A method for unbounded verification of privacy-type properties

Lucca Hirschi [a,*] David Baelde [b] Stéphanie Delaune [c,**]
[a] *Inria & LORIA, France*
E-mail: lucca.hirschi@inria.fr
[b] *LSV, ENS Paris-Saclay, CNRS & Université Paris-Saclay, France*
E-mail: david.baelde@lsv.fr
[c] *Univ Rennes, CNRS, IRISA, France*
E-mail: stephanie.delaune@irisa.fr

Abstract. In this paper, we consider the problem of verifying anonymity and unlinkability in the symbolic model, where protocols are represented as processes in a variant of the applied pi calculus, notably used in the ProVerif tool. Existing tools and techniques do not allow to verify directly these properties, expressed as behavioral equivalences. We propose a different approach: we design two conditions on protocols which are sufficient to ensure anonymity and unlinkability, and which can then be effectively checked automatically using ProVerif. Our two conditions correspond to two broad classes of attacks on unlinkability, *i.e.* data and control-flow leaks. This theoretical result is general enough that it applies to a wide class of protocols based on a variety of cryptographic primitives. In particular, using our tool, UKano, we provide the first formal security proofs of protocols such as BAC and PACE (e-passport), Hash-Lock (RFID authentication), etc. Our work has also lead to the discovery of new attacks, including one on the LAK protocol (RFID authentication) which was previously claimed to be unlinkable (in a weak sense).

Keywords: formal verification, security protocols, symbolic model, equivalence-based properties

1. Introduction

Security protocols aim at securing communications over various types of insecure networks (*e.g.* web, wireless devices) where dishonest users may listen to communications and interfere with them. A *secure communication* has a different meaning depending on the underlying application. It ranges from the confidentiality of data (medical files, secret keys, etc.) to, *e.g.* verifiability in electronic voting systems. Another example of a security notion is privacy. In this paper, we

[*] This work was conducted when Lucca Hirschi was working at LSV, ENS Paris-Saclay & Université Paris-Saclay, France and then at ETH Zurich, Switzerland.

[**] Corresponding author. E-mail: stephanie.delaune@irisa.fr. This work has received funding from the European Research Council (ERC) under the EU's Horizon 2020 research and innovation program (grant agreement No 714955-POPSTAR) and the ANR project SEQUOIA ANR-14-CE28-0030-01.

focus on two privacy-related properties, namely unlinkability (sometimes called untraceability), and anonymity. These two notions are informally defined in the ISO/IEC standard 15408 [2] as follows:

- Unlinkability aims at *ensuring that a user may make multiple uses of a service or resource without others being able to link these uses together.*
- Anonymity aims at *ensuring that a user may use a service or resource without disclosing its identity.*

Both are critical for instance for Radio-Frequency Identification Devices (RFID) and are thus extensively studied in that context (see, *e.g.* [52] for a survey of attacks on this type of protocols), but they are obviously not limited to it.

One extremely successful approach when designing and analyzing security protocols is the use of formal verification, *i.e.* the development of rigorous frameworks and techniques to analyze protocols. This approach has notably lead to the discovery of a flaw in the Single-Sign-On protocol used *e.g.* by Google Apps. It has been shown that a malicious application could very easily access to any other application (*e.g.* Gmail or Google Calendar) of their users [10]. This flaw has been found when analyzing the protocol using formal methods, abstracting messages by a term algebra and using the Avantssar validation platform. Another example is a flaw on vote-privacy discovered during the formal and manual analysis of an electronic voting protocol [34]. All these results have been obtained using *formal symbolic models*, where most of the cryptographic details are ignored using abstract structures. The techniques used in symbolic models have become mature and several tools for protocol verification are nowadays available, *e.g.* the Avantssar platform [11], the Tamarin prover [46], and the ProVerif tool [19].

Unfortunately, most of these results and tools focus on trace properties, that is, statements that something bad never occurs on any execution trace of a protocol. Secrecy and authentication are typical examples of trace properties: a data remains confidential if, for any execution, the attacker is not able to produce the data. However, privacy properties like unlinkability and anonymity are generally not defined as trace properties. Instead, they are usually defined as the fact that an observer cannot distinguish between two situations, which requires a notion of behavioural equivalence. Based on such a notion of equivalence, several definitions of privacy-type properties have been proposed (*e.g.* [7,23] for unlinkability, and [36,12] for vote-privacy). In this paper, we consider the well-established definitions of strong unlinkability and anonymity as defined in [7]. They have notably been used to establish privacy for various protocols either by hand or using ad hoc encodings (*e.g.* eHealth protocol [38], mobile telephony [8,9]). We provide a brief comparison with alternative definitions in Section 3.3.

Considering an unbounded number of sessions, the problem of deciding whether a protocol satisfies an equivalence property is undecidable even for a very limited fragment of protocols (see, *e.g.* [30]). Bounding the number of sessions suffices to retrieve decidability for standard primitives (see, *e.g.* [16,29]). However, analysing a protocol for a fixed (often low) number of sessions does not allow to prove security. Moreover, in the case of equivalence properties, existing tools scale badly and can only analyse protocols for a very limited number of sessions, typically 2 or 3. Another approach consists in implementing a procedure that is not guaranteed to terminate. This is in particular the case of ProVerif, a well-established tool for checking security of protocols. ProVerif is able to check a strong notion of equivalence (called diff-equivalence) between processes

that share the same structure. Despite recent improvements on diff-equivalence checking [28] intended to prove unlinkability of the BAC protocol (used in e-passport), ProVerif still cannot be used off-the-shelf to establish unlinkability properties, and therefore cannot conclude on most of the case studies presented in Section 6. Recently, similar approaches have been implemented in two other tools, namely Tamarin [15] and Maude–NPA [49]. They are based on a notion of diff-equivalence, and therefore suffer from the same drawbacks.

Our contribution. We believe that looking at trace equivalence of any pair of protocols is a too general problem and that much progress can be expected when one focuses on a few privacy goals and a class of protocols only (yet large and generic enough). We follow this different approach. We aim at proposing sufficient conditions that can be automatically checked, and that imply unlinkability and anonymity for a large class of security protocols. The success of our solution will be measured by confronting it to many real-world case studies.

More precisely, we identify a large class of 2-party protocols (simple else branches, arbitrary cryptographic primitives) and we devise two conditions called *frame opacity* and *well-authentication* that imply unlinkability and anonymity for an unbounded number of sessions. We show how these two conditions can be automatically checked using *e.g.* the ProVerif tool, and we provide tool support for that. Using our tool UKano (built on top of ProVerif), we have automatically analysed several protocols, among them the Basic Access Control (BAC) protocol as well as the Password Authenticated Connection Establishment (PACE) protocol that are both used in e-passports. We notably establish the first proof of unlinkability for ABCDH [5] and for the BAC protocol followed by the Passive Authentication (PA) and Active Authentication (AA) protocols. We also report on an attack that we found on the PACE protocol, and another one that we found on the LAK protocol [44] whereas it is claimed untraceable in [52]. It happens that our conditions are rather tight, and we believe that the overall methodology and proof method could be used for other classes of protocols and other privacy goals.

Our sufficient conditions. We now give an intuitive overview of our two sufficient conditions, namely *frame opacity* and *well-authentication*. In order to do this, assume that we want to design a mutual authentication protocol between a tag T and a reader R based on symmetric encryption, and we want this protocol to be unlinkable. We assume that k is a symmetric key shared between T and R.

A first attempt to design such a protocol is presented using Alice & Bob notation as follows (n_R is a fresh nonce):

$$1.\ R \to T : n_R$$
$$2.\ T \to R : \{n_R\}_k$$

This first attempt based on a challenge-response scheme is actually linkable. Indeed, an active attacker who systematically intercepts the nonce n_R and replaces it by a constant will be able to infer whether the same tag has been used in different sessions or not by comparing the answers he receives. Here, the tag is linkable because, for a certain behaviour (possibly malicious) of the attacker, some relations between messages leak information about the agents that are involved in the execution. Our first condition, namely *frame opacity*, actually checks that all outputted messages have only relations that only depend on what is already observable. Such relations can therefore not be exploited by the attacker to learn anything new about the involved agents.

Our second attempt takes the previous attack into account and randomises the tag's response and should achieve mutual authentication by requiring that the reader must answer to the challenge n_T. This protocol can be as follows:

$$1.\ R \to T : n_R$$
$$2.\ T \to R : \{n_R, n_T\}_k$$
$$3.\ R \to T : \{n_T\}_k$$

Here, Alice & Bob notation shows its limit. It does not specify how the reader and the tag are supposed to check that the messages they received are of the expected form, and how they should react when the messages are not well formed. This has to be precisely defined, since unlinkability depends on it. For instance, assume the tag does not check that the message he receives at step 3 contains the nonce n_T. We assume that it only checks that the received message is an encryption with its own key k, and it aborts the session otherwise. In such a flawed implementation, an active attacker can eavesdrop a message $\{n_T\}_k$ sent by R to a tag T, and try to inject this message at the third step of another session played by T'. The tag T' will react by either aborting or by continuing the execution of this protocol. Depending on the reaction of the tag, the attacker will be able to infer if T and T' are the same tag or not.

In this example, the attacker adopts a malicious behaviour that is not detected immediately by the tag who keeps executing the protocol. The fact that the tag passes successfully a conditional reveals crucial information about the agents that are involved in the execution. Our second condition, namely *well-authentication*, basically requires that when an execution deviates from the honest one, the agents that are involved cannot successfully pass a conditional, thus avoiding the leak of the binary information success/failure.

Our main theorem states that these two conditions, frame opacity and well-authentication, are actually sufficient to ensure both unlinkability and anonymity. This theorem is of interest as our two conditions are fundamentally simpler than the targeted properties: frame opacity can be expressed and established relying on diff-equivalence (without the aforementioned precision issue) and well-authentication is only a conjunction of reachability properties. In fact, they are both in the scope of existing automatic verification tools like ProVerif and Tamarin.

Some related work. The precision issue of diff-equivalence is well-known (acknowledged *e.g.* in [37,28,21,35]). So far, the main approach that has been developed to solve this issue consists in modifying the notion of diff-equivalence to get closer to trace equivalence. For instance, the swapping technique introduced in [37] and formally justified in [21] allows to relax constraints imposed by diff-equivalence in specific situations, namely in process algebras featuring a notion of phase, often used for modelling e-voting protocols. Besides, the limitation of the diff-equivalence w.r.t. conditional evaluations has been partially addressed in [28] by pushing away the evaluation of some conditionals into terms. Nevertheless, the problem remains in general and the limitation described above is not addressed by those works (incidentally, it is specifically acknowledged for the case of the BAC protocol in [28]). We have chosen to follow a novel approach in the same spirit as the one presented in [23]. However, [23] only considers a very restricted class of protocols (single-step protocols that only use hash functions), while we target more complex protocols.

This paper essentially subsumes the conference paper that has been published in 2016 [42]. Compared to that earlier work, we have greatly generalized the scope of our method and improved

its mechanization. First, we consider more protocols, including protocols where one party has a single identity (*e.g.* DAA) as well as protocols where sessions are executed sequentially instead of concurrently (*e.g.* e-passport scenarios). Second, we consider a much more general notion of frame opacity, which enables the analysis of more protocols. As a result of these two improvements, we could apply our method to more case studies (*e.g.* DAA, ABCDH).

Outline. In Section 2, we present our model inspired from the applied pi calculus as well as the class of protocols we consider. We then introduce in Section 3 the notion of trace equivalence that we then use to formally define the two privacy properties we study in this paper: unlinkability and anonymity. Our two conditions (frame opacity and well-authentication) and our main theorem are presented in Section 4. Finally, we discuss how to mechanize the verification of our conditions in Section 5 and present our case studies in Section 6, before concluding in Section 8. A detailed proof of our main result is provided in Appendix.

2. Modelling protocols

We model security protocols using a process algebra inspired from the applied pi calculus [4]. More specifically, we consider a calculus close to the one which is used in the ProVerif tool [20]. Participants are modeled as processes, and the communication between them is modeled by means of the exchange of messages that are represented by a term algebra.

2.1. Term algebra

We consider an infinite set $\mathcal{N}$ of *names* which are used to represent keys and nonces, and two infinite and disjoint sets of *variables*, denoted $\mathcal{X}$ and $\mathcal{W}$. Variables in $\mathcal{X}$ will typically be used to refer to unknown parts of messages expected by participants, while variables in $\mathcal{W}$, called *handles*, will be used to store messages learned by the attacker. We assume a *signature* Σ, *i.e.* a set of function symbols together with their arity, split into *constructor* and *destructor* symbols, *i.e.* $\Sigma = \Sigma_c \sqcup \Sigma_d$.

Given a signature $\mathcal{F}$ and a set of initial data A, we denote by $\mathcal{T}(\mathcal{F}, A)$ the set of terms built from elements of A by applying function symbols in $\mathcal{F}$. Terms of $\mathcal{T}(\Sigma_c, \mathcal{N} \cup \mathcal{X})$ will be called *constructor terms*. We note $vars(u)$ the set of variables that occur in a term u. A *message* is a constructor term u that is *ground*, *i.e.* such that $vars(u) = \varnothing$. We denote by $\overline{x}$, $\overline{n}$, $\overline{u}$, $\overline{t}$ a (possibly empty) sequence of variables, names, messages, and terms respectively. We also sometimes write them $(n_1, n_2, \ldots)$ or simply n (when the sequence is reduced to one element). Substitutions are denoted by σ, the domain of a substitution is written $dom(\sigma)$, and the application of a substitution σ to a term u is written $u\sigma$. The *positions* of a term are defined as usual.

Example 1. *Consider the following signature:*

$$\Sigma = \{\mathsf{enc},\ \mathsf{dec},\ \langle\,\rangle,\ \mathsf{proj}_1,\ \mathsf{proj}_2,\ \oplus,\ 0,\ \mathsf{eq},\ \mathsf{ok}\}.$$

The symbols enc *and* dec *of arity 2 represent symmetric encryption and decryption. Pairing is modeled using* $\langle\,\rangle$ *of arity 2, and projection functions* proj_1 *and* proj_2*, both of arity 1. The function symbol* $\oplus$ *of arity 2 and the constant* 0 *are used to model the exclusive or operator. Finally, we consider the symbol* eq *of arity 2 to model equality test, as well as the constant symbol* ok*. This signature is split into two parts:* $\Sigma_c = \{\mathsf{enc}, \langle\,\rangle, \oplus, 0, \mathsf{ok}\}$*, and* $\Sigma_d = \{\mathsf{dec}, \mathsf{proj}_1, \mathsf{proj}_2, \mathsf{eq}\}$.

As in the process calculus presented in [20], constructor terms are subject to an equational theory; this has proved very useful for modelling algebraic properties of cryptographic primitives (see *e.g.* [33] for a survey). Formally, we consider a congruence $=_{\mathsf{E}}$ on $\mathcal{T}(\Sigma_c, \mathcal{N} \cup \mathcal{X})$, generated from a set of equations E over $\mathcal{T}(\Sigma_c, \mathcal{X})$. Thus, this congruence relation is closed under substitution and renaming. We assume that it is not degenerate, *i.e.* there exist u, v such that $u \neq_{\mathsf{E}} v$.

Example 2. *To reflect the algebraic properties of the exclusive or operator, we may consider the equational theory generated by the following equations:*

$$x \oplus 0 = x \qquad x \oplus x = 0 \qquad x \oplus y = y \oplus x \qquad (x \oplus y) \oplus z = x \oplus (y \oplus z)$$

In such a case, we have that $\mathsf{enc}(a \oplus (b \oplus a), k) =_{\mathsf{E}} \mathsf{enc}(b, k)$.

We also give a meaning to destructor symbols through the notion of *computation relation*. As explained below, a computation relation may be derived from a rewriting system but we prefer to not commit to a specific construction, and therefore we introduce a generic notion of computation relation. Instead, we assume an arbitrary relation subject to a number of requirements, ensuring that the computation relation behaves naturally with respect to names, constructors, and the equational theory.

Definition 1. *A* computation relation *is a relation over* $\mathcal{T}(\Sigma, \mathcal{N}) \times \mathcal{T}(\Sigma_c, \mathcal{N})$*, denoted* $\Downarrow$*, that satisfies the following requirements:*

1. *if* $n \in \mathcal{N}$*, then* $n \Downarrow n$*;*
2. *if* $\mathsf{f} \in \Sigma_c$ *is a symbol of arity* k*, and* $t_1 \Downarrow u_1, \ldots, t_k \Downarrow u_k$*, then* $\mathsf{f}(t_1, \ldots, t_k) \Downarrow \mathsf{f}(u_1, \ldots, u_k)$*;*
3. *if* $t \Downarrow u$ *then* $t\rho \Downarrow u\rho$ *for any bijective renaming* ρ*;*
4. *if* t' *is a context built from* Σ *and* $\mathcal{N}$*,* $t \Downarrow u$*, and* $t'[u] \Downarrow v$ *then* $t'[t] \Downarrow v$*;*
5. *if* t' *is a context built from* Σ *and* $\mathcal{N}$*, and* t_1*,* t_2 *are constructor terms such that* $t_1 =_{\mathsf{E}} t_2$ *and* $t'[t_1] \Downarrow u_1$ *for some* u_1*, then* $t'[t_2] \Downarrow u_2$ *for some* u_2 *such that* $u_1 =_{\mathsf{E}} u_2$*;*
6. *if* $t \Downarrow u_1$ *then we have that* $t \Downarrow u_2$ *if, and only if,* $u_1 =_{\mathsf{E}} u_2$*.*

The last requirement expresses that the relation $\Downarrow$ associates, to any ground term t, at most one message up to the equational theory E. When no such message exists, we say that the *computation fails*; this is noted $t \not\Downarrow$. We may sometimes use directly $t\Downarrow$ as a message, when we know that the computation succeeds and the choice of representative is irrelevant.

