplicated τ actions corresponding to transformed get operations. Although the lemma is stated in terms of static equivalence, frames are actually equal. The convergence hypothesis guarantees that we avoid $\mathtt{diff}[\mathsf{bad}_\mathsf{L}, \mathsf{bad}_\mathsf{R}]$ which can artificially appear in the traces of the transformed bi-process. □

$$
\begin{array}{l}
\mathtt{in}(c, \mathtt{diff}[x^{\mathsf{L}}, x^{\mathsf{R}}]);\\
\mathtt{get}\ keys(\mathtt{diff}[y^{\mathsf{L}}, \_])\ \mathtt{st.}\ \mathsf{eq}(\mathsf{proj}_2(x^{\mathsf{L}}), \mathsf{h}(\mathsf{proj}_1(x^{\mathsf{L}}, y^{\mathsf{L}})))\ \mathtt{in}\\
\quad \mathtt{get}\ keys(\mathtt{diff}[\_, y^{\mathsf{R}}])\ \mathtt{st.}\ \mathsf{eq}(\mathsf{proj}_2(x^{\mathsf{R}}), \mathsf{h}(\mathsf{proj}_1(x^{\mathsf{R}}, y^{\mathsf{R}})))\ \mathtt{in}\\
\quad\ \mathtt{out}(c, \mathsf{ok})\\
\quad \mathtt{else}\ \mathtt{event}(\mathsf{FailR}(x^{\mathsf{R}})); \mathtt{out}(\_, \mathtt{diff}[\mathsf{bad}_{\mathsf{L}}, \mathsf{bad}_{\mathsf{R}}])\\
\mathtt{else}\\
\quad \mathtt{event}(\mathsf{FailL}(x^{\mathsf{L}}));\\
\quad \mathtt{get}\ keys(\mathtt{diff}[\_, y^{\mathsf{R}}])\ \mathtt{st.}\ \mathsf{eq}(\mathsf{proj}_2(x^{\mathsf{R}}), \mathsf{h}(\mathsf{proj}_1(x^{\mathsf{R}}, y^{\mathsf{R}})))\ \mathtt{in}\\
\quad\ \mathtt{out}(\_, \mathtt{diff}[\mathsf{bad}_{\mathsf{L}}, \mathsf{bad}_{\mathsf{R}}])\\
\quad \mathtt{else}\ \mathtt{out}(c, \mathsf{error})
\end{array}
$$

$$
\begin{array}{c}
\mathtt{event}(\mathsf{FailL}(x^{\mathsf{L}})) \wedge \mathtt{event}(\mathsf{Inserted}(\mathtt{diff}[y^{\mathsf{L}}, y^{\mathsf{R}}])) \Rightarrow \mathsf{proj}_2(x^{\mathsf{L}}) \neq \mathsf{h}(\mathsf{proj}_1(x^{\mathsf{L}}), y^{\mathsf{L}})\\
\mathtt{event}(\mathsf{FailR}(x^{\mathsf{R}})) \wedge \mathtt{event}(\mathsf{Inserted}(\mathtt{diff}[y^{\mathsf{L}}, y^{\mathsf{R}}])) \Rightarrow \mathsf{proj}_2(x^{\mathsf{R}}) \neq \mathsf{h}(\mathsf{proj}_1(x^{\mathsf{R}}), y^{\mathsf{R}})
\end{array}
$$

Fig. 4. Transformed reader bi-process for Basic Hash

Lemma 3: Let B_0 be a separated closed bi-process such that $\mathcal{T}_1(B_0)\downarrow\uparrow$, and $T \in \overline{\mathtt{traces}}(\mathcal{T}_1(B_0))$. There exists $T_{\mathsf{fst}} \in \mathtt{traces}(\mathsf{fst}(B_0))$ such that $T_{\mathsf{fst}} \cong \mathsf{fst}(T)$. A similar result holds for snd.

Proof. (Sketch) The projected traces are simply obtained by de-duplicating the τ actions corresponding to get subprocesses. Again, the convergence hypothesis guarantees that we avoid $\mathtt{diff}[\mathsf{bad}_{\mathsf{L}}, \mathsf{bad}_{\mathsf{R}}]$. □

C. Refining the analysis in failure branches

We transform a *model*, i.e. a bi-process together with restrictions, axioms and lemmas, to a new model that is easier to verify by ProVerif. We first illustrate this transformation on a very simple toy example.

Example 15: Consider the following bi-process B:

$$
\begin{array}{l}
\mathtt{insert}\ tbl(\mathsf{ok});\\
\mathtt{get}\ tbl(x)\ \mathsf{st.}\ \mathsf{true}\ \mathsf{in}\ \mathtt{out}(c, \mathsf{ok})\\
\qquad\qquad\qquad \mathsf{else}\ \mathtt{out}(c, \mathtt{diff}[\mathsf{ok}_{\mathsf{L}}, \mathsf{ok}_{\mathsf{R}}])
\end{array}
$$

On this simple example, ProVerif returns "observational equivalence cannot be proved", whereas it is relatively easy to see that the else branch cannot be taken (as there is an element in the table), and thus equivalence holds. Applying the transformation that we will formally present next will lead to the following bi-process:

$$
\begin{array}{l}
\mathtt{event}(\mathsf{Inserted}(\mathsf{ok})); \mathtt{insert}\ tbl(\mathsf{ok});\\
\mathtt{get}\ tbl(x)\ \mathsf{st.}\ \mathsf{true}\ \mathsf{in}\ \mathtt{out}(c, \mathsf{ok})\\
\qquad\qquad\qquad\qquad \mathsf{else}\ \mathtt{event}(\mathsf{Fail}()); \mathtt{out}(c, \mathtt{diff}[\mathsf{ok}_{\mathsf{L}}, \mathsf{ok}_{\mathsf{R}}])
\end{array}
$$

together with the following axiom:

$$\mathtt{event}(\mathsf{Fail}()) \wedge \mathtt{event}(\mathsf{Inserted}(\mathtt{diff}[y^{\mathsf{L}}, y^{\mathsf{R}}])) \Rightarrow \mathsf{false}.$$

On this model, ProVerif is able to conclude that equivalence holds.

We now formally define this transformation.

Definition 8: We have that

$$(B, \mathcal{R}, \mathcal{A}x, \mathcal{L}) \rightsquigarrow_{\mathsf{else}} (B', \mathcal{R}, \mathcal{A}x \cup \{ax\}, \mathcal{L})$$

when there exist a context C, two terms M_1 and M_2 without diff operators, and two processes P and Q such that:

- B is of the form:

  $C[\mathtt{get}\ tbl(\mathtt{diff}[x^{\mathsf{L}}, x^{\mathsf{R}}])\ \mathtt{st.}\ \mathsf{eq}(M_1, M_2)\ \mathtt{in}\ P\ \mathtt{else}\ Q]$
- B' is of the form:

  $C[\mathtt{get}\ tbl(\mathtt{diff}[x^{\mathsf{L}}, x^{\mathsf{R}}])\mathtt{st.}\ \mathsf{eq}(M_1, M_2)\ \mathtt{in}\ P$
  $\qquad\qquad\qquad\qquad \mathtt{else}\ \mathtt{event}(\mathsf{Fail}(\vec{n}, \vec{y})).Q]$

  where $\vec{n} = \mathsf{names}(\{M_1, M_2\})$, $\vec{y} = \mathsf{vars}(\{M_1, M_2\}) \setminus \{x^{\mathsf{L}}, x^{\mathsf{R}}\}$, and all operations $\mathtt{insert}\ tbl(M)$ have been replaced by $\mathtt{event}(\mathsf{Inserted}(M)).\mathtt{insert}\ tbl(M)$
- ax is of the form:

  $$\mathtt{event}(\mathsf{Fail}(\vec{z}, \vec{y})) \wedge \mathtt{event}(\mathsf{Inserted}(\mathtt{diff}[x^{\mathsf{L}}, x^{\mathsf{R}}])) \Rightarrow (M_1 \neq M_2)\{\vec{n} \mapsto \vec{z}\}$$

  where Fail and Inserted are distinct events that do not occur in $(B, \mathcal{R}, \mathcal{A}x, \mathcal{L})$, and $\vec{z}$ is a vector of fresh variables of the same length as $\vec{n}$.

Intuitively, the transformation instruments the process with events in order to express, in the added axiom, that we can only execute the else branch of the bi-process if no value inserted in tbl satisfies $M_1 = M_2$. This makes sense even if x^{L} and x^{R} do not occur in $\mathsf{eq}(M_1, M_2)$: in that case we are simply saying that we can only visit the else branch if either $\mathsf{eq}(M_1, M_2)$ is false, or no value has even been inserted in the table.

The above intuition is not entirely exact, though. Indeed, the semantics of the axiom is that if at some point in the trace the events Fail and Inserted have happened (for some values of the free variables $\vec{z}, \vec{y}, x^{\mathsf{L}}, x^{\mathsf{R}}$), then $\mathsf{eq}(M_1, M_2)$ evaluates to true. The semantics of bi-processes enforces a weaker version of this statement, where the Inserted must happen before Fail. It is not possible to express such precise temporal properties in axioms with ProVerif but, as we shall see, we can live with this limitation.

The transformation can be iterated. Obviously, the insert events of several transformations relying on the same table can be merged. We justify next a single step of the transformation; the result extends naturally for iterations.

Proposition 2: Let $(B, \mathcal{R}, \mathcal{A}x, \mathcal{L})$, and $(B', \mathcal{R}, \mathcal{A}x', \mathcal{L})$ be two models such that $(B, \mathcal{R}, \mathcal{A}x, \mathcal{L}) \rightsquigarrow_{\mathsf{else}} (B', \mathcal{R}, \mathcal{A}x', \mathcal{L})$. We have that $\mathcal{A}x' = \mathcal{A}x \cup \{ax\}$ for some ax. Assuming that all insert instructions are performed in a phase that precedes the phase of the modified get instruction, we have:

$$\mathtt{traces}^{\mathcal{R}}(B') \vdash ax$$

Proof. Consider a bi-trace $T' \in \mathtt{traces}^{\mathcal{R}}(B')$, and assume that $T' \not\vdash ax$. Then, for some substitution θ, we have $\mathsf{Inserted}(\mathtt{diff}[x^{\mathsf{L}}, x^{\mathsf{R}}])\theta$ in the bi-trace, as well as $\mathsf{Fail}(\vec{z}, \vec{y})\theta$, and $(M_1 =_{\mathsf{E}} M_2)\{\vec{n} \mapsto \vec{z}\}\theta$ is true. In other words, the execution of the `get` action, for the values given by θ, has stepped to the else branch, when there existed a bi-term $\mathtt{diff}[x^{\mathsf{L}}, x^{\mathsf{R}}]\theta$ in the table that satisfies the condition. This contradicts the semantics of bi-processes. □

This result, together with Theorem 2 and Theorem 1, shows that if ProVerif concludes on the transformed model, then trace equivalence holds for the original one. More precisely, we have the following result.

Theorem 3: Let $(B_0, \emptyset, \mathcal{A}x, \mathcal{L})$ be a separated ProVerif model, and $(B', \emptyset, \mathcal{A}x', \mathcal{L})$ be the model obtained after applying our two transformation, i.e. a model such that $(\mathcal{T}_1(B_0), \emptyset, \mathcal{A}x, \mathcal{L}) \rightsquigarrow^*_{\mathsf{else}} (B', \emptyset, \mathcal{A}x', \mathcal{L})$. Moreover, we assume that:

- for all $\varrho \in \mathcal{A}x$, we have that $\mathtt{traces}(B_0) \vdash \varrho$;
- for all $\varrho \in \mathcal{A}x$, we have that $\mathtt{traces}(\mathcal{T}_1(B_0)) \vdash \varrho$;
- all insert instructions in B_0 are performed in a phase that precedes the phase of the get instructions transformed by $\rightsquigarrow_{\mathsf{else}}$ above; and
- ProVerif establishes equivalence on the resulting model $(B', \emptyset, \mathcal{A}x', \mathcal{L})$ i.e. returns "diff-equivalence is true" on this model.

We conclude that $\mathsf{fst}(B_0) \approx_t \mathsf{snd}(B_0)$.

Note that the two first items are there to justify the axioms already present in the original model (not the ones added by our transformation).

Proof. By Proposition 2, we have $\mathtt{traces}(B') \vdash \varrho'$ for any $\varrho' \in \mathcal{A}x' \setminus \mathcal{A}x$. By hypothesis, we have $\mathtt{traces}(\mathcal{T}_1(B_0)) \vdash \varrho$ for any $\varrho \in \mathcal{A}x$, and this is also true for B' since the events inserted in the last transformation are new. Moreover, we have no restriction here. By Theorem 1, we deduce that $\mathtt{traces}(B')\downarrow\uparrow$, and we also have $\mathtt{traces}(\mathcal{T}_1(B_0))\downarrow\uparrow$ since B' and $\mathcal{T}_1(B_0)$ are the same up to some events. We finally obtain $\mathsf{fst}(B_0) \approx_t \mathsf{snd}(B_0)$ by Theorem 2. □

V. Case studies

In this section, we explain how our approach can be used to analyse unlinkability on several existing protocols.