A possible way to derive a computation relation is to consider an ordered set of rules of the form: $\mathsf{g}(u_1, \ldots, u_n) \to u$ where g is a destructor, and $u, u_1, \ldots, u_n \in \mathcal{T}(\Sigma_c, \mathcal{X})$. A ground term t can be rewritten into t' if there is a position p in t and a rule $\mathsf{g}(u_1, \ldots, u_n) \to u$ such that $t|_p = \mathsf{g}(v_1, \ldots, v_n)$ and $v_1 =_{\mathsf{E}} u_1\theta, \ldots, v_n =_{\mathsf{E}} u_n\theta$ for some substitution θ, and $t' = t[u\theta]_p$ (*i.e.* t in which the subterm at position p has been replaced by $u\theta$). Moreover, we assume that $u_1\theta, \ldots, u_n\theta$ as well as $u\theta$ are messages. In case there is more that one rule that can be applied at a given position p, we consider the one occurring first in the ordered set. We denote $\to^*$ the reflexive and transitive closure of $\to$, and $\Downarrow$ the relation induced by $\to$, *i.e.* $t \Downarrow u$ when $t \to^* u'$ and $u' =_{\mathsf{E}} u$.

Proving that an ordered set rewriting rules as defined above induces a computation relation is beyond the scope of this paper but the interested reader will find such a proof in [41].

Example 3. *The properties of symbols in* Σ_d *(Example 1) are reflected through the following rules:*

$$\mathsf{dec}(\mathsf{enc}(x, y), y) \to x \qquad \mathsf{eq}(x, x) \to \mathsf{ok} \qquad \mathsf{proj}_i(\langle x_1, x_2 \rangle) \to x_i \quad \text{for } i \in \{1, 2\}$$

This rewriting system induces a computation relation. For instance, we have that:

$$\mathsf{dec}(\mathsf{enc}(c, a \oplus b), b \oplus a) \Downarrow c, \qquad \mathsf{dec}(\mathsf{enc}(c, a \oplus b), b) \not\Downarrow, \text{ and} \qquad \mathsf{dec}(a, b) \oplus \mathsf{dec}(a, b) \not\Downarrow .$$

Example 4. *Ordered rewriting rules are expressive enough to define a destructor symbol* neq *such that* $\mathsf{neq}(u, v) \Downarrow \mathsf{yes}$ *if, and only if,* u *and* v *can be reduced to messages that are not equal modulo* E*. It suffices to consider* $\mathsf{neq}(x, x) \to \mathsf{no}$ *and* $\mathsf{neq}(x, y) \to \mathsf{yes}$ *(in this order) with* $\mathsf{yes}, \mathsf{no} \in \Sigma_c$*.*

For modelling purposes, we split the signature Σ into two parts, namely Σ_{pub} and Σ_{priv}. An attacker builds his own messages by applying public function symbols to terms he already knows and that are available through variables in $\mathcal{W}$. Formally, a computation done by the attacker is a *recipe*, *i.e.* a term in $\mathcal{T}(\Sigma_{\mathsf{pub}}, \mathcal{W})$. Recipes will be denoted by R, M, N. Note that, although we do not give the attacker the ability to generate fresh names to use in recipes, we obtain essentially the same capability by assuming an infinite supply of public constants in $\Sigma_c \cap \Sigma_{\mathsf{pub}}$.

2.2. Process algebra

We now define the syntax and semantics of the process algebra we use to model security protocols. We consider a calculus close to the one which is used in the ProVerif tool [20]. An important difference is that we only consider public channels. Our calculus also features, in addition to the usual replication (where an unbounded number of copies of a process are ran concurrently), a simple form of sequential composition and the associated *repetition* operation (where an unbounded number of copies of a process are ran sequentially, one after the other).

We consider a set $\mathcal{C}$ of channel names that are assumed to be public. Protocols are modeled through processes using the following grammar:

$P, Q := 0$	null	$\mid (P \mid Q)$	parallel	$\mid !P$	replication
$\mid \mathsf{in}(c, x).P$	input	$\mid \mathsf{new}\, \overline{n}.P$	restriction	$\mid ¡P$	repetition
$\mid \mathsf{out}(c, u).P$	output	$\mid \mathsf{let}\ \overline{x} = \overline{t}\ \mathsf{in}\ P\ \mathsf{else}\ Q$	evaluation	$\mid P; Q$	sequence

where $c \in \mathcal{C}$, $x \in \mathcal{X}$, $n \in \mathcal{N}$, $u \in \mathcal{T}(\Sigma_c, \mathcal{N} \cup \mathcal{X})$, and $\overline{x}$ and $\overline{t}$ are two sequences of the same length, respectively over variables ($\mathcal{X}$) and terms ($\mathcal{T}(\Sigma, \mathcal{N} \cup \mathcal{X})$).

We write $fv(P)$ for the set of *free variables* of P, *i.e.* the set of variables that are not bound by an input or a let construct. A process P is ground if $fv(P) = \varnothing$. Similarly, we write $fn(P)$ for the set of *free names* of P, *i.e.* the set of names that are not bound by a new construct.

Most constructs are standard in process calculi. The process 0 does nothing and we sometimes omit it. The process $\mathsf{in}(c, x).P$ expects a message m on channel c and then behaves like $P\{x \mapsto m\}$, *i.e.* P in which the (free) occurrences of x have been replaced by m. The process $\mathsf{out}(c, u).P$ emits u on channel c, and then behaves like P. The process $P \mid Q$ runs P and Q in parallel. The process $\mathsf{new}\, \overline{n}.P$ generates new names, binds it to $\overline{n}$, and continues as P.

The special construct $\mathsf{let}\ \overline{x} = \overline{t}\ \mathsf{in}\ P\ \mathsf{else}\ Q$ combines several standard constructions, allowing one to write computations and conditionals compactly. Such a process tries to evaluate the sequence of terms $\overline{t}$ and in case of success, *i.e.* when $\overline{t} \Downarrow \overline{u}$ for some messages $\overline{u}$, the process P in which $\overline{x}$ are replaced by $\overline{u}$ is executed; otherwise the process Q is executed. The goal of this construct is

to avoid nested let instructions to be able to define our class of protocols in a simple way later on. Note also that the let instruction together with the eq theory as defined in Example 3 can encode the usual conditional construction. Indeed, $\mathsf{let}\ x = \mathsf{eq}(u,v)\ \mathsf{in}\ P\ \mathsf{else}\ Q$ will execute P only if the computation succeeds on $\mathsf{eq}(u,v)$, that is only if $u \Downarrow u'$, $v \Downarrow v'$, and $u' =_{\mathsf{E}} v'$ for some messages u' and v'. For brevity, we sometimes omit $\mathsf{else}\ 0$.

The process $!P$ executes P an arbitrary number of times (in parallel). The last two constructs correspond to sequential compositions. The process $(P;Q)$ behaves like P at first, and after the complete execution of P it behaves like Q. The process $¡P$ executes P an arbitrary number of times in sequence, intuitively corresponding to $(P;P;P;\ldots)$. Such constructions are known to be problematic in process calculi. Our goal here is however quite modest: as will be visible in our operational semantics, our sequential composition is only meaningful for restricted processes. It could in fact be defined using recursion, but there is no point here to consider general recursion: our study is going to restrict to a simple class of protocols that would immediately exclude it.

Example 5. *We consider the RFID protocol due to Feldhofer* et al. *as described in [40] and which can be presented using Alice & Bob notation as follows:*

$$\begin{array}{l} 1.\ I \to R : n_I \\ 2.\ R \to I : \{n_I, n_R\}_k \\ 3.\ I \to R : \{n_R, n_I\}_k \end{array}$$

The protocol is between an initiator I (the reader) and a responder R (the tag) that share a symmetric key k. We consider the term algebra introduced in Example 3. The protocol is modelled by the parallel composition of P_I and P_R, corresponding respectively to the roles I and R.

$$P_{\mathsf{Fh}} \ := \ \mathsf{new}\, k.\ (\mathsf{new}\, n_I.P_I \ \mid \ \mathsf{new}\, n_R.P_R)$$

where P_I and P_R are defined as follows, with $u = \mathsf{dec}(x_1,k)$:

$$\begin{array}{ll} P_I := \mathsf{out}(c_I, n_I). & P_R := \mathsf{in}(c_R, y_1). \\ \quad \mathsf{in}(c_I, x_1). & \quad \mathsf{out}(c_R, \mathsf{enc}(\langle y_1, n_R\rangle, k)). \\ \quad \mathsf{let}\ x_2, x_3 = \mathsf{eq}(n_I, \mathsf{proj}_1(u)), \mathsf{proj}_2(u)\ \mathsf{in} & \quad \mathsf{in}(c_R, y_2). \\ \quad \mathsf{out}(c_I, \mathsf{enc}(\langle x_3, n_I\rangle, k)) & \quad \mathsf{let}\ y_3 = \mathsf{eq}(y_2, \mathsf{enc}(\langle n_R, y_1\rangle, k))\ \mathsf{in}\ 0 \end{array}$$

We may note that there are potentially several ways to implement the last reader's test. For instance, we may decide to replace the last line of the process P_R by $\mathsf{let}\ y_3 = \mathsf{eq}(\langle n_R, y_1\rangle, \mathsf{dec}(y_2,k))\ \mathsf{in}\ 0$. *This last check can also be simply removed. Alternatively, it could be followed by an observable action to make the outcome of the test manifest, e.g. the output of a public constant* open *in case of success and* close *in case of failure. This would be a reasonable model for many use cases, e.g. in access control scenarios a door may either open or remain close after the execution of the protocol.*

The operational semantics of processes is given by a labelled transition system over *configurations* (denoted by K) which are pairs $(\mathcal{P};\phi)$ where:

- $\mathcal{P}$ is a multiset of ground processes where null processes are implicitly removed;
- $\phi = \{w_1 \mapsto u_1, \ldots, w_n \mapsto u_n\}$ is a *frame*, *i.e.* a substitution where $w_1, \ldots, w_n$ are variables in $\mathcal{W}$, and $u_1, \ldots, u_n$ are messages.

$$\textsc{In}\quad (\mathsf{in}(c,x).P \cup \mathcal{P};\phi) \xrightarrow{\mathsf{in}(c,R)} (P\{x \mapsto u\} \cup \mathcal{P};\phi) \quad \text{where } R \text{ is a recipe such that } R\phi \Downarrow u \text{ for some message } u$$

$$\textsc{Out}\quad (\mathsf{out}(c,u).P \cup \mathcal{P};\phi) \xrightarrow{\mathsf{out}(c,w)} (P \cup \mathcal{P};\phi \cup \{w \mapsto u\}) \text{with } w \text{ a fresh variable in } \mathcal{W}$$

$$\textsc{New}\quad (\mathsf{new}\,\overline{n}.P \cup \mathcal{P};\phi) \xrightarrow{\tau} (P \cup \mathcal{P};\phi) \quad \text{where } \overline{n} \cap fn(\mathcal{P},\phi) = \varnothing$$

$$\textsc{Par}\quad (\{P_1 \mid P_2\} \cup \mathcal{P};\phi) \xrightarrow{\tau} (\{P_1,P_2\} \cup \mathcal{P};\phi)$$

$$\textsc{Then}\quad (\mathsf{let}\ \overline{x} = \overline{t}\ \mathsf{in}\ P\ \mathsf{else}\ Q \cup \mathcal{P};\phi) \xrightarrow{\tau_{\mathsf{then}}} (P\{\overline{x} \mapsto \overline{u}\} \cup \mathcal{P};\phi) \quad \text{when } \overline{t} \Downarrow \overline{u} \text{ for some } \overline{u}$$

$$\textsc{Else}\quad (\mathsf{let}\ \overline{x} = \overline{t}\ \mathsf{in}\ P\ \mathsf{else}\ Q \cup \mathcal{P};\phi) \xrightarrow{\tau_{\mathsf{else}}} (Q \cup \mathcal{P};\phi) \quad \text{when } t_i \not\Downarrow \text{ for some } t_i \in \overline{t}$$

$$\textsc{Rep}\text{–}!\quad (!P \cup \mathcal{P};\phi) \xrightarrow{\tau} (P \cup\, !P \cup \mathcal{P};\phi)$$

$$\textsc{Rep}\text{–}\mathsf{i}\quad (\mathsf{i}P \cup \mathcal{P};\phi) \xrightarrow{\tau} (\{P;\mathsf{i}P\} \cup \mathcal{P};\phi)$$

$$\textsc{Seq}\quad (P \cup \mathcal{P};\phi) \xrightarrow{\alpha} (P' \cup \mathcal{P};\phi') \quad \text{if } P \rightsquigarrow Q \text{ and } (Q \cup \mathcal{P};\phi) \xrightarrow{\alpha} (P' \cup \mathcal{P};\phi')$$

Fig. 1. Semantics for processes

$$\begin{aligned}
0;Q &\rightsquigarrow Q \\
(\mathsf{out}(c,u).P);Q &\rightsquigarrow \mathsf{out}(c,u).(P;Q) \\
(\mathsf{in}(c,x).P);Q &\rightsquigarrow \mathsf{in}(c,x).(P;Q) && \text{when } x \notin fv(Q) \\
(\mathsf{new}\,\overline{n}.P);Q &\rightsquigarrow \mathsf{new}\,\overline{n}.(P;Q) && \text{when } \overline{n} \cap fn(Q) = \varnothing \\
(\mathsf{let}\ \overline{x} = \overline{u}\ \mathsf{in}\ P'\ \mathsf{else}\ P'');Q &\rightsquigarrow \mathsf{let}\ \overline{x} = \overline{u}\ \mathsf{in}\ (P';Q)\ \mathsf{else}\ (P'';Q) && \text{when } \overline{x} \cap fv(Q) = \varnothing
\end{aligned}$$

Fig. 2. Sequence simplification rules

We often write $P \cup \mathcal{P}$ instead of $\{P\} \cup \mathcal{P}$. The terms in ϕ represent the messages that are known by the attacker. Given a configuration K, $\phi(K)$ denotes its second component. Sometimes, we consider processes as configurations: in such cases, the corresponding frame is the empty set $\varnothing$.

The operational semantics of a process is given by the relation $\xrightarrow{\alpha}$ defined as the least relation over configurations satisfying the rules in Figure 1. The rules are mostly standard and correspond to the intuitive meaning given previously. Rule In allows the attacker to send on channel c a message as soon as it is the result of a computation done by applying public function symbols on messages that are in his current knowledge. Rule Out corresponds to the output of a term: the corresponding term is added to the frame of the current configuration, which means that the attacker gains access to it. Rule New corresponds to the generation of a fresh name. As is standard, the bound names $\overline{n}$ can be renamed to achieve freshness so that the rule can always fire. The Par rule simply splits parallel compositions. The Then and Else rules correspond to the evaluation of a sequence of terms $\overline{t} = t_1, \ldots, t_n$; if this succeeds, *i.e.* if there exist messages $u_1, \ldots u_n$ such that $t_1 \Downarrow u_1, \ldots t_n \Downarrow u_n$ then variables $\overline{x}$ are bound to those messages, and P is executed; otherwise the process will continue with Q. Rules Rep–! and Rep–i unfold replication and repetition operators. The latter gives rise to a sequential composition, whose execution will have to rely, via the Seq rule, on the simplification rules of Figure 2. These rules only support

a limited set of operators, hence a sequence $P;Q$ is only executable for a restricted class of processes P, notably excluding parallel compositions. This is not an issue for our simple needs; our purpose here is not to define a general (and notoriously problematic) notion of sequence.

We note that our semantics enjoys some expected properties. Reduction is stable by bijective renaming, thanks to Definition 1, item 3: if $K_1 \xrightarrow{\alpha} K_2$ then $K_1\rho \xrightarrow{\alpha} K_2\rho$ where ρ is a bijection over $\mathcal{N}$, applied here to processes and frames. It is also compatible with our equational theory, thanks to Definition 1, items 5 and 6: if $K_1 \xrightarrow{\alpha} K_2$ then $K_1' \xrightarrow{\alpha} K_2'$ for any $K_1' =_{\mathsf{E}} K_1$ and $K_2' =_{\mathsf{E}} K_2$. By Definition 1, item 6, we also have that

$$(\mathcal{P};\phi) \xrightarrow{\mathsf{in}(c,R)} (\mathcal{P}';\phi) \text{ and } R\phi\Downarrow =_{\mathsf{E}} R'\phi\Downarrow \text{ yield } (\mathcal{P};\phi) \xrightarrow{\mathsf{in}(c,R')} (\mathcal{P}';\phi) \text{ (modulo } \mathsf{E}).$$

As usual, the relation $\xrightarrow{\alpha_1 \dots \alpha_n}$ between configurations (where $\alpha_1 \dots \alpha_n$ is a *trace*, *i.e.* a sequence of actions) is defined as the (labelled) reflexive and transitive closure of $\xrightarrow{\alpha}$.

Definition 2. *Input and output actions are called* observable*, while all other are unobservable. Given a trace* tr *we define* $\mathsf{obs}(\mathsf{tr})$ *to be the sub-sequence of observable actions of* tr*.*

We generally refer to τ, τ_{then} and τ_{else} as *unobservable actions*. It will become clear later on why we make a distinction when a process evolves using Then or Else.

Example 6. *Continuing Example 5. We have that* $P_{\mathsf{Fh}} \xrightarrow{\mathsf{tr}} (\varnothing;\phi_0)$ *where:*

- $\mathsf{tr} = \tau.\tau.\tau.\tau.\mathsf{out}(c_I,w_1).\mathsf{in}(c_R,w_1).\mathsf{out}(c_R,w_2).\mathsf{in}(c_I,w_2).\tau_{\mathsf{then}}.\mathsf{out}(c_I,w_3).\mathsf{in}(c_R,w_3).\tau_{\mathsf{then}}$*;*
- $\phi_0 = \{w_1 \mapsto n_I',\ w_2 \mapsto \mathsf{enc}(\langle n_I', n_R'\rangle, k'),\ w_3 \mapsto \mathsf{enc}(\langle n_R', n_I'\rangle, k')\}$*.*

The names k'*,* n_I' *and* n_R' *are fresh names. Actually, this execution corresponds to a normal execution of one session of the protocol.*

2.3. A generic class of two-party protocols

We aim to propose sufficient conditions to ensure unlinkability and anonymity for a generic class of two-party protocols. In this section, we define formally the class of protocols we are interested in.