A. Implementation

In order to analyse our protocols of interest, we have developed a tool that implements the two transformations described in the previous section. This tool makes it easy to test our approach on various protocols, and avoids mistakes during transformations. It takes as input a ProVerif model containing a bi-process specifying the equivalence to be verified and returns another ProVerif file. If ProVerif concludes that diff-equivalence holds on the output file then our result allows one to conclude that the left and right processes written in the input file are in trace equivalence.

In practice, we implemented a ProVerif front-end tool to automatically apply our two transformations ($\approx 2k$ OCaml LoC). It takes as input a description of the protocol as presented in Example 7 and outputs a new ProVerif model in which the two parts of the bi-process have been desynchronised and the axioms corresponding to else branches have been added. ProVerif can then be ran on this transformed model to automatically prove the security of the protocol. The source code of our tool and material to reproduce results can be found at [19]. This includes ProVerif models for our case studies but also the generated files corresponding to their transformations: the reader may thus inspect the effect of our transformation without running our tool.

B. Results

We first study four examples: Basic Hash (used as a running example), Hash-Lock, Feldhofer, and a variant of the LAK protocol (LAK-v1). All these examples are described, e.g., in [2]. They are similar in the sense that they involve monotonic states, i.e. states that are never updated. On all these examples, it was already well-known that ProVerif is unable to conclude on the model encoding unlinkability relying on standard bi-process and diff-equivalence. Using our tool, we obtain a file on which ProVerif is able to conclude in a few seconds, thus we conclude that these protocols ensure unlinkability by Theorem 3. A detailed description of how our transformations work on the Basic Hash protocol is given in Section IV-A.

We then consider a fixed version of the OSK protocol, described as OSK-v2 in [2]. The protocol can be described informally as follows:

$$T \to R: \ \mathsf{g}(\mathsf{h}(k_T))$$
$$T \text{ updates } k_T \text{ with } \mathsf{h}(k_T)$$

Here, g and h are two hash functions. Each tag has a secret key k_T, whose initial value is stored in the reader's database. The tag updates its key by applying the h function symbol at each session on its current key. The reader expects a message of the form $\mathsf{g}(\mathsf{h}^n(x))$ for some database entry x (h^n means that h is applied n times). This check is modelled relying on a private destructor $\mathsf{retrievekey}(\cdot)$ allowing one to extract k from a term of the form $\mathsf{h}(n, k)$ used to model $\mathsf{h}^n(k)$. As usual in ProVerif, we encode the state of the tag using a private channel. To avoid false attacks, we rely on some axioms to improve precision in the manipulation of private channels. This is done following the approach developed in GSVerif [14] for trace-based properties. Extending the approach of [14] for equivalence properties is beyond the scope of this paper, but on this particular example, it is easy to be convinced that these axioms are valid, and thus our encoding correct. Despite this and the addition of a lemma to help regarding termination,

ProVerif is not able to establish that the equivalence holds. However, ProVerif concludes successfully on the transformed model (produced by our tool) hence unlinkability holds for OSK-v2 by Theorem 3.

C. Comparison with earlier work

In [2], Baelde *et. al.* develop an approach to tackle the same class of protocols as the one we consider here, and using the same definition for unlinkability. It is thus natural to compare in more details this work with the present one.

First, we note that all the results presented above are consistent with the security analyses performed in [2] where unlinkability has been established relying on three sufficient conditions proved using the tool Tamarin.

Second, the verification technique proposed by [2], based on sufficient conditions, is at a high-level independent of the verification tool, whereas our work is specific to ProVerif. Moreover, the sufficient conditions themselves can provide some insight as to what makes unlinkability hold, or fail.

However, in practice, verifying the sufficient conditions of [2] has proved very difficult, and the authors of that paper could only obtain semi-automated Tamarin proofs. In general, we thus reckon that this verification technique will require more effort than our direct approach. In each of the case studies of [2], two different models had to be designed to verify the three sufficient conditions. Each time, several lemmas have to be written corresponding to the various conditions (between two and four depending on the protocol) and, in some cases, several intermediate lemmas are needed to help Tamarin prove the conditions.

In particular, regarding OSK-v2, the security analysis conducted in [2] has only been possible thanks to several simplifications and using some modelling tricks. For instance, one of the sufficient conditions has been established discarding the reader role, and the attacker was not given the power to apply a hash function on some terms.

Overall, our approach brings an interesting complement to the technique of [2], which can significantly ease the verification effort when ProVerif is applicable.

VI. Conclusion

We have shown that unlinkability can be automatically proved in ProVerif, by applying two generic transformations on the standard bi-process expressing unlinkability. Our approach significantly improves over existing automated proofs, avoiding several over-approximations as well as manual work. It builds on several specific features of ProVerif. In particular, we make crucial use of extended bi-processes with diff operators on bound variables, that have seen only limited use so far (and which would not make sense in e.g. Tamarin). More generally, we make use of the recent addition of axioms, lemmas and restrictions in ProVerif, and we provide a better understanding of how restrictions can be used to prove equivalences.

It would be natural to extend our theoretical development as well as our tool to generalise our second transformation beyond equality tests between constructor terms. To deal with arbitrary terms D, a more involved transformation which essentially consists in computing the variants of an expression to get rid of destructor symbols will be necessary [16]. In light of the usefulness and simplicity of our approach, a promising perspective would be to integrate it by default in ProVerif, but this requires the addition of temporal constraints in correspondence queries. Meanwhile, this work introduces a new approach which is not specific to unlinkability. We develop in [19] an example (actually a toy protocol provided in [23]) which demonstrates that our approach can be used to prove *anonymity* properties. We think that a similar approach could have been used in [7] to overcome ProVerif limitations when manipulating lookup tables for anonymity properties. It appears as an alternative to the encoding used in [7] (and detailed in [6] - Section 5.4.3). Their encoding is based on the use of restrictions which have to be manipulated with a lot of care when analysing equivalence-based properties.

Our contribution in this paper, namely relaxing diff-equivalence to be able to prove unlinkability in ProVerif, does not address other difficult aspects that might arise in this task. Handling the tag's mutable state in the OSK-v2 case study has proved difficult, but there is hope that advances with GSVerif will soon solve this issue. It is also natural to consider protocols with state updates on the reader's side, i.e. updates of database entries. Although such operations are not directly supported in ProVerif, there is hope to model them naturally enough using restrictions – which calls for a (slight) generalisation of our results.

References

[1] M. Arapinis, T. Chothia, E. Ritter, and M. Ryan. Analysing unlinkability and anonymity using the applied pi calculus. In *Proc. 23rd IEEE Computer Security Foundations Symposium, CSF 2010, Edinburgh, United Kingdom, July 17-19, 2010*, pages 107–121. IEEE Computer Society, 2010.

[2] D. Baelde, S. Delaune, and S. Moreau. A method for proving unlinkability of stateful protocols. In *Proc. of the 33rd IEEE Computer Security Foundations Symposium (CSF'20)*. IEEE Computer Society Press, July 2020.

[3] D. Basin, R. Sasse, and J. Toro-Pozo. The EMV standard: Break, fix, verify. In *2021 IEEE Symposium on Security and Privacy (S&P'21)*, pages 1766–1781. IEEE, 2021.

[4] D. A. Basin, J. Dreier, and R. Sasse. Automated symbolic proofs of observational equivalence. In I. Ray, N. Li, and C. Kruegel, editors, *Proc. 22nd ACM SIGSAC Conference on Computer and Communications Security (CCS'15)*, pages 1144–1155. ACM, 2015.

[5] K. Bhargavan, B. Blanchet, and N. Kobeissi. Verified models and reference implementations for the TLS 1.3 standard candidate. In *2017 IEEE Symposium on Security and Privacy, (S&P'17), San Jose, CA, USA, May 22-26, 2017*, pages 483–502. IEEE Computer Society, 2017.

[6] K. Bhargavan, V. Cheval, and C. Wood. Handshake Privacy for TLS 1.3 - Technical report. Research report, Inria Paris ; Cloudflare, Mar. 2022.

[7] K. Bhargavan, V. Cheval, and C. Wood. A symbolic analysis of privacy for tls 1.3 with encrypted client hello. In *Proceedings of the 29th ACM Conference on Computer and Communications Security (CCS'22)*, Los Angeles, USA, Nov. 2022. ACM Press.

[8] B. Blanchet. An efficient cryptographic protocol verifier based on prolog rules. In *14th IEEE Computer Security Foundations Workshop (CSFW'01), 11-13 June 2001, Cape Breton, Nova Scotia, Canada*, pages 82–96, 2001.

[9] B. Blanchet, M. Abadi, and C. Fournet. Automated verification of selected equivalences for security protocols. *Journal of Logic and Algebraic Programming*, 75(1):3–51, 2008.

[10] B. Blanchet, V. Cheval, and V. Cortier. Proverif with lemmas, induction, fast subsumption, and much more. In *Proceedings of the 42nd IEEE Symposium on Security and Privacy (S&P'22)*. IEEE Computer Society Press, 2022.

[11] M. Brusò, K. Chatzikokolakis, and J. den Hartog. Formal verification of privacy for RFID systems. In *Proc. 23rd IEEE Computer Security Foundations Symposium, (CSF'10)*, pages 75–88. IEEE Computer Society, 2010.

[12] M. Brusò, K. Chatzikokolakis, S. Etalle, and J. den Hartog. Linking unlinkability. In C. Palamidessi and M. D. Ryan, editors, *Trustworthy Global Computing - 7th International Symposium, TGC 2012, Newcastle upon Tyne, UK, September 7-8, 2012, Revised Selected Papers*, volume 8191 of *LNCS*, pages 129–144. Springer, 2012.

[13] V. Cheval, V. Cortier, and S. Delaune. Deciding equivalence-based properties using constraint solving. *Theor. Comput. Sci.*, 492:1–39, 2013.

[14] V. Cheval, V. Cortier, and M. Turuani. A little more conversation, a little less action, a lot more satisfaction: Global states in proverif. In *Proceedings of the 31st IEEE Computer Security Foundations Symposium (CSF'18)*, pages 344–358, 2018.

[15] T. Chothia and V. Smirnov. A traceability attack against e-passports. In R. Sion, editor, *Financial Cryptography and Data Security, 14th International Conference, FC 2010, Tenerife, Canary Islands, Spain, January 25-28, 2010, Revised Selected Papers*, volume 6052 of *LNCS*, pages 20–34. Springer, 2010.

[16] H. Comon-Lundh and S. Delaune. The finite variant property: How to get rid of some algebraic properties. In *Proceedings of the 16th International Conference on Rewriting Techniques and Applications (RTA'05)*, volume 3467 of *LNCS*, pages 294–307, Nara, Japan, Apr. 2005. Springer.

[17] V. Cortier, A. Dallon, and S. Delaune. Efficiently deciding equivalence for standard primitives and phases. In J. López, J. Zhou, and M. Soriano, editors, *Proc. 23rd European Symposium on Research in Computer Security, (ESORICS'18)*, volume 11098 of *LNCS*, pages 491–511. Springer, 2018.

[18] C. Cremers, M. Horvat, J. Hoyland, S. Scott, and T. van der Merwe. A comprehensive symbolic analysis of TLS 1.3. In B. M. Thuraisingham, D. Evans, T. Malkin, and D. Xu, editors, *Proc. ACM SIGSAC Conference on Computer and Communications Security, (CCS'17)*, pages 1773–1788. ACM, 2017.

[19] A. Debant, S. Delaune, and D. Baelde. Proving Unlinkability using ProVerif through Desynchronized Bi-Processes. working paper or preprint available at https://hal.inria.fr/hal-03674979, May 2022.

[20] J. Dreier, L. Hirschi, S. Radomirovic, and R. Sasse. Automated unbounded verification of stateful cryptographic protocols with exclusive OR. In *Proc. 31st IEEE Computer Security Foundations Symposium (CSF'18)*, pages 359–373. IEEE Computer Society, 2018.

[21] I. Filimonov, R. Horne, S. Mauw, and Z. Smith. Breaking unlinkability of the ICAO 9303 standard for e-passports using bisimilarity. In K. Sako, S. A. Schneider, and P. Y. A. Ryan, editors, *Computer Security - ESORICS 2019 - 24th European Symposium on Research in Computer Security, Luxembourg, September 23-27, 2019, Proceedings, Part I*, volume 11735 of *Lecture Notes in Computer Science*, pages 577–594. Springer, 2019.