Roles. We consider two-party protocols that are therefore made of two roles called the initiator and responder role respectively. We assume a set $\mathcal{L}$ of labels that will be used to name output actions in these roles, allowing us to identify outputs that are performed by a same syntactic output action. These labels have no effect on the semantics.

Definition 3. *An* initiator role *is a process that is obtained using the following grammar:*

$$P_I := 0 \ \mid\ \ell : \mathsf{out}(c,u).P_R$$

where $c \in \mathcal{C}$*,* $u \in \mathcal{T}(\Sigma_c, \mathcal{N} \cup \mathcal{X})$*,* $\ell \in \mathcal{L}$*, and* P_R *is obtained from the grammar of* responder roles*:*

$$P_R := 0 \quad \mid \mathsf{in}(c,y).\mathsf{let}\ \overline{x} = \overline{t}\ \mathsf{in}\ P_I\ \mathsf{else}\ P_{\mathsf{fail}} \qquad \text{with } P_{\mathsf{fail}} = 0 \mid \ell : \mathsf{out}(c',u')$$

where $c, c' \in \mathcal{C}$*,* $y \in \mathcal{X}$*,* $\overline{x}$ *(resp.* $\overline{t}$*) is a sequence of variables in* $\mathcal{X}$ *(resp. terms in* $\mathcal{T}(\Sigma, \mathcal{N} \cup \mathcal{X})$*),* $u' \in \mathcal{T}(\Sigma_c, \mathcal{N} \cup \mathcal{X})$*, and* $\ell \in \mathcal{L}$*.*

Moreover, an initiator (resp. responder) role is assumed to be ground, i.e., contain no free variable, though it may contain free names.

Intuitively, a role describes the actions performed by an agent. A responder role consists of waiting for an input and, depending on the outcome of a number of tests, the process will continue by sending a message and possibly waiting for another input, or stop possibly outputting an error message. An initiator behaves similarly but begins with an output. The grammar forces to add a conditional after each input. This is not a real restriction as it is always possible to add trivial conditionals with empty $\overline{x}$, and $\overline{t}$.

Example 7. *Continuing our running example, P_I (resp. P_R) as defined in Example 5 is an initiator (resp. responder) role, up to the addition of a trivial conditional in role P_R and distinct labels ℓ_1, ℓ_2, and ℓ_3 to decorate output actions.*

Then, a protocol notably consists of an initiator role and a responder role that can interact together producing an *honest trace*. Intuitively, an honest trace is a trace in which the attacker does not really interfere, and that allows the execution to progress without going into an else branch, which would intuitively correspond to a way to abort the protocol.

Definition 4. *A trace* tr (i.e. *a sequence of actions) is* honest *for a frame ϕ if $\tau_{\mathsf{else}} \notin \mathsf{tr}$ and $\mathsf{obs}(\mathsf{tr})$ is of the form $\mathsf{out}(\_, w_0).\mathsf{in}(\_, R_0).\mathsf{out}(\_, w_1).\mathsf{in}(\_, R_1)\ldots.$ for arbitrary channel names, and such that $R_i\phi{\Downarrow}w_i\phi$ for any action $\mathsf{in}(\_, R_i)$ occurring in tr.*

Identities and sessions. In addition to the pair of initiator and responder roles, more information is needed in order to meaningfully define a protocol. Among the names that occur in these two roles, we need to distinguish those that correspond to identity-specific, long-term data (*e.g.* k from Example 5), called *identity parameters* and denoted $\overline{k}$ below, and those which shall be freshly generated at each session (*e.g.* n_I, n_R from Example 5), called *session parameters* and denoted $\overline{n}_I$ and $\overline{n}_R$ below. We will require that any free name of roles must be either a session or an identity parameter. When necessary, we model long-term data that is not identity-specific (*i.e.* uniform for all agents) as private constants (*i.e.* terms in $\Sigma_c \cap \Sigma_{\mathsf{priv}}$).Depending on the protocol to be modelled, we shall see that either both the initiator and the responder or only one of those roles have identity parameters. The former case arises for protocols that involves different identities for each party while the latter concerns protocols whose only one party can be instantiated by different agents (see Examples 10, 11 below for a more detailed discussion).

We also need to know whether sessions (with the same identity parameters) can be executed *concurrently* or only *sequentially*. For instance, let us assume that the Feldhofer protocol is used in an access control scenario where all tags that are distributed to users have pairwise distinct identities. Assuming that tags cannot be cloned, it is probably more realistic to consider that a tag can be involved in at most one session at a particular time, *i.e.* a tag may run different sessions but only in sequence. Such a situation will also occur in the e-passport application where a same passport cannot be involved in two different sessions of the BAC protocol (resp. PACE protocol) concurrently. This is the purpose of the components $\dagger_I$ and $\dagger_R$ in the following definition. When one role has no identity parameter, we also consider both cases: whether the only identity instantiating this role may have concurrent sessions or only sequential sessions. Moreover, we require that the process P_Π which models a single session of the protocol can produce an honest trace.

Definition 5. *A protocol Π is a tuple $(\overline{k}, \overline{n}_I, \overline{n}_R, \dagger_I, \dagger_R, \mathcal{I}, \mathcal{R})$ where $\overline{k}$, $\overline{n}_I$, $\overline{n}_R$ are three disjoint sets of names, $\mathcal{I}$ (resp. $\mathcal{R}$) is an initiator (resp. responder) role such that $fn(\mathcal{I}) \subseteq \overline{k} \sqcup \overline{n}_I$, $fn(\mathcal{R}) \subseteq$*

$\overline{k} \sqcup \overline{n}_R$, *and* $\dagger_I, \dagger_R \in \{!, \mathsf{i}\}$. *Labels of* $\mathcal{I}$ *and* $\mathcal{R}$ *must be pairwise distinct. Names* $\overline{k}$ *(resp.* $\overline{n}_I \sqcup \overline{n}_R$*) are called* identity parameters *(resp.* session parameters*).*

Given a protocol Π*, we define* $P_\Pi := \mathsf{new}\,\overline{k}.(\mathsf{new}\,\overline{n}_I.\mathcal{I} \mid \mathsf{new}\,\overline{n}_R.\mathcal{R})$ *and we assume that* $P_\Pi \xrightarrow{\mathsf{tr}_h} (\varnothing; \phi_h)$ *for some frame* ϕ_h *and some trace* tr_h *that is honest for* ϕ_h*.*

Given a protocol Π, we also associate another process $\mathcal{M}_\Pi$ that represents the situation where the protocol can be executed by an arbitrary number of identities, with the possibility of executing an arbitrary number of sessions for a given identity. The formal definition differs slightly depending on whether identity parameters occur in both roles or only in the role $\mathcal{I}$ (resp. $\mathcal{R}$).

Definition 6. *Given a protocol* $\Pi = (\overline{k}, \overline{n}_I, \overline{n}_R, \dagger_I, \dagger_R, \mathcal{I}, \mathcal{R})$*, the process* $\mathcal{M}_\Pi$ *is defined as follows:*

- *If* $\overline{k} \cap fn(\mathcal{I}) \neq \varnothing$ *and* $\overline{k} \cap fn(\mathcal{R}) \neq \varnothing$*, then* $\mathcal{M}_\Pi = !\,\mathsf{new}\,\overline{k}.(\dagger_I\,\mathsf{new}\,\overline{n}_I.\mathcal{I} \mid \dagger_R\,\mathsf{new}\,\overline{n}_R.\mathcal{R})$*;*
- *If* $\overline{k} \cap fn(\mathcal{I}) = \varnothing$ *and* $\overline{k} \cap fn(\mathcal{R}) \neq \varnothing$*, then* $\mathcal{M}_\Pi = \dagger_I\,\mathsf{new}\,\overline{n}_I.\mathcal{I} \mid !\,\mathsf{new}\,\overline{k}.\ \dagger_R\,\mathsf{new}\,\overline{n}_R.\mathcal{R}$*.*

For the sake of simplicity, in case identity parameters only occur in one role, we assume that this role is the responder role. The omitted case where identity parameters occur only in the initiator role is very much similar. In fact, swapping the initiator and responder roles can also be formally achieved by adding an exchange of a fresh nonce at the beginning of the protocol under consideration. Note that the case where both $\overline{k} \cap fn(\mathcal{I})$ and $\overline{k} \cap fn(\mathcal{R})$ are empty means that no identity parameters are involved and therefore there is no issue regarding privacy. As expected, in such a situation, our definitions of unlinkability and anonymity (see Definition 12 and Definition 10) will be trivially satisfied.

Example 8. *Let* $\Pi_{\mathsf{Fh}} = (k, n_I, n_R, !, !, P_I, P_R)$ *with* P_I *and* P_R *as defined in Example 5 (up to the addition of a trivial conditional). Let* $P_{\mathsf{Fh}} = \mathsf{new}\,k.(\mathsf{new}\,n_I.P_I \mid \mathsf{new}\,n_R.P_R)$*,* $\mathsf{tr}_h = \mathsf{tr}$ *(up to the addition of an action* τ_{then}*), and* $\phi_h = \phi_0$ *as defined in Example 6. They satisfy the requirements stated in Definition 5, and therefore* Π_{Fh} *is a protocol according to our definition. For this protocol, the identity parameter* k *occurs both in the role* P_I *and* P_R*, and therefore we have that* $\mathcal{M}_{\Pi_{\mathsf{Fh}}} = !\,\mathsf{new}\,k.(!\,\mathsf{new}\,n_I.P_I \mid !\,\mathsf{new}\,n_R.P_R)$*.*

Example 9. *In order to illustrate our method and the use of the repetition operator, we introduce a toy protocol which, as we shall see later, satisfies unlinkability only when sessions of the initiator role are executed sequentially. Using Alice & Bob notation, this protocol can be described as follows:*

$$\begin{array}{l} 1.\ T \to R : n_T \\ 2.\ R \to T : n_R \\ 3.\ T \to R : \mathsf{mac}(\langle n_R, n_T \rangle, k) \\ 4.\ R \to T : \mathsf{mac}(\langle n_T, n_R \rangle, k) \end{array}$$

The protocol is between a tag T *(the initiator) and a reader* R *(the responder) which share a symmetric key* k*. To avoid an obvious reflection attack (where* $n_T = n_R$*), we assume that the tag systematically checks that the first message it receives is not the one he sent initially.*

To formalise such a protocol, we consider

$$\Sigma_c = \{\mathsf{mac}, \langle\ \rangle, \mathsf{ok}, \mathsf{yes}, \mathsf{no}\}, \textit{ and } \Sigma_d = \{\mathsf{proj}_1, \mathsf{proj}_2, \mathsf{eq}, \mathsf{neq}\}.$$

All symbols of the signature are public: ok, yes, *and* no *have arity* 0*;* proj_1 *and* proj_2 *have arity* 1*; other function symbols have arity* 2*. The destructors* $\mathsf{proj}_1, \mathsf{proj}_2, \mathsf{eq}$ *and* neq *are defined as in Examples 3 and 4. The symbol* mac *will be used to model message authentication code.*

The processes modelling the initiator and the responder roles are as follows:

$$
\begin{array}{ll}
P'_T := \mathsf{out}(c_T, n_T).\\
\quad \mathsf{in}(c_T, x_1).\\
\quad \mathsf{let}\ x_{\mathsf{test}} = \mathsf{eq}(\mathsf{yes}, \mathsf{neq}(x_1, n_T))\ \mathsf{in}\\
\quad \mathsf{out}(c_T, \mathsf{mac}(\langle x_1, n_T\rangle, k)).\mathsf{in}(c_T, x_2).\\
\quad \mathsf{let}\ x'_{\mathsf{test}} = \mathsf{eq}(x_2, \mathsf{mac}(\langle n_T, x_1\rangle, k))\ \mathsf{in}\ 0
&
P'_R := \mathsf{in}(c_R, y_1).\\
\quad \mathsf{out}(c_R, n_R).\\
\quad \mathsf{in}(c_R, y_2).\\
\quad \mathsf{let}\ y_{\mathsf{test}} = \mathsf{eq}(y_2, \mathsf{mac}(\langle n_R, y_1\rangle, k))\ \mathsf{in}\\
\quad \mathsf{out}(c_R, \mathsf{mac}(\langle y_1, n_R\rangle, k)).\ 0
\end{array}
$$

We may note that different choices can be made. For instance, we may decide to replace the process 0 *occurring in* P'_T *with* $\mathsf{out}(c_T, \mathsf{ok})$ *to make the outcome of the test manifest. The tuple* $\Pi^{\mathsf{i}}_{\mathsf{Toy}} = (k, (n_T), (n_R), \mathsf{i}, !, P'_T, P'_R)$ *is a protocol according to Definition 5. For this protocol, we have again that* k *occurs both in the roles* P'_T *and* P'_R*, and therefore:*

$$\mathcal{M}_{\Pi^{\mathsf{i}}_{\mathsf{Toy}}} = !\,\mathsf{new}\,k.\big(\,\mathsf{i}\,\mathsf{new}\,n_T.P'_T \mid !\,\mathsf{new}\,n_R.P'_R\big).$$

As a last example, we will consider one for which identity parameters only occur in one role. This example can be seen as a simplified version of the Direct Anonymous Attestation (DAA) sign protocol that will be detailed in Section 6.

Example 10. *We consider a simplified version of the protocol DAA sign (adapted from [50]). Note that a comprehensive analysis of the protocol DAA sign (as well as the protocol DAA join) will be conducted in Section 6. Before describing the protocol itself, we introduce the term algebra that will allow us to model the signature and zero knowledge proofs used in that protocol. We consider:*

- $\Sigma_c = \{\mathsf{sign}, \mathsf{zk}, \mathsf{pk}, \langle\,\rangle, \mathsf{tuple}, \mathsf{ok}, \mathsf{sk}_{\mathsf{I}}, \mathsf{error}\}$*, and*
- $\Sigma_d = \{\mathsf{check}_{\mathsf{sign}}, \mathsf{check}_{\mathsf{zk}}, \mathsf{public}_{\mathsf{zk}}, \mathsf{proj}_1, \mathsf{proj}_2, \mathsf{proj}^4_1, \mathsf{proj}^4_2, \mathsf{proj}^4_3, \mathsf{proj}^4_4\}$.

We consider the computation relation induced by the empty set of equations, and the rules:

$$
\begin{array}{ll}
\mathsf{check}_{\mathsf{sign}}(\mathsf{sign}(x, y), \mathsf{pk}(y)) \to x & \mathsf{proj}_i(\langle y_1, y_2\rangle) \to y_i \quad i \in \{1, 2\}\\
\mathsf{check}_{\mathsf{zk}}(\mathsf{zk}(\mathsf{sign}(\langle x_k, x_{id}\rangle, z_{\mathsf{sk}}), x_k, \mathsf{tuple}(y_1, y_2, y_3, \mathsf{pk}(z_{\mathsf{sk}})))) \to \mathsf{ok} & \\
\mathsf{public}_{\mathsf{zk}}(\mathsf{zk}(x, y, z)) \to z & \mathsf{proj}^4_i(\mathsf{tuple}(y_1, y_2, y_3, y_4)) \to y_i \quad i \in \{1, 2, 3, 4\}
\end{array}
$$

The protocol is between a client C *(the responder) and a verifier* V *(the initiator). The client is willing to sign a message* m *using a credential issued by some issuer and then he has to convince* V *that the latter signature is genuine. The client* C *has a long-term secret key* k_C*, an identity* id_C*, and some credential* $cred_C = \mathsf{sign}(\langle k_C, id_C\rangle, \mathsf{sk}_{\mathsf{I}})$ *issued by some issuer* I *having* sk_{I} *as a long-term signature key. Such a credential would be typically obtained once and for all through a protocol similar to DAA join. We give below an Alice & Bob description of the protocol:*

1. $V \to C : n_V$
2. $C \to V : \mathsf{zk}(cred_C, k_C, \mathsf{tuple}(n_V, n_C, m, \mathsf{pk}(\mathsf{sk}_{\mathsf{I}})))$

The verifier starts by challenging the client with a fresh nonce, the latter then sends a complex zero-knowledge proof bound to this challenge proving that he knows a credential from the expected issuer bound to the secret k_C he knows. Before accepting this zero-knowledge proof, the verifier V (i) checks the validity of the zero-knowledge proof using the $\mathsf{check_{zk}}$ *operator, and (ii) verifies that this proof is bound to the challenge n_V and to the public key of I using the* $\mathsf{public_{zk}}$ *operator. The processes P_C and P_V are defined as follows:*

$$
\begin{aligned}
P_V \coloneqq\ & \mathsf{out}(c_V, n_V).\\
& \mathsf{in}(c_V, x_1).\\
& \mathsf{let}\ x_2,\ x_3,\ x_4 =\\
& \mathsf{eq}(\mathsf{check_{zk}}(x_1), \mathsf{ok}),\ \mathsf{eq}(\mathsf{proj}_1^4(\mathsf{public_{zk}}(x_1)), n_V),\ \mathsf{eq}(\mathsf{proj}_4^4(\mathsf{public_{zk}}(x_1)), \mathsf{pk}(\mathsf{sk_I}))\ \mathsf{in}\ 0\\
& \mathsf{else}\ \mathsf{out}(c_V, \mathsf{error})
\end{aligned}
$$

$$
\begin{aligned}
P_C \coloneqq\ & \mathsf{in}(c_R, y_1).\\
& \mathsf{out}(c_R, \mathsf{zk}(\mathsf{sign}(\langle k_C, id_C\rangle, \mathsf{sk_I}), k_C, \mathsf{tuple}(y_1, n_C, m, \mathsf{pk}(\mathsf{sk_I})))).
\end{aligned}
$$

*This protocol falls in our class, the two parties being the verifier P_V and the client P_C. The protocol DAA sign, and the simplified version we consider here, has been designed to provide privacy (*i.e. *unlinkability and anonymity as defined in Section 3) to users (*i.e. *clients) inside a group associated to a single issuer. In other words, the privacy set [48] that is typically considered is the set of users who obtained a credential from a single, given issuer. Therefore, as we are interested in modelling different clients having credentials signed by the same issuer, we model* $\mathsf{sk_I}$ *as a private constant in $\Sigma_c \cap \Sigma_{\mathsf{priv}}$ rather than as an identity parameter (we explore this different modelling choice in Example 11).*

The tuple $\Pi_{\mathsf{DAA}} = ((k_C, id_C), n_V, (n_C, m), !, !, P_V, P_C)$ is a protocol according to our Definition 5. We have that k_C or id_C only occur in P_C, and therefore following Definition 6, we have that:

$$\mathcal{M}_{\Pi_{\mathsf{DAA}}} = \left(!\ \mathsf{new}\, n_V.P_V\right)\ \mid\ \left(!\ \mathsf{new}\, (k_C, id_C).\ !\ \mathsf{new}\, (n_C, m).P_C\right)$$

This models infinitely many different clients who obtained credentials from a single issuer having the signature key $\mathsf{sk_I}$*. Any of those clients may take part to infinitely many sessions of the protocol with any verifier associated to that issuer, which executes always the same role (he has no proper identity). We consider here a scenario where sessions can be executed concurrently (both for clients and verifiers). We shall see that our verification methods allows one to automatically prove that privacy is preserved in this scenario.*

Example 11 (Continuing Example 10)**.** *The flexibility of our notion of protocol allows for a subtly different scenario to be analyzed by considering* $\mathsf{sk_I}$ *as being identity-specific (*i.e. *as an identity parameter) instead of being uniform for all identities (*i.e. *private constant). Privacy would then be considered between users associated to different issuers (the privacy set being all users); this is a stronger property that the protocol is not expected to meet. Indeed, a verifier sends ZK proofs whose public parts contain the public key of its credential issuer. We still consider the two parties P_V and P_C but we now model* $\mathsf{sk_I}$ *as an identity parameter. Therefore, we remove* $\mathsf{sk_I}$ *from Σ_c defined in Example 10. The tuple $\Pi_{\mathsf{DAA}} = ((\mathsf{sk_I}, k_C, id_C), n_V, (n_C, m), !, !, P_V, P_C)$ is a protocol according to our Definition 5. We have that* $\mathsf{sk_I}$ *occurs both in P_C and P_V, and therefore following*

Definition 6, we have that:

$$\mathcal{M}_{\Pi_{\mathsf{DAA}}^{\mathsf{sk_I}}} = !\ \mathsf{new}\,(\mathsf{sk_I}, k_C, id_C).\ \left(!\ \mathsf{new}\, n_V.P_V\ \mid\ !\ \mathsf{new}\,(n_C, m).P_C\right)$$

This models (i) infinitely many different clients who obtained pairwise different credentials from infinitely many issuers having pairwise different signature keys $\mathsf{sk_I}$*, and, (ii) infinitely many different verifiers who check for credentials that have been signed by pairwise different issuers. Any of those clients and verifiers may take part to infinitely many sessions of the protocol. We consider here a scenario where users and verifiers sessions can be executed concurrently. Note however that only one user per group (associated to a single issuer) is considered. Relaxing this constraint would require a more generic notion of protocols with 3 parties (this limitation will be discussed in Section 7.1.2). Even for such a weaker scenario, we shall see that our verification methods allows one to find that privacy (unlinkability and anonymity) is already broken (see Figure 3 and details in Section 6.6.2).*

Discussion about shared and non-shared protocols. As mentioned earlier and shown in Definition 6, we distinguish two cases depending on whether *(i)* both roles use identity parameters (*i.e.* when $fn(\mathcal{I}) \cap \overline{k} \neq \varnothing$ and $fn(\mathcal{R}) \cap \overline{k} \neq \varnothing$) or *(ii)* only one role uses identity parameters (*i.e.* when $fn(\mathcal{I}) \cap \overline{k} = \varnothing$ and $fn(\mathcal{R}) \cap \overline{k} \neq \varnothing$, the other case being symmetrical). The case *(i)* corresponds to the case where we should consider an arbitrary number of users for each role, whereas regarding case *(ii)* it is sufficient to consider an arbitrary number of users for role $\mathcal{R}$ only. In addition to this distinction, note that there are two different kinds of protocols that lie in class *(i)*:

(i-a) The *shared case* when $fn(\mathcal{I}) \cap fn(\mathcal{R}) \neq \varnothing$. In such a situation, roles $\mathcal{I}$ and $\mathcal{R}$ share names in $fn(\mathcal{I}) \cap fn(\mathcal{R})$. In practice, this shared knowledge may have been established in various ways such as by using prior protocols, using another communication channel (*e.g.* optical scan of a password as it is done with e-passports, use of PIN codes) or by retrieving the identity from a database that matches the first received message as it is often done with RFID protocols. For such protocols, it is expected that an initiator user and a responder user can communicate successfully producing an honest execution *only if* they have the same identity (*i.e.* they share the same names $\overline{k}$).

(i-b) The *non-shared case* when $fn(\mathcal{I}) \cap fn(\mathcal{R}) = \varnothing$. In such a case, both roles do not share any specific prior knowledge, and it is therefore expected that an initiator and a responder can communicate successfully producing an honest execution whatever their identities.

Unlinkability and anonymity will be uniformly expressed for the cases *(i-a)* and *(i-b)* but our sufficient conditions will slightly differ depending on the case under study.

3. Modelling security properties

This section is dedicated to the definition of the security properties we seek to verify on protocols: unlinkability and anonymity. Those properties are defined using the notion of *trace equivalence* which relates indistinguishable processes.

3.1. Trace equivalence

Intuitively, two configurations are trace equivalent if an attacker cannot tell whether he is interacting with one or the other. Before formally defining this notion, we first introduce a notion of equivalence between frames, called *static equivalence*.

Definition 7. *A frame ϕ is* statically included *in ϕ' when $dom(\phi) = dom(\phi')$, and*

- *for any recipe R such that $R\phi \Downarrow u$ for some u, we have that $R\phi' \Downarrow u'$ for some u';*
- *for any recipes R_1, R_2 such that $R_1\phi \Downarrow u_1$, $R_2\phi \Downarrow u_2$, and $u_1 =_{\mathsf{E}} u_2$, we have that $R_1\phi'\Downarrow =_{\mathsf{E}} R_2\phi'\Downarrow$,* i.e. *there exist v_1, v_2 such that $R_1\phi' \Downarrow v_1$, $R_2\phi' \Downarrow v_2$, and $v_1 =_{\mathsf{E}} v_2$.*

Two frames ϕ and ϕ' are in static equivalence*, written $\phi \sim \phi'$, if the two static inclusions hold.*

Intuitively, an attacker can distinguish two frames if he is able to perform some computation (or a test) that succeeds in ϕ and fails in ϕ' (or the converse).

Example 12. *Let ϕ_0 be the frame given in Example 6, we have that $\phi_0 \sqcup \{w_4 \mapsto k'\} \not\sim \phi_0 \sqcup \{w_4 \mapsto k''\}$. An attacker may observe a difference relying on the computation $R = \mathsf{dec}(w_2, w_4)$.*

Then, *trace equivalence* is the active counterpart of static equivalence, taking into account the fact that the attacker may interfere during the execution of the process. In order to define this, we first introduce $\mathsf{trace}(K)$ for a configuration $K = (\mathcal{P}; \phi)$:

$$\mathsf{trace}(K) = \{(\mathsf{tr}, \phi') \mid (\mathcal{P}, \phi) \xrightarrow{\mathsf{tr}} (\mathcal{P}'; \phi') \text{ for some configuration } (\mathcal{P}'; \phi')\}.$$

Definition 8. *Let K and K' be two configurations. We say that K is* trace included *in K', written $K \sqsubseteq K'$, when, for any $(\mathsf{tr}, \phi) \in \mathsf{trace}(K)$ there exists $(\mathsf{tr}', \phi') \in \mathsf{trace}(K')$ such that $\mathsf{obs}(\mathsf{tr}') = \mathsf{obs}(\mathsf{tr})$ and $\phi \sim \phi'$. They are in* trace equivalence*, written $K \approx K'$, when $K \sqsubseteq K'$ and $K' \sqsubseteq K$.*

Example 13. *Resuming Example 8, we may be interested in checking whether the configurations $K = (!P_{\Pi_{\mathsf{Fh}}}; \varnothing)$ and $K' = (\mathcal{M}_{\Pi_{\mathsf{Fh}}}; \varnothing)$ are in trace equivalence. This equivalence models the fact that Π_{Fh} is unlinkable: each session of the protocol appears to an attacker as if it has been initiated by a different tag, since a given tag can perform at most one session in the idealised scenario K. This equivalence actually holds. It is non-trivial, and cannot be established using existing verification tools such as* ProVerif *or* Tamarin*. The technique developed in this paper will notably allow one to establish it automatically.*

3.2. Security properties under study

In this paper, we focus on two privacy-related properties, namely *unlinkability* and *anonymity*.

3.2.1. Unlinkability

According to the ISO/IEC standard 15408 [2], unlinkability aims at ensuring that a user may make multiple uses of a service or a resource without others being able to link these uses together. In terms of our modelling, a protocol preserves unlinkability if any two sessions of a same role look to an outsider as if they have been executed with different identity names. In other words, an ideal version of the protocol with respect to unlinkability, allows the roles $\mathcal{I}$ and $\mathcal{R}$ to be

executed at most once for each identity names. An outside observer should then not be able to tell the difference between the original protocol and the ideal version of this protocol.

In order to precisely define this notion, we have to formally define this ideal version of a protocol Π. This ideal version, denoted $\mathcal{S}_\Pi$, represents an arbitrary number of agents that can at most execute one session each. Such a process is obtained from $\mathcal{M}_\Pi$ by simply removing the symbols ! and ¡ that are in the scope of identity names. Indeed, those constructs enable each identity to execute an arbitrary number of sessions (respectively concurrently and sequentially). Formally, depending on whether identity names occur in both roles, or only in the responder role, this leads to slightly different definitions.

Definition 9. *Given a protocol* $\Pi = (\overline{k}, \overline{n}_I, \overline{n}_R, \dagger_I, \dagger_R, \mathcal{I}, \mathcal{R})$*, the process* $\mathcal{S}_\Pi$ *is defined as follows:*

- *If* $\overline{k} \cap fn(\mathcal{I}) \neq \varnothing$ *and* $\overline{k} \cap fn(\mathcal{R}) \neq \varnothing$*, then* $\mathcal{S}_\Pi := ! \,\mathsf{new}\, \overline{k}.(\mathsf{new}\, \overline{n}_I.\mathcal{I} \mid \mathsf{new}\, \overline{n}_R.\mathcal{R})$*;*
- *If* $\overline{k} \cap fn(\mathcal{I}) = \varnothing$ *and* $\overline{k} \cap fn(\mathcal{R}) \neq \varnothing$*, then* $\mathcal{S}_\Pi := \dagger_I \,\mathsf{new}\, \overline{n}_I.\mathcal{I} \mid ! \,\mathsf{new}\, \overline{k}.\mathsf{new}\, \overline{n}_R.\mathcal{R}$*.*

Unlinkability is defined as a trace equivalence between $\mathcal{S}_\Pi$ (where each identity can execute at most one session) and $\mathcal{M}_\Pi$ (where each identity can execute an arbitrary number of sessions).

Definition 10. *A protocol* $\Pi = (\overline{k}, \overline{n}_I, \overline{n}_R, \dagger_I, \dagger_R, \mathcal{I}, \mathcal{R})$ *ensures* unlinkability *if* $\mathcal{M}_\Pi \approx \mathcal{S}_\Pi$*.*

Example 14. *Going back to our running example (Example 8), unlinkability is expressed through the equivalence given in Example 13 and recalled below:*

$$! \,\mathsf{new}\, k.(! \,\mathsf{new}\, n_I.P_I \mid ! \,\mathsf{new}\, n_R.P_R) \approx ! \,\mathsf{new}\, k.(\mathsf{new}\, n_I.P_I \mid \mathsf{new}\, n_R.P_R).$$

This intuitively represents the fact that the real situation where a tag and a reader may execute many sessions in parallel is indistinguishable from an idealized one where a given tag and a given reader can execute at most one session for each identity.

Although unlinkability of only one role (*e.g.* the tag for RFID protocols) is often considered in the literature (including [7]), we consider a stronger notion here since both roles are treated symmetrically. As illustrated through the case studies developed in Section 6 (see Sections 6.2 and 6.4), this is actually needed to not miss some practical attacks.

Example 15. *We consider the variant of the toy protocol described in Example 9 where concurrent sessions are authorised for the initiator:* $\Pi^{!}_{\mathsf{Toy}} := (k, (n_T), (n_R), !, !, P'_T, P'_R)$*. We may be interested in checking unlinkability as in Example 14, i.e. whether the following equivalence holds or not:*

$$! \,\mathsf{new}\, k.\big(! \,\mathsf{new}\, n_T.P'_T \mid ! \,\mathsf{new}\, n_R.P'_R\big) \approx ! \,\mathsf{new}\, k.\big(\mathsf{new}\, n_T.P'_T \mid \mathsf{new}\, n_R.P'_R\big)$$

Actually, this equivalence does not hold. When concurrent sessions are authorised, the following scenario is possible: two tags of the same identity can start a session. Then, the attacker just forwards messages from one tag to the other. They can thus complete the protocol. In particular, the mac-key verification stage goes well and the attacker observes that the last conditional of the two tags holds. Such a scenario (which is possible on the left-hand side of the equivalence) cannot be mimicked on the right-hand side (each tag can execute only once). Therefore we have a trace that can only be executed by the multiple sessions process: the equivalence does not hold.

However, we shall see that the original toy protocol of Example 9 (with sessions of the initiator running sequentially only) can be shown unlinkable using the technique developed in this paper. Formally, the following equivalence holds:

$$!\,\mathsf{new}\,k.(\mathsf{new}\,n_T.P'_T \mid \mathsf{new}\,n_R.P'_R) \approx !\,\mathsf{new}\,k.(¡\,\mathsf{new}\,n_T.P'_T \mid ¡\,\mathsf{new}\,n_R.P'_R)$$

3.2.2. Anonymity

According to the ISO/IEC standard 15408 [2], anonymity aims at ensuring that a user may use a service or a resource without disclosing its identity. In terms of our modelling, a protocol preserves anonymity of some identities $\overline{id} \subseteq \overline{k}$, if a session executed with some particular (public) identities $\overline{\mathsf{id}_0}$ looks to an outsider as if it has been executed with different identity names. In other words, an outside observer should not be able to tell the difference between the original protocol and a version of the protocol where the attacker knows that specific roles $\mathcal{I}$ and $\mathcal{R}$ with identities id_0 (known by the attacker) are present.

Definition 11. *Given a protocol* $\Pi = (\overline{k}, \overline{n}_I, \overline{n}_R, \dagger_I, \dagger_R, \mathcal{I}, \mathcal{R})$*, and* $\overline{id} \subseteq \overline{k}$*, the process* $\mathcal{M}_{\Pi,\overline{id}}$ *is defined as follows:*

- *If* $\overline{k} \cap fn(\mathcal{I}) \neq \varnothing$ *and* $\overline{k} \cap fn(\mathcal{R}) \neq \varnothing$*, then* $\mathcal{M}_{\Pi,\overline{id}} := \mathcal{M}_\Pi \mid \mathsf{new}\,\overline{k}.(\dagger_I \mathsf{new}\,\overline{n}_I.\mathcal{I}_0 \mid \dagger_R \mathsf{new}\,\overline{n}_R.\mathcal{R}_0)$.
- *If* $\overline{k} \cap fn(\mathcal{I}) = \varnothing$ *and* $\overline{k} \cap fn(\mathcal{R}) \neq \varnothing$*, then* $\mathcal{M}_{\Pi,\overline{id}} := \mathcal{M}_\Pi \mid \mathsf{new}\,\overline{k}.\ \dagger_R \mathsf{new}\,\overline{n}_R.\mathcal{R}_0$.

where $\mathcal{I}_0 = \mathcal{I}\{\overline{id} \mapsto \overline{\mathsf{id}_0}\}$ *and* $\mathcal{R}_0 = \mathcal{R}\{\overline{id} \mapsto \overline{\mathsf{id}_0}\}$ *for some fresh public constants* $\overline{\mathsf{id}_0}$.

Definition 12. *Let* $\Pi = (\overline{k}, \overline{n}_I, \overline{n}_R, \dagger_I, \dagger_R, \mathcal{I}, \mathcal{R})$*, and* $\overline{id} \subseteq \overline{k}$*. We say that* Π *ensures* anonymity w.r.t. $\overline{id}$ *if* $\mathcal{M}_{\Pi,\overline{id}} \approx \mathcal{M}_\Pi$.