[22] L. Hirschi, D. Baelde, and S. Delaune. A method for verifying privacy-type properties: The unbounded case. In *IEEE Symposium on Security and Privacy, (S&P'16), San Jose, CA, USA, May 22-26, 2016*, pages 564–581. IEEE Computer Society, 2016.

[23] L. Hirschi, D. Baelde, and S. Delaune. A method for unbounded verification of privacy-type properties. *Journal of Computer Security*, 27(3):277–342, 2019.

[24] R. J. Horne, S. Mauw, and S. Yurkov. Unlinkability of an improved key agreement protocol for EMV 2nd gen payments. In *Proceedings of the 35th IEEE Computer Security Foundations Symposium (CSF'22)*, Haifa, Israel, Aug. 2022. IEEE Computer Society Press.

[25] S. Meier, B. Schmidt, C. Cremers, and D. A. Basin. The TAMARIN prover for the symbolic analysis of security protocols. In *Proc. 25th International Conference on Computer Aided Verification (CAV'13)*, pages 696–701, 2013.

[26] P. Paradzik and A. Derek. Conditional observational equivalence and off-line guessing attacks in multiset rewriting. In *Proceedings of the 35th IEEE Computer Security Foundations Symposium (CSF'22)*. IEEE Computer Society, 2022.

[27] T. van Deursen, S. Mauw, and S. Radomirovic. Untraceability of RFID protocols. In *Information Security Theory and Practices. Smart Devices, Convergence and Next Generation Networks, Second IFIP WG 11.2 International Workshop, WISTP 2008, Seville, Spain, May 13-16, 2008. Proceedings*, volume 5019 of *LNCS*, pages 1–15. Springer, 2008.

APPENDIX

A. On the soundness of ProVerif for non-separated bi-processes

Theorem 1 is established in [10] for separated bi-processes. We outline the argument next, referring to results in the technical report associated to that paper. In order to verify diff-equivalence, ProVerif first translates the definition of cryptographic primitives and the initial bi-process into a set of Horn clauses. It is shown in [10, Lemma 11] that if the bi-process admits non-convergent traces, then there exists a logical derivation of some bad atom using the Horn clauses. Then, [10, Theorem 3] justifies that the derivability of bad would be detected by the saturation step of ProVerif's algorithm.

The first step in this reasoning easily carries over to non-separated bi-processes. In fact, the generalised bi-processes made available by the `allowDiffPatterns` option are nothing more than a back-translation: mechanisms already available in ProVerif clauses are added to bi-processes, with a semantics that naturally follows the Horn clause semantics. We briefly explain this point below, referring the reader to [23, §5.1] for a more detailed discussion, and to [10, §4.3.2] for a full definition of the translation.

a) Translating bi-processes into Horn clauses: Clauses in ProVerif use an atomic proposition $\mathsf{att}'(m^{\mathsf{L}}, m^{\mathsf{R}})$ to express the fact that the attacker can compute m^{L} using some recipe after executing some trace with the left projection of the bi-process, and would compute m^{R} using the same recipe after executing the same trace against the right projection. In terms of bi-executions, this simply means that some recipe yields $\mathtt{diff}[m^{\mathsf{L}}, m^{\mathsf{R}}]$ when applied to the bi-frame obtained after executing the trace on the bi-process.

Consider for example the following separated bi-process, where cst is an arbitrary constant:

$$\mathsf{in}(c, x); \mathtt{if}\ x = \mathsf{cst}\ \mathtt{then}\ \mathsf{out}(c, x)\ \mathtt{else}\ \mathsf{out}(c, m)$$

The translation would notably produce the following clauses for that bi-process (we assume that m does not contain occurrences of x, but may contain diff operators):

$$\begin{aligned} \mathsf{att}'(\mathsf{cst}, \mathsf{cst}) &\Rightarrow \mathsf{att}'(\mathsf{cst}, \mathsf{cst}) \\ \mathsf{att}'(x^{\mathsf{L}}, x^{\mathsf{R}}) \wedge x^{\mathsf{L}} \neq \mathsf{cst} \wedge x^{\mathsf{R}} \neq \mathsf{cst} &\Rightarrow \mathsf{att}'(\mathsf{fst}(m), \mathsf{snd}(m)) \\ \mathsf{att}'(\mathsf{cst}, x^{\mathsf{R}}) \wedge x^{\mathsf{R}} \neq \mathsf{cst} &\Rightarrow \mathsf{bad} \end{aligned}$$

The first clause encodes the execution of our bi-process when the conditional succeeds on both projections; it brings no new knowledge to the attacker. The second one corresponds to the case where the conditional fails in both projections. The last one corresponds to a non-convergence case, where the conditional passes on the left but not on the right.

As is visible in this example, input variables are duplicated during the translation into their left and right counterparts. We

can back-translate this to bi-processes, obtaining for instance a bi-process that is rigorously equivalent to the previous one for ProVerif's translation (and which we make equivalent as a bi-process by providing it with the suitable semantics):

$$\begin{array}{l}\mathsf{in}(c, \mathtt{diff}[x^{\mathsf{L}}, x^{\mathsf{R}}]);\\ \mathtt{if}\ \mathtt{diff}[x^{\mathsf{L}}, x^{\mathsf{R}}] = \mathsf{cst}\ \mathtt{then}\ \mathsf{out}(c, \mathtt{diff}[x^{\mathsf{L}}, x^{\mathsf{R}}])\\ \mathtt{else}\ \mathsf{out}(c, m)\end{array}$$

More interestingly, we can modify our process to only perform the test on the left projection, as we do in our first transformation:

$$\begin{array}{l}\mathsf{in}(c, \mathtt{diff}[x^{\mathsf{L}}, x^{\mathsf{R}}]);\\ \mathtt{if}\ x^{\mathsf{L}} = \mathsf{cst}\ \mathtt{then}\ \mathsf{out}(c, \mathtt{diff}[x^{\mathsf{L}}, x^{\mathsf{R}}])\\ \mathtt{else}\ \mathsf{out}(c, m)\end{array}$$

The generated clauses would now become:

$$\begin{array}{rcl}\mathsf{att}'(\mathsf{cst}, x^{\mathsf{R}}) & \Rightarrow & \mathsf{att}'(\mathsf{cst}, x^{\mathsf{R}})\\ \mathsf{att}'(x^{\mathsf{L}}, x^{\mathsf{R}}) \wedge x^{\mathsf{L}} \neq \mathsf{cst} & \Rightarrow & \mathsf{att}'(\mathsf{fst}(m), \mathsf{snd}(m))\end{array}$$

Note that the clause deriving bad has disappeared since the test $x^{\mathsf{L}} = \mathsf{cst}$ cannot succeed on one projection only, since it does not contain diff operators anymore. Also note that the second clause above is simply obtained from the second clause of the first bi-process by dropping the atom corresponding to the right projection of the original conditional.