Example 16. *Going back to Example 10, anonymity w.r.t. identity of the client (*i.e. id_C*) is expressed through the following equivalence:*

$$!\,\mathsf{new}\,n_V.P_V \mid (!\,\mathsf{new}\,(k_C, id_C).\,!\,\mathsf{new}\,(n_C, m).P_C) \mid (\mathsf{new}\,(k_C, id_C).!\,\mathsf{new}\,(n_C, m).P_C\{id_C \mapsto \mathsf{id}_0\})$$
$$\approx !\,\mathsf{new}\,n_V.P_V \mid (!\,\mathsf{new}\,(k_C, id_C).\,!\,\mathsf{new}\,(n_C, m).P_C)$$

This intuitively represents the fact that the situation in which a specific client with some known identity id_0 *may execute some sessions is indistinguishable from a situation in which this client is not present at all. Therefore, if these two situations are indeed indistinguishable from the point of view of the attacker, it would mean that there is no way for the attacker to deduce whether a client with a specific identity is present or not.*

3.3. Discussion

The notion of strong unlinkability that we consider is inspired by [7]. In this paper, the authors first propose a definition of *weak unlinkability* that is not expressed via a process equivalence, then they give a notion of *strong unlinkability* that implies the former notion and is expressed via a labelled bisimilarity. The authors argue that, compared to weak unlinkability, the strong variant is too constraining but has the advantage of being more amenable to verification. The first claim is based on an example protocol [7, Theorem 1] where a reader emits an observable "beep" when it sees the same tag twice, which breaks strong unlinkability but not weak unlinkability. Unlike the authors of [7], we do not consider this to be a spurious attack, but a potentially

threatening linkability issue. The second claim is only substantiated by the fact that tools exist for automatically verifying bisimilarities. As discussed before, this is not sufficient. Moreover, there might be spurious attacks on strong unlinkability just because bisimilarity is a very restrictive equivalence: for this reason we would also consider that the strong unlinkability of [7] is too strong, and instead advocate for our variant based on trace equivalence.

We now show formally that, in the setting that we consider, our notion of unlinkability (definition 10) indeed corresponds to the strong unlinkability of [7, Definition 12] where trace equivalence is required rather than bisimilarity. In this original formulation, strong unlinkability is a property of one specific role and not of the whole protocol. As a more technical difference, protocols in [7] may involve more than two roles, and agents may use private channels. In practice, this is used to communicate honest identities in setup phases, as is the case in their BAC case study. If we specialise the setting of [7] to two roles R and T (for Reader and Tag) which fall into the format of definition 3 and do not use the distinguished private channel c, strong unlinkability of the tag role T corresponds to the following labelled bisimilarity:

$$\begin{array}{rl} & \mathsf{new}\, c.\ \big((!\ \mathsf{new}\, \overline{k}.\ !\ \mathsf{out}(c,\overline{k}).\mathsf{new}\, \overline{n}_T.T) \mid (!\ \mathsf{in}(c,\overline{k}).\mathsf{new}\, \overline{n}_R.R)\big) \\ \approx_\ell & \mathsf{new}\, c.\ \big((!\ \mathsf{new}\, \overline{k}.\ \ \mathsf{out}(c,\overline{k}).\mathsf{new}\, \overline{n}_T.T) \mid (!\ \mathsf{in}(c,\overline{k}).\mathsf{new}\, \overline{n}_R.R)\big) \end{array} \tag{1}$$

As communications on channel c are private, eq. (1) is equivalent to:

$$!\ \mathsf{new}\, \overline{k}.\ \big((!\ \mathsf{new}\, \overline{n}_T.T) \mid (!\ \mathsf{new}\, \overline{n}_R.R)\big) \quad \approx_\ell \quad !\ \mathsf{new}\, \overline{k}.\ \big((\mathsf{new}\, \overline{n}_T.T) \mid (\mathsf{new}\, \overline{n}_R.R)\big) \tag{2}$$

The key observation here is that, even though we had only removed replication for the tag role (on the right of eq. (1)), replications are removed for both tags and readers in eq. (2) because communications on c are linear (*i.e.* can be triggered only once). Thus, in this particular case, the only difference between strong unlinkability and our unlinkability is that we rely on trace equivalence rather than labelled bisimilarity.

Several other definitions of unlinkability have been proposed in the literature (see, *e.g.* [24,22] for a comparison). In particular, various game-based formulations have been considered, both in the computational and symbolic models. We first discuss the most common kind of games, called *two-agents games* in [24] and seen e.g. in [13,43,32]. As we shall see, these games can be accurately verified through diff-equivalence, but systematically miss some linkability attacks. We will not need any formal definition, but simply rely on the general idea behind these games, which run in two phases:

1. *Learning phase:* During this phase, the attacker can trigger an arbitrary number of sessions of the two roles (namely tag and reader) with the identity of his choice. This allows him to gain some knowledge. Eventually, the attacker chooses to end the learning phase and enter the second phase.
2. *Guessing phase:* The challenger chooses an identity x among two distinguished identities id_1 and id_2. The attacker is allowed to interact again (an arbitrary number of times) with roles of x, or of identities other than id_1 and id_2.

The attacker wins the game if he can infer whether x is id_1 or id_2, *i.e.* if he is able to distinguish between these two scenarios. The following example shows that these two-agent games miss some linkability attacks, and do not imply unlinkability in our sense for this reason.

Example 17. *We consider a protocol between a tag T and a reader R sharing a symmetric key k. We consider that sessions can be executed in parallel, and we assume that T aborts in case the nonce n_R he receives is equal to the nonce n_T he sent previously (in the same session).*

$$\begin{array}{l} 1.\, T \to R : \{n_T\}_k \\ 2.\, R \to T : \{n_R\}_k \\ 3.\, T \to R : \{n_R \oplus n_T\}_k \end{array}$$

We consider the term algebra introduced in Example 1, and the equational theory introduced in Example 2 with in addition the equation $\mathsf{dec}(\mathsf{enc}(x,y),y) = x$. *To show that the property formally stated in Definition 10 does not hold, consider the following scenario.*

$$\begin{array}{ll} 1.\, T \to R :\ \{n_T\}_k & 1'.\, T' \to R :\ \{n'_T\}_k \\ 2.\, I(R) \to T :\ \{n'_T\}_k & 2'.\, I(R) \to T' :\ \{n_T\}_k \\ 3.\, T \to R :\ \{n'_T \oplus n_T\}_k & 3'.\, T' \to R :\ \{n_T \oplus n'_T\}_k \end{array}$$

A same tag starts two sessions[1] *and therefore generates two nonces n_T and n'_T. The attacker answers to these requests by sending back the two encrypted messages to the tag who will accept both of them, and sends on the network two messages that are actually equal (the exclusive or operator is commutative). Therefore the attacker observes a test, namely the equality between the last two messages, which has no counterpart in the single session scenario. Therefore, this protocol does not ensure unlinkability. In practice, this can be very harmful. Suppose, for example, that tags are distributed among distinct groups (*e.g. *for access control policies) sharing each the same key k. By interacting with two tags, the attacker would then be able to know if they belong to the same group and thus be able to trace groups.*

The previous example illustrates a general phenomenon: two-agent games do not capture concurrent attacks. This is also seen with the protocol of Example 9, which suffers from the attack shown in Example 15, but is secure in the sense of two-agent games — this can actually be proved in ProVerif because the protocol does not involve the exclusive-or primitive. Due to this general weakness, two-agent games do not adequately express unlinkability. They are however convenient for automation, as they can be directly and accurately expressed using the notions of diff-equivalence available in ProVerif or Tamarin. For instance, two-agent games have been used in [13] for unbounded sessions of the DAA protocols — although in this work the security property expressed in this way is called pseudonymity rather than unlinkability.

As pointed out in [24], *three-agent games* have also been considered where the challenge phase is changed as follows: the attacker chooses three tags (a, a_1, a_2) and must distinguish interactions with several tags including tags x and y with the same identity as a, and interactions with x and y having the respective identities of a_1 and a_2. This allows to capture the attacks described above which the two-agent games missed. Three-agent games have successfully been used in [23]

[1] This is possible if different *physical* tags share the same identity, as may be the case e.g. in access control scenarios. In such cases, two different *physical* tags may run sessions concurrently.

for automated verification of unlinkability, though only for a restrictive class of protocols and for bounded sessions only.

We suspect that three-agent games still miss some linkability attacks, though counter-example protocols are likely to be artificial. Further generalisations of these games could then be considered to obtain stronger security properties, and get closer to our notion of unlinkability. In any case, it is important to remark that this line of thought fundamentally relies on having a centralised reader since the attacker must distinguish between scenarios that differ only in the identities of some tags. This contrasts with our notion of unlinkability, which does not assume a centralised reader but treats symmetrically the tag and reader role, more generally called initiator and responder. Such a symmetric treatment is required to model unlinkability when the two parties share a dedicated channel or have an initial shared knowledge, *e.g.* in secure messaging protocols. We also argue that our definition has some value even when analysing protocols featuring centralised readers. In such cases, having reader roles expecting a specific identity seems artificial, but it can actually be seen as a way to model the successive states of a reader (e.g. in LAK, where the tags and reader evolve a common state almost in synchronisation) or a pre-established communication (e.g. in BAC or PACE, where an optical scan is performed to securely exchange a first secret). In any case, our analysis of the aforementioned protocols using our notion of unlinkability has revealed actual attacks that were previously unknown (see Section 6).

4. Our approach

We now define our two conditions, namely frame opacity and well-authentication, and our result which states that these conditions are sufficient to ensure unlinkability and anonymity as defined in Section 3. Before doing that, we shall introduce annotations in the semantics of our processes, in order to ease their analysis.

4.1. Annotations

We shall now define an annotated semantics whose transitions are equipped with more informative actions. The annotated actions will feature labels identifying which concurrent process has performed the action. This will allow us to identify which specific agent (with some specific identity and session names) performed some action.

Given a protocol $\Pi = (\overline{k}, \overline{n}_I, \overline{n}_R, \dagger_I, \dagger_R, \mathcal{I}, \mathcal{R})$ and $\overline{id} \subseteq \overline{k}$, consider any execution of $\mathcal{M}_{\Pi,\overline{id}}$, $\mathcal{M}_\Pi$ or $\mathcal{S}_\Pi$. In such an execution, τ actions are solely used to create new agents (*i.e.* instantiations of $\mathcal{I}$ and $\mathcal{R}$ with new names or constants from id_0) by unfolding replications (*i.e.* !) or repetitions (*i.e.* i), breaking parallel compositions or choosing fresh session and identity parameters. Actions other than τ (that is, input, output and conditionals) are then only performed by the created agents. Formally, we say that an agent is either an instantiation of one of the two roles with some identity and session parameters, or its continuation after the execution of some actions. When $\dagger_A = !$, agents of role A are simply found at toplevel in the multiset of processes. When $\dagger_A = \mathsf{i}$, they may be followed by another process. For instance, in traces of $\mathcal{M}_\Pi$ when $\dagger_I = \mathsf{i}$, newly created initiator agents occur on the left of the sequence in processes of the form:

$$\mathcal{I}\{\overline{k} \mapsto \overline{l}, \overline{n}_I \mapsto \overline{n}\}; \mathsf{i}\, \mathsf{new}\, \overline{m}.\ \mathcal{I}\{\overline{k} \mapsto \overline{l}, \overline{n}_I \mapsto \overline{m}\}.$$

The previous remark allows us to define an *annotated semantics* for our processes of interest. We consider *annotations* of the form $A(\overline{k},\overline{n})$ where $A \in \{I,R\}$ and $\overline{k},\overline{n}$ are sequences of names, or constants from $\overline{\mathsf{id}_0}$. Annotations are noted with the letter a, and the set of annotations is noted $\mathcal{A}$. We can then define an annotated semantics, where agents are decorated by such annotations, indicating their identity and session parameters. An agent P decorated with the annotation a is written $P[a]$, and the actions it performs are also decorated with a, written $\alpha[a]$. Note that this includes τ_{then} and τ_{else} actions; in the annotated semantics, the only non-annotated action is τ. For instance, let us consider $\mathcal{M}_\Pi$ in the annotated semantics when $\dagger_I = \mathsf{i}$. Newly created initiator agents now appear as $\mathcal{I}\{\overline{k}\mapsto\overline{l},\overline{n}_I\mapsto\overline{n}\}[I(\overline{l},\overline{n})];\mathsf{i}\,\mathsf{new}\,\overline{m}.\,\mathcal{I}\{\overline{k}\mapsto\overline{l},\overline{n}_I\mapsto\overline{m}\}$; they execute actions of the form $\alpha[I(\overline{l},\overline{n})]$ with $\alpha\neq\tau$; upon termination of the agent, unannotated τ actions can be executed to create a new agent annotated $I(\overline{l},\overline{m})$ for fresh names $\overline{m}$. We stress that agents having constants $\overline{\mathsf{id}_0}$ as identity parameters shall be annotated with some $A(\overline{k},\overline{n})$ where $\overline{\mathsf{id}_0}\subseteq\overline{k}$. Intuitively, in such a case, we keep in the annotation the information that the identity parameters $\overline{\mathsf{id}_0}$ of that agent has been disclosed to the attacker.

Traces of the annotated semantics will be denoted by ta. We assume[2] that $\overline{n}_I\neq\varnothing$ and $\overline{n}_R\neq\varnothing$, so that at any point in the execution of an annotated trace, an annotation a may not decorate more than one agent in the configuration. Thus, an annotated action may be uniquely traced back to the annotated process that performed it. We also assume that labels used to decorate output actions (*i.e.* elements of $\mathcal{L}$) are added to the produced output actions so that we can refer to them when needed: output actions are thus of the form $\ell:\mathsf{out}(c,w)[a]$.

In annotated traces, τ actions are not really important. We sometimes need to reason up to these τ actions. Given two annotated trace ta and ta', we write $\mathsf{ta}\overset{\tau}{=}\mathsf{ta}'$ when both traces together with their annotations are equal up to some τ actions (but not τ_{then} and τ_{else}). We write $K\overset{\mathsf{ta}}{\Longrightarrow}K'$ when $K\xrightarrow{\mathsf{ta}'}K'$ for some ta' such that $\mathsf{ta}\overset{\tau}{=}\mathsf{ta}'$.

Example 18. *Considering the protocol Π_{Fh} defined in Example 8, process $\mathcal{S}_{\Pi_{\mathsf{Fh}}}$ can notably perform the execution seen in Example 6. The annotated execution has the trace ta given below (up to some τ), where k', n'_I and n'_R are fresh names, $a_I = I(k',n'_I)$ and $a_R = R(k',n'_R)$:*

$$\begin{aligned}\mathsf{ta} = {}& \ell_1:\mathsf{out}(c_I,w_1)[a_I].\mathsf{in}(c_R,w_1)[a_R].\tau_{\mathsf{then}}[a_R].\\ & \ell_2:\mathsf{out}(c_R,w_2)[a_R].\mathsf{in}(c_I,w_2)[a_I].\tau_{\mathsf{then}}[a_I].\\ & \ell_3:\mathsf{out}(c_I,w_3)[a_I].\mathsf{in}(c_R,w_3)[a_R].\tau_{\mathsf{then}}[a_R]\end{aligned}$$

After the initial τ actions, the annotated configuration is $(\{\mathcal{I}\sigma_I[a_I],\mathcal{R}\sigma_R[a_R],\mathcal{S}_\Pi\};\varnothing)$ where $\sigma_I=\{k\mapsto k',n_I\mapsto n'_I\}$, and $\sigma_R=\{k\mapsto k',n_R\mapsto n'_R\}$. The structure is preserved for the rest of the execution with three processes in the multiset (until they become null). After ta, the annotated configuration is $(\{\mathcal{S}_{\Pi_{\mathsf{Fh}}}\};\phi_0)$ where ϕ_0 has been defined in Example 6.

Example 19. *Going back to Example 16 and starting with $\mathcal{M}_{\Pi,\overline{id}}$, a possible annotated configuration obtained after some τ actions can be $K=(\mathcal{P};\varnothing)$ where $\mathcal{P}$ is a multiset containing:*

- $P_C\{k_C\mapsto k_C^0, id_C\mapsto\mathsf{id}_0, n_C\mapsto n_C^0, m\mapsto m_C^0\}[a_C^0]$;
- $!\mathsf{new}\ (n_C,m).P_C\{k_C\mapsto k_C^0, id_C\mapsto\mathsf{id}_0\}$;

[2] This assumption only serves the purpose of uniquely identifying agents. The assumed session nonces do not have to occur in the corresponding roles, so this does not require to change the protocol under study.

– $P_V\{n_V \mapsto n_V^1\}[a_V^1]$; *and*
– $\mathcal{M}_{\Pi_{\mathsf{DAA}}}$.

where $a_V^1 = \mathcal{I}(\epsilon, n_V^1)$ *and* $a_C^0 = \mathcal{R}((k_C^0, \mathsf{id}_0), (n_C^0, m^0))$. *We may note that the annotation* a_V^1 *contains the empty sequence* ϵ *since the initiator role does not rely on identity names; and the annotation* a_C^0 *contains* id_0.

4.2. Frame opacity

In light of attacks based on leakage from messages where non-trivial relations between outputted messages are exploited by the attacker to trace an agent, our first condition will express that all relations the attacker can establish on output messages only depend on what is already observable by him and never depend on a priori hidden information such as identity names of specific agents. Therefore, such relations cannot be exploited by the attacker to learn anything new about the agents involved in the execution. We achieve this by requiring that any reachable frame must be indistinguishable from an *idealised frame* that only depends on data already observed in the execution, and not on the specific agents (and their names) of that execution.

As a first approximation, one might take the idealisation of a frame $\{w_1 \mapsto u_1, \ldots, w_l \mapsto u_n\}$ to be $\{w_1 \mapsto n_1, \ldots, w_l \mapsto n_l\}$ where the $n_1, \ldots, n_l$ are distinct fresh names. It would then be very strong to require that frames obtained in arbitrary protocol executions are statically equivalent to their idealisation defined in this way. Although this would allow us to carry out our theoretical development, it would not be realistic since any protocol using, *e.g.* a pair, would fail to satisfy this condition. We thus need a notion of idealisation that retains part of the shape of messages, which a priori does not reveal anything sensitive to the attacker. We also want to allow outputs to depend on session names or previous inputs in ways that are observable, *e.g.* to cover the output of the signature of a previously inputted message.

Our idealised frames will be obtained by replacing each message, produced by an output of label ℓ, by a context that only depends on ℓ, whose holes are filled with fresh session names and (idealisations of) previously inputted messages. Intuitively, this is still enough to ensure that the attacker does not learn anything that is identity-specific. In order to formalise this notion, we assume two disjoint and countable subsets of variables: *input variables* $\mathcal{X}^{\mathsf{i}} = \{x_1^{\mathsf{i}}, x_2^{\mathsf{i}}, \ldots\} \subseteq \mathcal{X}$, and *name variables* $\mathcal{X}^{\mathsf{n}} = \{x_1^{\mathsf{n}}, x_2^{\mathsf{n}}, \ldots\} \subseteq \mathcal{X}$. We also consider a fixed but arbitrary *idealisation operator* $\mathrm{ideal}(\cdot) : \mathcal{L} \to \mathcal{T}(\Sigma, \mathcal{X}^{\mathsf{i}} \cup \mathcal{X}^{\mathsf{n}})$. Variables x_j^{i} intuitively refers to the j-nth variable received by the agent of interest. Therefore, we assume that our idealisation operator satisfies the following: for all $\ell \in \mathcal{L}$, we have that $\mathrm{ideal}(\ell) \cap \mathcal{X}^{\mathsf{i}} \subseteq \{x_1^{\mathsf{i}}, \ldots, x_k^{\mathsf{i}}\}$ where k is the number of inputs preceding the output labelled ℓ.

Definition 13. *Let* $\mathrm{fr} : \mathcal{A} \times \mathcal{X}^{\mathsf{n}} \to \mathcal{N}$ *be an injective function assigning names to each agent and name variable. We define the idealised frame associated to* ta*, denoted* $\Phi^{\mathrm{fr}}_{\mathrm{ideal}}(\mathsf{ta})$*, inductively on the annotated trace* ta*:*

– $\Phi^{\mathrm{fr}}_{\mathrm{ideal}}(\epsilon) = \varnothing$ *and* $\Phi^{\mathrm{fr}}_{\mathrm{ideal}}(\mathsf{ta}.\alpha) = \Phi^{\mathrm{fr}}_{\mathrm{ideal}}(\mathsf{ta})$ *if* α *is not an output;*
– $\Phi^{\mathrm{fr}}_{\mathrm{ideal}}\big(\mathsf{ta}.(\ell : \mathsf{out}(c, w)[a])\big) = \Phi^{\mathrm{fr}}_{\mathrm{ideal}}(\mathsf{ta}) \cup \{w \mapsto \mathrm{ideal}(\ell)\sigma^{\mathsf{i}}\sigma^{\mathsf{n}}\Downarrow\}$ *where*
  * $\sigma^{\mathsf{n}}(x_j^{\mathsf{n}}) = \mathrm{fr}(a, x_j^{\mathsf{n}})$ *when* $x_j^{\mathsf{n}} \in \mathcal{X}^{\mathsf{n}}$*, and*
  * $\sigma^{\mathsf{i}}(x_j^{\mathsf{i}}) = R_j\Phi^{\mathrm{fr}}_{\mathrm{ideal}}(\mathsf{ta})$ *when* $x_j^{\mathsf{i}} \in \mathcal{X}^{\mathsf{i}}$ *and* R_j *is the recipe corresponding to the* j*-th input of agent* a *in* ta.

We may note this notion is not necessarily well-defined, as $\mathsf{ideal}(\ell)\sigma^{\mathsf{i}}\sigma^{\mathsf{n}}$ may not compute to a message. Note also that well-definedness does not depend on the choice of the function fr. Remark also that, by definition, $\Phi^{\mathrm{fr}}_{\mathrm{ideal}}(\mathsf{ta})$ never depends on the specific identity names occurring in ta. In particular, idealised frames do not depend on whether agents rely on the specific constants $\overline{\mathsf{id}_0}$ or not.

Example 20. *Continuing Example 18, we consider the idealisation operator defined as follows:* $\ell_1 \mapsto x_1^{\mathsf{n}}, \ell_2 \mapsto x_2^{\mathsf{n}}, \ell_3 \mapsto x_3^{\mathsf{n}}$. *Let* fr *be an injective function such that* $\mathrm{fr}(a_I, x_j^{\mathsf{n}}) = n_j^I$ *and* $\mathrm{fr}(a_R, x_j^{\mathsf{n}}) = n_j^R$. *We have that* $\Phi^{\mathrm{fr}}_{\mathrm{ideal}}(\mathsf{ta}) = \{w_1 \mapsto n_1^I, w_2 \mapsto n_2^R, w_3 \mapsto n_3^I\}$.

On the latter simple example, such an idealisation will be sufficient to establish that any reachable frame obtained through an execution of $\mathcal{M}_{\Pi_{\mathsf{Fh}}}$ is indistinguishable from its idealisation. However, as illustrated by the following two examples, we sometimes need to consider more complex idealisation operators.

Example 21. *Continuing Example 9, to establish our indistinguishability property, namely* frame opacity *defined below, we will consider:*

$$\ell_1 \mapsto x_1^{\mathsf{n}},\ \ell_2 \mapsto x_2^{\mathsf{n}},\ \ell_3 \mapsto x_3^{\mathsf{n}},\ \ell_4 \mapsto x_4^{\mathsf{n}}$$

assuming that the four outputs are labelled with ℓ_1, ℓ_2, ℓ_3, *and* ℓ_4 *respectively.*

Example 22. *Regarding Example 10, we also need to define an idealisation that retains the shape of the second outputted message. Moreover, the idealisation of the second outputted message will depend on the nonce previously received. Assuming that the outputs are labelled with* ℓ_1 *and* ℓ_2 *respectively, we consider:* $\ell_1 \mapsto x_1^{\mathsf{n}}$, $\ell_2 \mapsto \mathsf{zk}(\mathsf{sign}(\langle x_2^{\mathsf{n}}, x_3^{\mathsf{n}}\rangle, \mathsf{sk}_{\mathsf{I}}), x_2^{\mathsf{n}}, \mathsf{tuple}(x_1^{\mathsf{i}}, x_4^{\mathsf{n}}, x_5^{\mathsf{n}}, \mathsf{pk}(\mathsf{sk}_{\mathsf{I}})))$. *Note that such an idealisation would not work for Example 11; there* sk_{I} *is an identity-parameter which, following Definition 13, cannot occur in* $\mathsf{ideal}(\ell_2)$. *It turns out that frame opacity cannot be established (for any heuristics considered by our tool UKano) for a good reason: unlinkability fails to hold for this variant. Essentially, this is because verifiers send ZK proofs whose the public parts contain the public key of their issuers.*

The following proposition establishes that the particular choice of fr in $\Phi^{\mathrm{fr}}_{\mathrm{ideal}}(\mathsf{ta})$ is irrelevant with respect to static equivalence. We can thus note $\Phi_{\mathrm{ideal}}(\mathsf{ta}) \sim \phi$ instead of there exists fr such that $\Phi^{\mathrm{fr}}_{\mathrm{ideal}}(\mathsf{ta}) \sim \phi$.

Proposition 1. *Let* $\Phi^{\mathrm{fr}}_{\mathrm{ideal}}(\mathsf{ta})$ *(resp.* $\Phi^{\mathrm{fr}'}_{\mathrm{ideal}}(\mathsf{ta})$*) be the idealised frame associated to* ta *relying on* fr *(resp.* fr′*). We have that* $\Phi^{\mathrm{fr}}_{\mathrm{ideal}}(\mathsf{ta}) \sim \Phi^{\mathrm{fr}'}_{\mathrm{ideal}}(\mathsf{ta})$.

Proof. It is sufficient to observe that $\Phi^{\mathrm{fr}}_{\mathrm{ideal}}(\mathsf{ta})$ and $\Phi^{\mathrm{fr}'}_{\mathrm{ideal}}(\mathsf{ta})$ are equal up to a bijective renaming of names. □

We can now formalise the notion of frame opacity as announced: it requires that all reachable frames must be statically equivalent to idealised frames.

Definition 14. *The protocol* Π *ensures* frame opacity *w.r.t.* ideal *if for any execution* $(\mathcal{M}_{\Pi,\overline{id}}; \varnothing) \xrightarrow{\mathsf{ta}} (Q; \phi)$ *we have that* $\Phi_{\mathrm{ideal}}(\mathsf{ta})$ *is defined and* $\Phi_{\mathrm{ideal}}(\mathsf{ta}) \sim \phi$.

There are many ways to choose the idealisation operator ideal(·). We present below a *syntactical construction* that is sufficient to deal with almost all our case studies. This construction has been implemented as a heuristic to automatically build idealisation operators in the tool UKano. The tool UKano also provides other heuristics that generally lead to better performance but are less tight (*i.e.* they cannot always be used to establish frame opacity). We explain how UKano verifies frame opacity and compare the different heuristics it can leverage in Section 6.

At first reading, it is possible to skip the rest of the section and directly go to Section 4.3 since proposed canonical constructions are just instantiations of our generic notion of idealisation.

4.2.1. Syntactical idealisation

Intuitively, this construction builds the idealisation operator by examining the initiator and responder roles as syntactically given in the protocol definition. The main idea is to consider (syntactical) outputted terms one by one, and to replace identity parameters, as well as variables bound by a let construct by pairwise distinct names variables, *i.e.* variables in $\mathcal{X}^{\mathsf{n}}$.

Definition 15. *Let* $\Pi = (\overline{k}, \overline{n}_I, \overline{n}_R, \dagger_I, \dagger_R, \mathcal{I}, \mathcal{R})$ *be a protocol that uses input variables* $\{x^{\mathsf{i}}_1, x^{\mathsf{i}}_2, \ldots\} \subseteq \mathcal{X}^{\mathsf{i}}$ *(in this order) for its two roles, and distinct variables from* $\mathcal{X}_{\mathsf{let}}$ *in let constructions. Let* $\sigma : \overline{k} \cup \overline{n_R} \cup \overline{n_I} \cup \mathcal{X}_{\mathsf{let}} \to \mathcal{X}^{\mathsf{n}}$ *be an injective renaming. The syntactical idealisation operator maps any* $\ell \in \mathcal{L}$ *occurring in an output action* $\ell : \mathsf{out}(c, u)$ *in* $\mathcal{I}$ *or* $\mathcal{R}$ *(for some* c *and some* u*) to* $u\sigma$*.*

Example 23. *Continuing Example 8, we first perform some renaming to satisfy the conditions imposed by the previous definition. We therefore replace* x_1 *by* x^{i}_1 *in role* $\mathcal{I}$*, and* y_1, y_2 *by* $x^{\mathsf{i}}_1, x^{\mathsf{i}}_2$ *in role* $\mathcal{R}$*. We assume that* x_2, x_3*, and* y_3 *are elements of* $\mathcal{X}_{\mathsf{let}}$*. We consider a renaming* σ *that maps* $k, n_I, n_R, x_2, x_3, y_3$ *to* $x^{\mathsf{n}}_1, \ldots, x^{\mathsf{n}}_6$*. We obtain the following idealisation operator:*

$$\ell_1 \mapsto x^{\mathsf{n}}_2; \quad \ell_2 \mapsto \mathsf{enc}(\langle x^{\mathsf{i}}_1, x^{\mathsf{n}}_3\rangle, x^{\mathsf{n}}_1); \quad \ell_3 \mapsto \mathsf{enc}(\langle x^{\mathsf{n}}_5, x^{\mathsf{n}}_2\rangle, x^{\mathsf{n}}_1).$$

Considering fr *as defined in Example 20,* i.e. *such that* $\mathrm{fr}(a_I, x^{\mathsf{n}}_j) = n^I_j$ *and* $\mathrm{fr}(a_R, x^{\mathsf{n}}_j) = n^R_j$*, and relying on the idealisation operator defined above, and* ta *as given in Example 18, we have that:* $\Phi^{\mathrm{fr}}_{\mathrm{ideal}}(\mathsf{ta}) = \{w_1 \mapsto n^I_2,\ w_2 \mapsto \mathsf{enc}(\langle n^I_2, n^R_3\rangle, n^R_1),\ w_3 \mapsto \mathsf{enc}(\langle n^I_5, n^I_2\rangle, n^I_1)\}$*. This idealisation is different from the one described in Example 20, but it also allows us to establish frame opacity.*

Example 24. *Continuing Example 10, we consider a renaming* σ *that maps:* k_C*,* id_C*,* n_V*,* n_C*,* m*,* x_2*,* x_3*,* x_4*,* y_3 *to* $x^{\mathsf{n}}_1, x^{\mathsf{n}}_2, \ldots, x^{\mathsf{n}}_9$*. We obtain the following idealisation operator:*

$$\ell_1 \mapsto x^{\mathsf{n}}_3; \quad \ell_2 \mapsto \mathsf{zk}(\mathsf{sign}(\langle x^{\mathsf{n}}_1, x^{\mathsf{n}}_2\rangle, \mathsf{sk_I}), x^{\mathsf{n}}_1, \mathsf{tuple}(x^{\mathsf{i}}_1, x^{\mathsf{n}}_4, x^{\mathsf{n}}_5, \mathsf{pk}(\mathsf{sk_I}))); \quad \ell_3 \mapsto \mathsf{error}.$$

Such an idealisation operator is also suitable to establish frame opacity.

As illustrated by the previous examples, the syntactical idealisation is sufficient to conclude on most examples. Actually, using this canonical construction, we automatically build the idealisation operator and check frame opacity for all the examples we have introduced in the previous sections and for most of the case studies presented in Section 6.

4.2.2. Semantical idealisation

The previous construction is clearly purely syntactic and therefore closely connected to the way the roles of the protocol are written. Its main weakness lies in the way variables are bound by let constructions. Since there is no way to statically guess the shape of messages that will be instantiated for those variables, the previous technique replaces them by fresh session names. False negatives may result from such over-approximations. We may therefore prefer to build an idealisation operator looking at the messages outputted during a concrete execution. In such a case, we may simply retain part of the shape of messages, which a priori does not reveal anything sensitive to the attacker (*e.g.* pairs, lists). This can be formalised as follows:

Definition 16. *A symbol* f *(of arity* n*) in* Σ *is* transparent *if it is a public constructor symbol that does not occur in* E *and such that: for all* $1 \leq i \leq n$*, there exists a recipe* $R_i \in \mathcal{T}(\Sigma_{\mathsf{pub}}, \{w\})$ *such that for any message* $u = \mathsf{f}(u_1, \ldots, u_n)$*, we have that* $R_i\{w \mapsto u\} \Downarrow v_i$ *for some* v_i *such that* $v_i =_{\mathsf{E}} u_i$*.*

Example 25. *Considering the signature and the equational theory introduced in Example 1 and Example 2, the symbols* $\langle\ \rangle$ *and* ok *are the only ones that are transparent. Regarding the pairing operator, the recipes* $R_1 = \mathsf{proj}_1(w)$ *and* $R_2 = \mathsf{proj}_2(w)$ *satisfy the requirements.*

Once the set Σ_t of transparent functions is fixed, the idealisation associated to a label ℓ occurring in Π will be computed relying on a particular (but arbitrary) message u that has been outputted with this label ℓ during a concrete execution of $\mathcal{M}_\Pi$. The main idea is to go through transparent functions until getting stuck, and then replacing the remaining sub-terms using distinct name variables from $\mathcal{X}^{\mathsf{n}}$.

Example 26. *Considering the protocol given in Example 8, the resulting idealisation associated to* Π *(considering messages in* ϕ_0 *as defined in Example 6) is:* $\ell_1 \mapsto x_1^{\mathsf{n}}$; $\ell_2 \mapsto x_2^{\mathsf{n}}$; $\ell_3 \mapsto x_3^{\mathsf{n}}$*. Even if this idealisation operators is quite different from the one presented in Example 23. it is also suitable to establish frame opacity.*

In [42], the idealisation operator associated to Π was exclusively computed using this method. The technique is implemented in the tool UKano, and yields simple idealisations for which frame opacity often holds and can be established quickly. However, it happens to be insufficient to establish frame opacity in presence of function symbols that are neither transparent nor totally opaque such as signatures. Indeed, a signature function symbol is not transparent according to our definition: an attacker can make the difference between a signature $\mathsf{sign}(m, sk(A))$ and a random nonce. Therefore, replacing such a term by a fresh session name will never allow one to establish frame opacity. That is why we also defined other types of idealisations that produce more complex idealised messages but allow for a much better level of precision. In practice, our tool UKano has three different built-in heuristics for computing idealisations which span the range between precision (syntactical idealisation) and efficiency (semantical idealisation).

4.3. Well-authentication

Our second condition will prevent the attacker from obtaining some information about agents through the outcome of conditionals. To do so, we will essentially require that conditionals of $\mathcal{I}$ and $\mathcal{R}$ can only be executed successfully in honest, intended interactions. However, it is unnecessary to impose such a condition on conditionals that never leak any information, which are

found in several security protocols. We characterise below a simple class of such conditionals, for which the attacker will always know the outcome of the conditional based on the past interaction.

Definition 17. *For a protocol* Π*, a conditional* $\mathsf{let}\ \overline{z} = \overline{t}\ \mathsf{in}\ P\ \mathsf{else}\ Q$ *occurring in* $\mathcal{A} \in \{\mathcal{I}, \mathcal{R}\}$ *is* safe *if* $\overline{t} \in \mathcal{T}(\Sigma_{\mathsf{pub}}, \{x_1, \ldots, x_n\} \cup \{u_1, \ldots, u_m\})$*, where the* x_i *are the variables bound by the previous inputs of that role, and* u_i *are the messages used in the previous outputs of that role.*

Example 27. *Consider the process* $\mathsf{out}(c, u).\mathsf{in}(c, x).\mathsf{let}\ z = \mathsf{neq}(x, u)\ \mathsf{in}\ P\ \mathsf{else}\ Q$*. The conditional is used to ensure that the agent will not accept as input the message he sent at the previous step. Such a conditional is safe according to our definition.*

Note that trivial conditionals required by the grammar of protocols (Definition 3) are safe and will thus not get in the way of our analysis. We can now formalise the notion of association, which expresses that two agents are having an honest, intended interaction, *i.e.* the attacker essentially did not interfere in their communications. For an annotated trace ta and annotations a and a', we denote by $\mathsf{ta}|_{a,a'}$ the subsequence of ta that consists of actions of the form $\alpha[a]$ or $\alpha[a']$.