The fact that [10, Lemma 1] remains true for extended bi-processes simply stems from the fact that the semantics of extended bi-processes (as presented in this paper) are adequately captured by this simple extension of the translation process, which is the one currently implemented in ProVerif.

b) Soundness of the saturation procedure: As it is, [10, Theorem 3] relies on the assumption that clauses satisfy a so-called well-origination condition. The condition notably imposes that, if a clause concludes with $\mathsf{att}'(m, n)$, then the variables in m (resp. n) are introduced in some hypothesis of the clause. The definition is actually unclear in the case of clauses corresponding to biprocesses and, due to the complexity of the proofs, we could not determine how to clarify it.

The authors of [10] have indicated in personal communication that they expect to lift this technical condition completely in the future, which would almost immediately justify our use of extended biprocesses. Their goal is obviously to clarify and generalise the result, but also to justify an option that is currently available in ProVerif (under the name `maxHyps`) which allows the saturation procedure to drop hypotheses in large clauses in order to achieve termination more easily. This perspective would provide much more reasonable ways to justify ProVerif's soundness when used on (a class of) non-separated bi-processes. In fact, the justification of `maxHyps` would automatically allow the clauses resulting from the bi-processes used in our transformations: as in the example above, our bi-processes only exploit "unilateral conditionals", and their clauses are obtained from the clauses of standard bi-processes by dropping the unwanted projection of some conditionals.

B. Proofs of Section IV

Given a multiset $\mathcal{P}$ of bi-processes, we define $\mathcal{T}_1(\mathcal{P})$ the multiset of all transformed elements of $\mathcal{P}$. We extend this notion to bi-configurations as follows: $\mathcal{T}_1(K) = (\mathcal{T}_1(\mathcal{P}); \Phi; \mathcal{S}; i)$ when $K = (\mathcal{P}; \Phi; \mathcal{S}; i)$. Similarly, we extend the notion fst and snd to multiset of processes and to configurations as follows: $\mathsf{fst}(K) = (\mathsf{fst}(\mathcal{P}); \Phi_{\mathsf{fst}}; \mathcal{S}_{\mathsf{fst}}; i)$ when $K = (\mathcal{P}; \Phi; \mathcal{S}; i)$, and similarly for snd.

Lemma 2: Let B_0 be a separated closed bi-process such that $\mathcal{T}_1(B_0)\downarrow\uparrow$, and $T_{\mathsf{fst}} \in \mathtt{traces}(\mathsf{fst}(B_0))$. There exists $T \in \overline{\mathtt{traces}}(\mathcal{T}_1(B_0))$ such that $T_{\mathsf{fst}} \cong \mathsf{fst}(T)$. A similar result holds for snd.

Proof. We prove the result for fst, the case of snd being symmetric. We start by establishing the following claim.

Claim. Let K_0 be a separated bi-configuration, and $K = \mathcal{T}_1(K_0)$ such that $K\downarrow\uparrow$. If $\mathsf{fst}(K_0) \xrightarrow{\alpha} K'_L$ for some α and some K'_L, then there exist a separated bi-configuration K'_0 and K' such that $K'\downarrow\uparrow$, $K' = \mathcal{T}_1(K'_0)$, and $\mathsf{fst}(K'_0) = K'_L$. Moreover, either this step corresponds to the execution of a `get` instruction, and we have that $\alpha = \tau$, and $K \xrightarrow{\tau\tau} K'$ (with the execution of two `get` instructions in a row); otherwise we have that $K \xrightarrow{\alpha} K'$.

Note that the result stated above allows us to prove the statement in the lemma by induction on the length of the execution trace T_{fst}. Now, it remains to establish this claim. We do a case analysis depending on the rule used to perform $\mathsf{fst}(K_0) \xrightarrow{\alpha} K'_L$.

Case IN. In such a case, we have that:

- $K_0 = (\{i : \mathtt{in}(c, \mathtt{diff}[x^{\mathsf{L}}, x^{\mathsf{R}}]); P\} \cup \mathcal{P}; \Phi; \mathcal{S}; i)$;
- $\alpha = \mathtt{in}(c, R)$ for some recipe R such that $R\Phi_{\mathsf{fst}}\Downarrow =_{\mathsf{E}} M$, and $R\Phi_{\mathsf{snd}}\Downarrow =_{\mathsf{E}} M'$;
- $K'_L = (\{i : \mathsf{fst}(P)\{x^{\mathsf{L}} \mapsto M\}\} \cup \mathsf{fst}(\mathcal{P}); \Phi_{\mathsf{fst}}; \mathcal{S}_{\mathsf{fst}}; i)$; and
- $K = \mathcal{T}_1(K_0) = (\{i : \mathtt{in}(c, \mathtt{diff}[x^{\mathsf{L}}, x^{\mathsf{R}}]); \mathcal{T}_1(P)\} \cup \mathcal{T}_1(\mathcal{P}); \Phi; \mathcal{S}; i)$.

We consider the two following bi-configurations:

- $K'_0 = (\{i : P\{x^{\mathsf{L}} \mapsto M, x^{\mathsf{R}} \mapsto M'\}\} \cup \mathcal{P}; \Phi; \mathcal{S}; i)$; and
- $K' = (\{i : \mathcal{T}_1(P)\{x^{\mathsf{L}} \mapsto M, x^{\mathsf{R}} \mapsto M'\}\} \cup \mathcal{T}_1(\mathcal{P}); \Phi; \mathcal{S}; i)$.

Since K_0 is separated, we have that K'_0 is separated too, and we have that $\mathsf{fst}(K'_0) = K'_L$. Relying on the fact that our transformation is compatible with the substitution of diff-free terms, i.e., $\mathcal{T}(B\sigma) = \mathcal{T}(B)\sigma$ for all bi-process B and substitution σ which do not introduce terms with diff operators, we deduce that $\mathcal{T}_1(K'_0) = K'$. Then, it is easy to see that $K \xrightarrow{\mathtt{in}(c,R)} K'$, and thus $K'\downarrow\uparrow$.