Definition 18. *Given a protocol* Π*, two annotations* $a_1 = A_1(\overline{k}_1, \overline{n}_1)$ *and* $a_2 = A_2(\overline{k}_2, \overline{n}_2)$ *are* associated *in* (ta, ϕ) *if:*

- *they are* dual, i.e. $A_1 \neq A_2$*, and* $\overline{k_1} = \overline{k_2}$ *when* $fn(\mathcal{R}) \cap fn(\mathcal{I}) \neq \varnothing$ *(the shared case);*
- *the interaction* $\mathsf{ta}|_{a_1,a_2}$ *is honest for* ϕ *(see Definition 4).*

Example 28. *Continuing Example 18,* $I(k', n'_I)$ *and* $R(k', n'_R)$ *are associated in* (ta, ϕ_0)*.*

Finally, we can state our second condition.

Definition 19. *The protocol* Π *is* well-authenticating *if, for any* $(\mathcal{M}_{\Pi,\overline{id}}; \varnothing) \xrightarrow{\mathsf{ta}.\tau_{\mathsf{then}}[a]} (\mathcal{P}; \phi)$*, either the last action corresponds to a safe conditional of* $\mathcal{I}$ *or* $\mathcal{R}$*, or there exists* a' *such that:*

(i) *The annotations* a *and* a' *are associated in* (ta, ϕ)*;*
(ii) *Moreover, when* $fn(\mathcal{R}) \cap fn(\mathcal{I}) \neq \varnothing$ *(the shared case),* a' *(resp.* a*) is only associated with* a *(resp.* a'*) in* (ta, ϕ)*.*

Intuitively, this condition does not require anything for safe conditionals as we already know that they cannot leak new information to the attacker (he already knows their outcome). For unsafe conditionals, condition *(i)* requires that whenever an agent a evaluates them positively (*i.e.* he does not abort the protocol), it must be the case that this agent a is so far having an honest interaction with a dual agent a'. Indeed, as discussed in introduction, it is crucial to avoid such unsafe conditionals to be evaluated positively when the attacker is interfering because this could leak crucial information. In the rest of the paper, when considering a protocol Π, we will say that a conditional in a process resulting from $\mathcal{M}_{\Pi,\overline{id}}$ or $\mathcal{S}_\Pi$ is *safe* when it corresponds to a safe conditional in $\mathcal{I}$ or $\mathcal{R}$.

As illustrated in the following example, condition *(ii)* is needed to prevent from having executions where an annotation is associated to several annotations, which would break unlinkability in the shared case (*i.e.* when $fn(\mathcal{R}) \cap fn(\mathcal{I}) \neq \varnothing$).

Example 29. *We consider a protocol between an initiator and a responder that share a symmetric key* k*. The protocol can be described informally as follows:*

$$1.\ I \to R : \{n_I\}_k$$
$$2.\ R \to I : n_R$$

Assuming that the two outputs are labelled with ℓ_1 and ℓ_2 respectively, the idealisation operator $\ell_1 \mapsto x_1^{\mathsf{n}}$, $\ell_2 \mapsto x_2^{\mathsf{n}}$ is suitable to establish frame opacity. We may note that the only conditional is the one performed by the responder role when receiving the ciphertext. He will check whether it is indeed an encryption with the expected key k. When an action $\tau_{\mathsf{then}}[R(k, n_R)]$ occurs, it means that a ciphertext encrypted with k has been received by $R(k, n_R)$ and since the key k is unknown by the attacker, such a ciphertext has been sent by a participant: this is necessarily a participant executing the initiator role with key k. Hence condition (i) of well-authentication holds (and can actually be formally proved). However, condition (ii) fails to hold since two responder roles may accept a same ciphertext $\{n_I\}_k$ and therefore be associated to the same agent acting as an initiator. This corresponds to an attack scenario w.r.t. our formal definition of unlinkability since such a trace will have no counterpart in $\mathcal{S}_\Pi$. More formally, the trace $\mathsf{tr} = \mathsf{out}(c_I, w_0).\mathsf{in}(c_R, w_0).\tau_{\mathsf{then}}.\mathsf{out}(c_R, w_1).\mathsf{in}(c_R, w_0).\tau_{\mathsf{then}}.\mathsf{out}(c_R, w_2)$ will be executable starting from $\mathcal{M}_\Pi$ and will allow one to reach $\phi = \{w_0 \mapsto \mathsf{enc}(n_I, k);\ w_1 \mapsto n_R;\ w_2 \mapsto n'_R\}$. Starting from $\mathcal{S}_\Pi$ the second action τ_{then} will not be possible, and more importantly this will prevent the observable action $\mathsf{out}(c_R, w_2)$ to be triggered.

While the condition (i) of well-authentication is verifiable quite easily by expressing it as simple reachability properties (as explained in Section 5.2), the required condition (ii) for the shared-case is actually harder to express in existing tools. We therefore shall prove that, for the shared case, once condition (i) of well-authentication is known to hold, condition (ii) is a consequence of two simpler conditions that are easier to verify (as shown in Section 5.2.2). First, the first conditional of the responder role should be safe — remark that if this does not hold, similar attacks as the one discussed above may break unlinkability. Second, messages labelled by some ℓ outputted in honest interactions by different agents should always be different.

Lemma 1. *Let $\Pi = (\overline{k}, \overline{n}_I, \overline{n}_R, \dagger_I, \dagger_R, \mathcal{I}, \mathcal{R})$ be a protocol such that $fn(\mathcal{I}) \cap fn(\mathcal{R}) \neq \varnothing$ (shared case) that satisfies condition (i) of well-authentication. Then well-authentication holds provided that:*

(a) the first conditional that occurs in $\mathcal{R}$ is safe;

(b) for any execution $(\mathcal{M}_{\Pi,\overline{id}}; \varnothing) \xrightarrow{\mathsf{ta}} (\mathcal{P}; \phi)$, if $\mathsf{ta}_1 = \mathsf{ta}|_{a_1,b_1}$ and $\mathsf{ta}_2 = \mathsf{ta}|_{a_2,b_2}$ are honest with $a_1 \neq a_2$ then for any $\ell : \mathsf{out}(c, w_1)[a_1] \in \mathsf{ta}_1$ and $\ell : \mathsf{out}(c, w_2)[a_2] \in \mathsf{ta}_2$ then $\phi(w_1) \not=_{\mathsf{E}} \phi(w_2)$.

Proof. Consider an execution $\mathcal{M}_{\Pi,\overline{id}} \xrightarrow{\mathsf{ta}.\tau_{\mathsf{then}}[a']} (\mathcal{P}; \phi)$ where two agents a and a' are associated and a' has performed the last τ_{then}. If this test corresponds to a safe conditional, there is nothing to prove. Otherwise, we shall prove that a is only associated to a', and vice versa.

Agent a' is only associated to a. Consider the last input of a' (the one just before $\tau_{\mathsf{then}}[a']$) and the output of a that occurs before this input of a':

$$\mathcal{M}_{\Pi,\overline{id}} \xrightarrow{\mathsf{ta}.\mathsf{out}(c,w_\ell)[a].\mathsf{ta}'.\mathsf{in}(c',R)[a'].\mathsf{ta}''.\tau_{\mathsf{then}}[a']} (\mathcal{P}; \phi)$$

We have $R\phi \Downarrow \phi(w_\ell)$ where w_ℓ is labelled ℓ. Assume, for the sake of contradiction, that a' is associated to another agent $b \neq a$. Then, we have $R\phi \Downarrow_{=_{\mathsf{E}}} \phi(w'_\ell)$ for some handle, and thus thanks to Item 6 of Definition 1, we have that $\phi(w_\ell) =_{\mathsf{E}} \phi(w'_\ell)$, for a handle w'_ℓ corresponding to some output of b labelled ℓ in the honest trace $\mathsf{ta}|_{a',b}$. This contradicts assumption (b).

Agent a is only associated to a'. Agent a must have performed an input in ta: this is obvious if a is a responder, and follows from assumption (a) otherwise. Let $\ell : \mathsf{out}(c, w_\ell)[a']$ be the output label (with annotation a') occurring in ta just before the input of a mentioned above. The considered execution is thus of the following form:

$$\mathcal{M}_{\Pi,\overline{id}} \xrightarrow{\mathsf{ta}.\mathsf{out}(c,w_\ell)[a'].\mathsf{ta}'.\mathsf{in}(c,R)[a].\mathsf{ta}''.\tau_{\mathsf{then}}[a']} (\mathcal{P};\phi)$$

We know that the message m, satisfying $R\phi \Downarrow m$, which is inputted by a is equal (modulo E) to the previous output of a', that is $\phi(w_\ell)$. As for the previous case, condition (b) implies that it cannot be equal to the output of another agent having an honest interaction in ta, thus a is only associated to a'. □

4.4. Main result

Our main theorem establishes that the previous two conditions are sufficient to ensure unlinkability and anonymity.

Theorem 1. *Consider a protocol $\Pi = (\overline{k}, \overline{n}_I, \overline{n}_R, \dagger_I, \dagger_R, \mathcal{I}, \mathcal{R})$ and some identity names $\overline{id} \subseteq \overline{k}$. If the protocol ensures both well-authentication and frame opacity w.r.t. $\overline{id}$, then Π ensures unlinkability and anonymity w.r.t. $\overline{id}$.*

Note that, when $\overline{id} = \varnothing$, we have that $\mathcal{M}_{\Pi,\overline{id}} \approx \mathcal{M}_\Pi$ and our two conditions coincide on $\mathcal{M}_{\Pi,\overline{id}}$ and $\mathcal{M}_\Pi$. We thus have as a corollary that if $\mathcal{M}_\Pi$ ensures well-authentication and frame opacity, then Π is unlinkable.

The proof of this theorem is detailed in Appendix A, and we explain in Section 5 how to check these two conditions in practice relying on existing verification tools. We apply our method on various case studies that are detailed in Section 6. Below, we only briefly summarize the result of the confrontation of our method to our various running examples, focusing on unlinkability.

Protocol	Frame opacity	Well-authentication	Unlinkability
Feldhofer (Example 8)	✓	✓	**safe**
Toy protocol with ! (Example 15)	✓	×	attack
Toy protocol with ¡ (Example 9)	✓	✓(with Tamarin)	**safe**
DAA-like with one issuer (Example 10)	✓	✓	**safe**
DAA-like with many issuers (Example 11)	×	✓	attack

Fig. 3. Summary of our running examples.

We note ✓ for a condition automatically checked using UKano and × when the condition does not hold. For the analysis of the Toy protocol with ¡, we do not rely on UKano (which is based on ProVerif) since ProVerif does not support the ¡ operator. We establish the well-authentication property using Tamarin and frame opacity using ProVerif by allowing sessions to run concurrently and thus doing a sound over-approximation of the protocol's behaviors. Frame opacity has been established relying on the syntactical idealisation as well as the semantical one, except for Example 10. Indeed, as explained at the end of Section 4.2, the semantical idealisation is not suitable in this case.

5. Mechanization

We now discuss how to verify unlinkability and anonymity in practice, through the verification of our two conditions. More specifically, we describe how appropriate encodings allow one to verify frame opacity (Section 5.1) and well-authentication (Section 5.2), respectively through diff-equivalence and correspondence properties in ProVerif.

We additionally provide a tool, called UKano [51] (Section 5.3), which mechanises the encodings described in this section. Our tool takes as input a specification of a protocol in our class, computes encodings, and calls ProVerif to automatically check our two conditions, and thus unlinkability and anonymity. As briefly mentioned in Section 4 and detailed in Section 6, UKano concludes on many interesting case studies.

5.1. Frame opacity

We shall describe how to encode frame opacity using the diff-equivalence of ProVerif [20]. In a nutshell, we will check this strong notion of equivalence between $\mathcal{M}_{\Pi,\overline{id}}$ and a modified version of it that produces idealised outputs instead of real ones, in order to check static equivalence between all pairs of frames of the form $(\Phi, \Phi_{\text{ideal}}(\mathsf{ta}))$ where ta is executable by $\mathcal{M}_{\Pi,\overline{id}}$ and Φ is the resulting frame. The main issue in implementing this idea arises from diff-equivalence being too strong regarding tests and computations of idealized terms. In [42], we had proposed a solution that avoided this problem by largely over-approximating the set of executable traces, which is sound but very imprecise. Moreover, this first solution is only adequate for the notion of idealization considered in [42], and not for the generalization proposed in the present paper. We describe below a simpler solution, that is much more precise and efficient, and can accommodate our generalized notion of idealization, at the cost of a slight extension of ProVerif's diff-equivalence. We shall start with a brief reminder on diff-equivalence in ProVerif, in order to describe how we extend it, before showing how this extension allows us to naturally encode frame opacity.

Diff-equivalence. Intuitively, diff-equivalence is obtained from trace equivalence by forcing the two processes (or configurations) being compared to follow the same execution. It has been introduced in [20] as a means to automatically verify observational equivalence in ProVerif. This paper deals with the full process algebra supported by ProVerif, which is more general but compatible with the process algebra of the present paper, the main difference being that we do not account for private channels. Processes of [20] are equipped with a reduction semantics, noted $P \to P'$. This allows to define observational equivalence, which implies trace equivalence in our sense. The key notion in [20] is that of a *bi-process*, that is a process in which some terms are replaced by *bi-terms* of the form $\mathsf{choice}[u_1, u_2]$. Given a bi-process P, its first *projection* $\mathsf{fst}(P)$ is defined by taking the first component of all choice operators occurring in it. The second projection $\mathsf{snd}(P)$ is defined analogously. Bi-processes are given a reduction semantics by taking

the same rules as those for processes (which we do not recall) with three modified rules[3]:

$$\mathsf{out}(c,u).Q \mid \mathsf{in}(c,x).P \;\to\; Q \mid P\{x \mapsto u\} \qquad \text{(Red I/O)}$$
$$\mathsf{let}\ \overline{x} = \overline{t}\ \mathsf{in}\ P\ \mathsf{else}\ Q \;\to\; P\{\overline{x} \mapsto \mathsf{choice}[\overline{u}_1, \overline{u}_2]\} \quad \text{if } \mathsf{fst}(\overline{t}) \Downarrow \overline{u}_1 \text{ and } \mathsf{snd}(\overline{t}) \Downarrow \overline{u}_2 \qquad \text{(Red Fun 1)}$$
$$\mathsf{let}\ \overline{x} = \overline{t}\ \mathsf{in}\ P\ \mathsf{else}\ Q \;\to\; Q \qquad \text{if } \mathsf{fst}(\overline{t}) \not\Downarrow \text{ and } \mathsf{snd}(\overline{t}) \not\Downarrow \qquad \text{(Red Fun 2)}$$

A bi-process execution step thus consists of strongly synchronized execution steps of its two projections. In particular, a conditional in a bi-process succeeds (resp. fails) if it succeeds (resp. fails) for both of its projections. Formally, we have that $P \to P'$ implies $\mathsf{fst}(P) \to \mathsf{fst}(P')$ and $\mathsf{snd}(P) \to \mathsf{snd}(P')$. However, it could be that some execution step of $\mathsf{fst}(P)$ (resp. $\mathsf{snd}(P)$) cannot be obtained in this way from an execution step of P. In fact, [20, Theorem 1] shows that the two projections of a bi-process P are observationally equivalent if, for all C, all reducts of $C[P]$ are uniform in the following sense:

Definition 20 ([20])**.** *A bi-process P is* uniform *if for all reductions $\mathsf{fst}(P) \to P_1$, there exists P' such that $P \to P'$ and $\mathsf{fst}(P') = P_1$, and symmetrically for $\mathsf{snd}(P)$.*

From now on, we say that a bi-process is *diff-equivalent* when it satisfies the condition of [20, Theorem 1]. By extension, we say that the two projections of a bi-process are diff-equivalent when the bi-process is.

Diff-equivalence verification in ProVerif*.* The next contribution of [20] is to show that diff-equivalence can be automatically verified in ProVerif by adapting its Horn clause encoding and resolution algorithm to bi-processes. We will not recall the encoding and its modifications in detail here, but only present some key ideas at a high level. In the single-process case, a unary predicate $\mathsf{att}(\cdot)$ is used to encode that the attacker knows some message. The attacker's capabilities are expressed as Horn clauses involving this predicate, *e.g.* the ability to encrypt is translated as

$$\forall x \forall y.\ \mathsf{att}(x) \wedge \mathsf{att}(y) \Rightarrow \mathsf{att}(\mathsf{enc}(x,y)).$$

Then, a process fragment of the form $\mathsf{in}(c,x).\mathsf{let}\ y = \mathsf{dec}(x,k)\ \mathsf{in}\ \mathsf{out}(c',t)$, where t is a constructor term with free variable y, is encoded as

$$\forall y.\ \mathsf{att}(\mathsf{enc}(y,k)) \Rightarrow \mathsf{att}(t).$$

Note that the variable x does not appear in the clause, but has been refined into $\mathsf{enc}(y,k)$ as part of the translation.