Most of the cases can be done in a similar way. We focus now on the more interesting cases, i.e. the ones corresponding to `get` instructions.

Case GET-T. In such a case, we have that $\alpha = \tau$, and:

- $K_0 = (\{i : \mathtt{get}\ tbl(\mathtt{diff}[x^{\mathsf{L}}, x^{\mathsf{R}}])\ \mathtt{st}.\ D\ \mathtt{in}\ B_{\mathsf{then}}\ \mathtt{else}\ B_{\mathsf{else}}\} \cup \mathcal{P}; \Phi; \mathcal{S}; i)$;

- $K'_L = (\{i : \mathsf{fst}(B_{\mathsf{then}})\{x^{\mathsf{L}} \mapsto M_{\mathsf{L}}\}\} \cup \mathsf{fst}(\mathcal{P}); \Phi_{\mathsf{fst}}; \mathcal{S}_{\mathsf{fst}}; i)$ where M_{L} is such that
  $tbl(M_{\mathsf{L}}) \in \mathcal{S}_{\mathsf{fst}}$, and $\mathsf{fst}(D)\{x^{\mathsf{L}} \mapsto M_{\mathsf{L}}\}\Downarrow =_{\mathsf{E}} \mathsf{true}$;
- $K = \mathcal{T}_1(K_0) = (\{i : B'\} \cup \mathcal{T}_1(\mathcal{P}); \Phi; \mathcal{S}; i)$ where:

$$B' = \begin{cases} \mathtt{get}\ tbl(\mathtt{diff}[x^{\mathsf{L}}, \_])\ \mathtt{st.}\ \mathsf{fst}(D)\ \mathtt{in} \\ \quad \mathtt{get}\ tbl(\mathtt{diff}[\_, x^{\mathsf{R}}])\ \mathtt{st.}\ \mathsf{snd}(D)\ \mathtt{in} \\ \quad\ \mathcal{T}_1(B_{\mathsf{then}}) \\ \quad \mathtt{else}\ \mathsf{out}(\_, \mathtt{diff}[\mathsf{bad}_{\mathsf{L}}, \mathsf{bad}_{\mathsf{R}}]) \\ \mathtt{else} \\ \quad \mathtt{get}\ tbl(\mathtt{diff}[\_, x^{\mathsf{R}}])\ \mathtt{st.}\ \mathsf{snd}(D)\ \mathtt{in} \\ \quad\ \mathsf{out}(\_, \mathtt{diff}[\mathsf{bad}_{\mathsf{L}}, \mathsf{bad}_{\mathsf{R}}]) \\ \quad \mathtt{else}\ \mathcal{T}_1(B_{\mathsf{else}}) \end{cases}$$

Let M_{R} be such that $tbl(\mathtt{diff}[M_{\mathsf{L}}, M_{\mathsf{R}}]) \in \mathcal{S}$, and $\mathtt{diff}[M'_{\mathsf{L}}, M'_{\mathsf{R}}]$ be such that $tbl(\mathtt{diff}[M'_{\mathsf{L}}, M'_{\mathsf{R}}]) \in \mathcal{S}$ with $\mathsf{snd}(D)\{x^{\mathsf{R}} \mapsto M'_{\mathsf{R}}\}\Downarrow =_{\mathsf{E}} \mathsf{true}$. Note that the first element exists since we know that $tbl(M_{\mathsf{L}}) \in \mathcal{S}_{\mathsf{fst}}$. Regarding the second one, we know that it exists relying on our hypothesis $K\downarrow\uparrow$. Therefore, we consider the two following bi-configurations:

- $K'_0 = (\{i : B_{\mathsf{then}}\{x^{\mathsf{L}} \mapsto M_{\mathsf{L}}, x^{\mathsf{R}} \mapsto M'_{\mathsf{R}}\}\} \cup \mathcal{P}; \Phi; \mathcal{S}; i)$; and
- $K' = (\{i : \mathcal{T}_1(B_{\mathsf{then}})\{x^{\mathsf{L}} \mapsto M_{\mathsf{L}}, x^{\mathsf{R}} \mapsto M'_{\mathsf{R}}\}\} \cup \mathcal{T}_1(\mathcal{P}); \Phi; \mathcal{S}; i)$.

Since K_0 is separated, we have that K'_0 is separated too, and we have that $\mathsf{fst}(K'_0) = K'_L$. Relying on the fact that our transformation is compatible with substitution, we deduce that $\mathcal{T}_1(K'_0) = K'$. Then, it is easy to see that $K \xrightarrow{\tau\tau} K'$, and thus $K'\downarrow\uparrow$.

The case of the rule GET-E can be done in a similar way. This allows us to conclude. □

Lemma 3: Let B_0 be a separated closed bi-process such that $\mathcal{T}_1(B_0)\downarrow\uparrow$, and $T \in \overline{\mathtt{traces}}(\mathcal{T}_1(B_0))$. There exists $T_{\mathsf{fst}} \in \mathtt{traces}(\mathsf{fst}(B_0))$ such that $T_{\mathsf{fst}} \cong \mathsf{fst}(T)$. A similar result holds for snd.

Proof. We prove the result for fst, starting with the following claim.

Claim. Let K_0 be a separated bi-configuration, $K = \mathcal{T}_1(K_0)$ such that $K\downarrow\uparrow$. If $K \xrightarrow{\alpha} K'$ for some α and some K' with a rule different from GET-T and GET-E, then there exists K'_0 a separated bi-configuration such that $K' = \mathcal{T}_1(K'_0)$, and $\mathsf{fst}(K_0) \xrightarrow{\alpha} \mathsf{fst}(K'_0)$. A similar result holds when considering $K \xrightarrow{\tau\tau} K'$ corresponding to the execution of two `get` instructions in a row. In such a case, we have that $\mathsf{fst}(K_0) \xrightarrow{\tau} \mathsf{fst}(K'_0)$.

Note that the result stated above allows us to prove the statement in the lemma by induction on the length of the execution trace T. Now, it remains to establish this claim. We do a case analysis on the rule(s) used to perform $K \to K'$.

Case IN. In such a case, we have that:

- $K_0 = (\{i : \mathtt{in}(c, \mathtt{diff}[x^{\mathsf{L}}, x^{\mathsf{R}}]); P\} \cup \mathcal{P}; \Phi; \mathcal{S}; i)$;
- $K = (\{i : \mathtt{in}(c, \mathtt{diff}[x^{\mathsf{L}}, x^{\mathsf{R}}]); \mathcal{T}_1(P)\} \cup \mathcal{T}_1(\mathcal{P}); \Phi; \mathcal{S}; i)$;
- $\alpha = \mathtt{in}(c, R)$ for some recipe R such that $R\Phi_{\mathsf{fst}}\Downarrow =_{\mathsf{E}} M$, and $R\Phi_{\mathsf{snd}}\Downarrow =_{\mathsf{E}} M'$;
- $K' = (\{i : \mathcal{T}_1(P)\{x^{\mathsf{L}} \mapsto M, x^{\mathsf{R}} \mapsto M'\}\} \cup \mathcal{T}_1(\mathcal{P}); \Phi; \mathcal{S}; i)$.

We consider the following bi-configuration:

$$K'_0 = (\{i : P\{x^{\mathsf{L}} \mapsto M, x^{\mathsf{R}} \mapsto M'\}\} \cup \mathcal{P}; \Phi; \mathcal{S}; i)$$

Since K_0 is separated, we have that K'_0 is separated too. Since $K \xrightarrow{\alpha} K'$ and $K\downarrow\uparrow$, we have that $K'\downarrow\uparrow$. Relying on the fact that our transformation is compatible with substitution, we deduce that $K' = \mathcal{T}_1(K'_0)$. Then, it is easy to see that $\mathsf{fst}(K_0) \xrightarrow{\alpha} \mathsf{fst}(K'_0)$.

Most of the cases can be done in a similar way. We focus now on the most interesting ones, corresponding to the execution of two `get` instructions in a row.We simply consider the most interesting cases, corresponding to the execution of two `get` instructions in a row.

Case GET-T *followed by* GET-T. In such a case, we have that:

- $K_0 =$
  $(\{i : \mathtt{get}\ tbl(\mathtt{diff}[x^{\mathsf{L}}, x^{\mathsf{R}}])\ \mathtt{st.}\ D\ \mathtt{in}\ B_{\mathsf{then}}\ \mathtt{else}\ B_{\mathsf{else}}\}$
  $\cup \mathcal{P}; \Phi; \mathcal{S}; i)$;
- $K = \mathcal{T}_1(K_0) = (\{i : B'\} \cup \mathcal{T}_1(\mathcal{P}); \Phi; \mathcal{S}; i)$ where:

$$B' = \begin{cases} \mathtt{get}\ tbl(\mathtt{diff}[x^{\mathsf{L}}, \_])\ \mathtt{st.}\ \mathsf{fst}(D)\ \mathtt{in} \\ \quad \mathtt{get}\ tbl(\mathtt{diff}[\_, x^{\mathsf{R}}])\ \mathtt{st.}\ \mathsf{snd}(D)\ \mathtt{in} \\ \quad\ \mathcal{T}_1(B_{\mathsf{then}}) \\ \quad \mathtt{else}\ \mathsf{out}(\_, \mathtt{diff}[\mathsf{bad}_{\mathsf{L}}, \mathsf{bad}_{\mathsf{R}}]) \\ \mathtt{else} \\ \quad \mathtt{get}\ tbl(\mathtt{diff}[\_, x^{\mathsf{R}}])\ \mathtt{st.}\ \mathsf{snd}(D)\ \mathtt{in} \\ \quad\ \mathsf{out}(\_, \mathtt{diff}[\mathsf{bad}_{\mathsf{L}}, \mathsf{bad}_{\mathsf{R}}]) \\ \quad \mathtt{else}\ \mathcal{T}_1(B_{\mathsf{else}}) \end{cases}$$

- $K' = (\{i : \mathcal{T}_1(B_{\mathsf{then}})\{x^{\mathsf{L}} \mapsto M_{\mathsf{L}}, x^{\mathsf{R}} \mapsto M_{\mathsf{R}}\}\} \cup \mathcal{T}_1(\mathcal{P}); \Phi; \mathcal{S}; i)$.

We consider the following bi-configuration:

$$K'_0 = (\{i : B_{\mathsf{then}}\{x^{\mathsf{L}} \mapsto M_{\mathsf{L}}, x^{\mathsf{R}} \mapsto M_{\mathsf{R}}\}\} \cup \mathcal{P}; \Phi; \mathcal{S}; i).$$

Since K_0 is separated, we have that K'_0 is separated too. Since $K \xrightarrow{\tau\tau} K'$, and $K\downarrow\uparrow$, we have that $K'\downarrow\uparrow$. Relying on the fact that our transformation is compatible with substitution, we deduce that $K' = \mathcal{T}_1(K'_0)$. Then, it is easy to see that $\mathsf{fst}(K_0) \xrightarrow{\tau} \mathsf{fst}(K'_0)$.

Case GET-T *followed by* GET-E. In such a case, we have that

$$K' = (\{i : \mathtt{out}(c, \mathtt{diff}[\mathsf{bad}_{\mathsf{L}}, \mathsf{bad}_{\mathsf{R}}])\} \cup \mathcal{P}; \Phi; \mathcal{S}; i).$$

It is easy to see that K' diverges, and thus K diverges too, contradicting our hypothesis.

The other cases GET-E followed by GET-E, and GET-E followed by GET-T can be done in a similar way. This allows us to conclude. □

Theorem 2: Let B_0 be a separated closed bi-process such that $\mathcal{T}_1(B_0)\downarrow\uparrow$. We have that $\mathsf{fst}(B_0) \approx_t \mathsf{snd}(B_0)$.

Proof. Let $T_{\mathsf{fst}} \in \mathtt{traces}(\mathsf{fst}(B_0))$. By Lemma 2, we know that there exists $T \in \overline{\mathtt{traces}}(\mathcal{T}_1(B_0))$ such $T_{\mathsf{fst}} \cong \mathsf{fst}(T)$. Then, relying on Lemma 3 (considering the case snd), we know that there exists $T_{\mathsf{snd}} \in \mathtt{traces}(\mathsf{snd}(B_0))$ such that $T_{\mathsf{snd}} \cong \mathsf{snd}(T)$. Moreover, since $T \in \overline{\mathtt{traces}}(\mathcal{T}_1(B_0))$ and $\mathcal{T}_1(B_0)\downarrow\uparrow$, we have that $\mathsf{fst}(T) \cong \mathsf{snd}(T)$. This allows us to conclude that $T_{\mathsf{fst}} \cong \mathsf{fst}(T) \cong \mathsf{snd}(T) \cong T_{\mathsf{snd}}$. Hence, the result. □