Consider now the analogue bi-process fragment

$$\mathsf{in}(c,x).\mathsf{let}\ y = \mathsf{dec}(x,k)\ \mathsf{in}\ \mathsf{out}(c', \mathsf{choice}[t_1,t_2]).$$

It corresponds to two processes, each of which may receive a message and attempt to decrypt it using k. Upon success, the processes output t_1 and t_2 respectively, relying on the value y obtained

[3]We use our own notations here, taking advantage of the fact that our calculus is a simplification of the one used in [20]: in particular, channels are public constants and choice operators cannot be used in channel positions. We also use the notation $\mathsf{choice}[\cdot,\cdot]$ as in the tool ProVerif, rather than $\mathsf{diff}[\cdot,\cdot]$ as in [20].

from the respective decryptions. In ProVerif, this bi-process would translate into the Horn clause

$$\forall y_1 \forall y_2.\ \mathsf{att}'(\mathsf{enc}(y_1, k), \mathsf{enc}(y_2, k)) \Rightarrow \mathsf{att}'(t_1', t_2')$$

where t_1' (resp. t_2') is t_1 (resp. t_2) in which all the occurrences of y have been replaced by y_1 (resp. y_2). This time, a binary predicate $\mathsf{att}'(\cdot,\cdot)$ is used to encode the attacker's knowledge on each side of the bi-process run: the clause roughly says that if, at some point of the execution of the bi-process, the attacker can deduce (using the same derivation) a term of the form $\mathsf{enc}(y_1, k)$ from the left frame and a term $\mathsf{enc}(y_2, k)$ from the right frame, then he will learn t_1 on the left and t_2 on the right. The attacker's capabilities are also modified to encode the effect of the attacker's capabilities on each side of the bi-process run, *e.g.* for encryption:

$$\forall x_1 \forall x_2 \forall y_1 \forall y_2.\ \mathsf{att}'(x_1, x_2) \wedge \mathsf{att}'(y_1, y_2) \Rightarrow \mathsf{att}'(\mathsf{enc}(x_1, y_1), \mathsf{enc}(x_2, y_2)).$$

An extension of bi-processes. In the original notion of bi-processes [20], the two sides of a bi-process are isolated and can execute independently. However, the Horn clause encoding of bi-processes that is used for verification in ProVerif makes it easy to lift this restriction in a way that enables interesting new applications of diff-equivalence. Specifically, we introduce the possibility of binding two variables at once in a bi-process input, which we write $\mathsf{in}(c, \mathsf{choice}[x_1, x_2]).P$. For simplicity, we can consider that all inputs feature such choice variables, as the usual form $\mathsf{in}(c, x).P$ can be replaced by $\mathsf{in}(c, \mathsf{choice}[x_1, x_2]).P\{x \mapsto \mathsf{choice}[x_1, x_2]\}$. The intuitive semantics of such a construct is that x_1 is bound to the message received on the left side of the bi-process run, while x_2 is bound to the message received on the right. Formally, we change the (Red I/O) rule as follows:

$$\mathsf{out}(c, u).Q \mid \mathsf{in}(c, \mathsf{choice}[x_1, x_2]).P \quad \to \quad Q \mid P\{x_1 \mapsto \mathsf{fst}(u), x_2 \mapsto \mathsf{snd}(u)\}.$$

Crucially, each side of the bi-process will then have access to both x_1 and x_2, allowing a form of communication between the two sides.

With this modification, [20, Theorem 1] does not hold anymore. In fact, $\mathsf{fst}(P)$ and $\mathsf{snd}(P)$ may not be ground processes when P uses choice variables in inputs, so that it does not even make sense to compare them for observational equivalence. More generally, we cannot talk in general of the projection of a bi-process execution: the fact that $P \to P'$ implies $\mathsf{fst}(P) \to \mathsf{fst}(P')$ becomes not only false but also ill-defined in general. However, the notion of uniformity is still meaningful, if properly adapted to be mathematically well-defined: a bi-process P is uniform if, *whenever* $\mathsf{fst}(P)$ *is ground*, it is the case that for all reductions $\mathsf{fst}(P) \to P_1$ there exists P' such that $P \to P'$ and $\mathsf{fst}(P') = P_1$, and symmetrically for $\mathsf{snd}(P)$. Furthermore, we shall see that it can be useful in cases where at least one projection of the executions of a bi-process is well-defined, as will be the case with our encoding of the notion of frame opacity through (extended) diff-equivalence.

Besides the problem of the new meaning of diff-equivalence, an important question is whether extended diff-equivalence can be verified automatically, and how. We claim that it is straightforward, as the Horn clause encoding of bi-processes already features what is needed for adequately encoding extended bi-processes, namely the duplicated input variables seen in the above example — accordingly, we only had to modify a few tenth of lines of the ProVerif tool to implement

our extension. A formal justification of this claim would require to adapt the long technical development of [20], and is thus out of the scope of the present paper. We simply illustrate the idea here by getting back to our running example illustrating the various Horn clause encodings, considering now the process fragment

$$\mathsf{in}(c, \mathsf{choice}[x_1, x_2]).\mathsf{let}\ y = \mathsf{dec}(x_1, k)\ \mathsf{in}\ \mathsf{out}(c', \mathsf{choice}[t_1, t_2])$$

where t_1 and t_2 are constructor terms with free variable y. This would be encoded as

$$\forall y_1 \forall x_2.\ \mathsf{att}'(\mathsf{enc}(y_1, k), x_2) \Rightarrow \mathsf{att}'(t_1', t_2')$$

where t_1' (resp. t_2') is t_1 (resp. t_2) in which all the occurrences of y have been replaced by y_1. This time, x_2 is not refined since the bi-process does not attempt to deconstruct it, on either side. The clause expresses that if the attacker can derive a term $\mathsf{enc}(y_1, k)$ on the left (regardless of what would be the corresponding term on the right) then he will learn t_1' on the left and t_2' on the right.

Encoding frame opacity through extended diff-equivalence. Using this extended notion of bi-process, we can now directly express frame opacity as the diff-equivalence of a bi-process. This bi-process will have $\mathcal{M}_{\Pi,\overline{id}}$ as its first projection. Its second projection should replace each message output with its idealization, so that diff-equivalence of P^{ideal} implies $\Phi \sim \Phi_{\mathrm{ideal}}(\mathsf{ta})$ for any ta that is executable by $\mathcal{M}_{\Pi,\overline{id}}$ with Φ as the resulting frame. In itself, this can be achieved easily, as it suffices to create new names to use as values for $\mathcal{X}^n$ variables and use appropriate input variables for the $\mathcal{X}^i$ variables. The difficulty lies with tests (and computation failures) which need to be carefully used to obtain a correct encoding of frame opacity.

First, we need to ensure that any reduction of $\mathcal{M}_{\Pi,\overline{id}}$ (in any context) can be obtained as the projection of a reduction of P^{ideal} (in the same context). In other words, the second projection of any test should agree with its first projection, which corresponds to a normal execution of $\mathcal{M}_{\Pi,\overline{id}}$. This would not hold in general if we performed the same test on the idealizations which are computed in the second projection. We obtain the desired behaviour using our extension of bi-processes, by performing tests using only left-hand side input variables.

Second, we need to ensure that computation failures that occur while computing idealizations result in a diff-equivalence failure. This is necessary to obtain a match with frame opacity, which requires that idealizations are well-defined even when destructors are involved. Hence, when the bi-process computes idealized values in its second projection, a failsafe computation should happen in the first projection. Moreover, idealizations should be computed using the values of the input variables from the right side of the bi-process execution, in line with the definition of idealization, *i.e.* Definition 13.

Before proving more formally that our translation is adequate, let us illustrate it on our running example, *i.e.* Example 8. We first give in Figure 4 the description of the protocol in ProVerif syntax. This syntax is actually very close to the one we introduced in Section 2. The main difference is the fact that ProVerif relies on types, and here any name or variable is given the generic type bitstring. Next, we show in Figure 5 the bi-process expressing frame opacity as described above, using the syntaxic idealisation (Section 4.2.1). Note that, when decrypting the first input of the initiator role, the variable `x` is used, corresponding to the left side of the bi-

```
let I (k:bitstring) =
  new nI:bitstring;
  out(ci, nI);
  in(ci, x:bitstring);
  let (=nI, xnr:bitstring) = dec(x, k)
      in
  out(ci, enc((xnr,nI),k)).

let R (k:bitstring) =
  new nR:bitstring;
  in(cr, ynI:bitstring);
  out(cr, enc((ynI, nR), k));
  in(cr, y:bitstring);
  let (=nR,=ynI) = dec(y,k) in
  out(cr, ok).

let FH = ! new k:bitstring; ! (I(k) | R(k)).
```

Fig. 4. Our running example (Feldhofer) using ProVerif's syntax

process execution. The variable `xid`, correspond to the right (idealized) side, is not used in that case because this input is not used in idealizations. However, the variable `ynIid` corresponding to an idealized input in the responder role is used in the first output. In this example, the idealisation operator does not contain any destructor, hence the computation of the idealisation can never fail. If destructors were present, they would be computed using let constructs inside the right component of the $\mathsf{choice}[\cdot,\cdot]$ operator[4] in output, so that their failure would result in a non-equivalence.

We now conclude with a formal correctness argument.

Proposition 2. *Let* ideal *be an idealisation operator, and note* P^{ideal} *the corresponding encoding of a process* P *into a bi-process. Assume that, for all* C *and* B *such that* $C[P^{\mathsf{ideal}}] \to^* B$, B *is uniform. Then* P *satisfies frame opacity wrt.* ideal.

Proof. Assume, by contradiction, that there exists an execution $(P;\varnothing) \xrightarrow{\mathsf{ta}} (Q;\Phi)$ with $\Phi \not\sim \Phi_{\mathrm{ideal}}(\mathsf{ta})$. Then there exists a test $T = \big(\mathsf{let}\ \overline{y} = \overline{u}\ \mathsf{in}\ \mathsf{out}(o,\mathsf{ok})\big)$ with $fv(T) = fv(\overline{u}) \subseteq dom(\Phi)$ such that $T\Phi$ can perform an output (after the successful evaluation of its let) while $T\Phi_{\mathrm{ideal}}(\mathsf{ta})$ cannot. Further, we can construct in a standard way[5] from ta a context C_{ta} such that $C_{\mathsf{ta}}[P] \to^* T\Phi$, with only communications and internal reductions of P in that reduction. Because of this, the reduction can be lifted to the idealized bi-process as follows, by definition of P^{ideal}:

$$C_{\mathsf{ta}}[P^{\mathsf{ideal}}] \to^* T' = T\{w \mapsto \mathsf{choice}[\Phi(w), \Phi_{\mathrm{ideal}}(\mathsf{ta})(w)]\}_{w\in dom(\Phi)}$$

We have obtained our contradiction, since T' is not uniform: indeed, the reduction step $\mathsf{fst}(T') \to \mathsf{out}(c,\mathsf{ok})$ cannot be obtained as a projection of a reduction of T', which in fact cannot perform any reduction at all. □

[4]This is not possible with the theoretical notion of bi-process, where $\mathsf{choice}[\cdot,\cdot]$ operators can only contain terms, which cannot contain let constructs. However, it is available in (vanilla) ProVerif as syntactic sugar: it is equivalent to performing the let outside the $\mathsf{choice}[\cdot,\cdot]$ with a dummy first projection, which is exactly what we need.

[5]Define $C_{\mathsf{ta}}[\bullet] = (\bullet \mid T_{\mathsf{ta}})$ with $T_\epsilon = T$, $T_{\tau.\mathsf{ta}} = T_{\mathsf{ta}}$, $T_{\mathsf{out}(c,w).\mathsf{ta}} = \mathsf{in}(c,w).T_{\mathsf{ta}}$ and $T_{\mathsf{in}(c,R).\mathsf{ta}} = \mathsf{out}(c,R).T_{\mathsf{ta}}$. In full details, the last case should include the creation of names that are used in R but not previously in the trace.

```
let SYSTEM =
( !
  new k : bitstring;
     !
      ((
        new nI: bitstring;
        out(ci, nI);
        in(ci, choice[x,xid]: bitstring);
        let ((=nI,xnr: bitstring)) = dec(x,k) in
        new hole__xnr_I_0: bitstring;
        new hole__k_I_1: bitstring;
        out(ci, choice[enc((xnr,nI),k),enc((hole__xnr_I_0,nI),hole__k_I_1)])
      )|(
        in(cr, choice[ynI,ynIid]: bitstring);
        new nR: bitstring;
        new hole__k_R_2: bitstring;
        out(cr, choice[enc((ynI,nR),k),enc((ynIid,nR),hole__k_R_2)]);
        in(cr, choice[y,yid]: bitstring);
        let ((=nR,=ynI)) = dec(y,k) in
        out(cr, ok)
      ))
).
```

Fig. 5. ProVerif file checking frame opacity generated by UKano (Feldhofer)

Practical application. Our tool UKano automatically constructs the bi-process described above from a description of the protocol, and calls the extension of ProVerif in order to check frame opacity. Until this extension is integrated in the next release of ProVerif, the source files of this slight extension of ProVerif are distributed with UKano [51]. Out tool does not require the user to input the idealisation function. Instead, a default idealisation is extracted from the protocol's outputs. The user is informed about this idealisation, and if he wants to, he can bypass it using annotations or choose another heuristic to build idealisation operators. In practice, this is rarely necessary; we provide more details about this in Section 5.3. Also note that, although ProVerif does not support the repetition operator i, we can over-approximate the behaviours of protocols using it by replacing occurrences of i with ! before checking frame opacity.

5.2. Well-authentication

We explain below how to check condition *(i)* of well-authentication (see Definition 19). Once that condition is established, together with frame opacity, we shall see that condition *(ii)* is actually a consequence of a simple assumption on the choice of idealisation, which is always guaranteed when using UKano. This result is established relying on the sub-conditions that have been proved to be sufficient in Lemma 1.

5.2.1. Condition (i)

Condition *(i)* of well-authentication is basically a conjunction of reachability properties, which can be checked in ProVerif using correspondence properties [3]. To each syntactical output $\mathsf{out}(c, m_0)$ of the initiator role, we associate an event, namely $\mathtt{Iout_i}(\overline{k}_I, \overline{n}_I, \overline{m}_I)$ which uniquely identifies the action. We have that:

– $\overline{k}_I$ are the identity parameters used in the intiator role;

– $\overline{n}_I$ are the sessions parameters; and
– $\overline{m}_I$ are the messages inputted and outputted so far in this role including m_0.

Such an event is placed just before the action $\mathsf{out}(c, m_0)$. We proceed similarly for each syntactical input $\mathsf{in}(c, m_0)$ putting the event $\mathtt{Iin_i}(\overline{k}_I, \overline{n}_I, \overline{m}_I)$ just after the corresponding input. Lastly, we also apply this transformation on the responder role using events of the form $\mathtt{Rout_i}(\overline{k}_R, \overline{n}_R, \overline{m}_R)$ and $\mathtt{Rin_i}(\overline{k}_R, \overline{n}_R, \overline{m}_R)$. To be able to express condition (i) relying on events, we need to consider some events that will be triggered when conditional are passed successfully. Therefore, we add events of the form $\mathtt{Itest_i}(\overline{k}_I, \overline{n}_I, \overline{m}_I)$ (resp. $\mathtt{Rtest_i}(\overline{k}_R, \overline{n}_R, \overline{m}_R)$) at the beginning of each then branch of the initiator (resp. responder) role.

For each conditional of the protocol, we first check if the simple syntactical definition of *safe* conditionals holds (see Definition 17). If it is the case we do nothing for this conditional. Otherwise, we need to check condition (i) of well-authentication. This condition can be easily expressed as a correspondence property relying on the events we have introduced. Let $\overline{k}_I = (k_I^1, \ldots, k_I^p)$ and $\overline{k}_R = (k_R^1, \ldots, k_R^q)$. We denote $\overline{x}_I = (x_{k_I^1}, \ldots, x_{k_I^p})$ and $\overline{x}_R = (x_{k_R^1}, \ldots, x_{k_R^q})$. Note that when $\overline{k}_I \cap \overline{k}_R \neq \varnothing$ (shared case), we have also that $\overline{x}_I \cap \overline{x}_R \neq \varnothing$ and the correspondence property (see below) will therefore allow us to express duality of the two underlying agents.

For instance, given a conditional of the initiator role tagged with event $\mathtt{Itest_i}(\overline{k}_I, \overline{n}_I, \overline{m}_I)$, we express as a correspondence property the fact that if the conditional is positively evaluated, then the involved agent must be associated to a dual agent as follows:

1. when the event $\mathtt{Itest_i}(\overline{x}_I, \overline{y}_I, (z_1, \ldots, z_\ell))$ is fired,
2. there must be a previous event $\mathtt{Iin_i}(\overline{x}_I, \overline{y}_I, (z_1, \ldots, z_\ell))$ (the one just before the conditional),
3. and a previous event $\mathtt{Rout_j}(\overline{x}_R, \overline{y}_R, (z_1, \ldots, z_\ell))$ (the one corresponding to the output that fed the input $\mathtt{Iin_i}$ in an honest execution),
4. and a previous event $\mathtt{Rin_j}(\overline{x}_R, \overline{y}_R, (z_1, \ldots, z_{\ell-1}))$ (the one just before the output $\mathtt{Rout_j}$), etc.

Note that by using the same variables $(z_1, \ldots, z_\ell)$ in both the intiator and responder roles, we express that the messages that are outputted and inputted are equal modulo the equational theory E. We provide in Figure 6 the process obtained by applying the transformation on the Feldhofer protocol (Example 8). In Figure 7, we show the ProVerif queries we have to consider to check condition (i) on the two conditionals.

Some practical considerations. In our tool, safe conditionals are not automatically identified. Actually, the tool lists all conditionals and tells which ones satisfy condition (i) of well-authentication. The user can thus easily get rid of the conditionals that he identifies as safe. Furthermore, the structure of the ProVerif file produced by UKano makes it easy for the user to remove the proof obligations corresponding to safe conditionals. To obtain more precise encodings once the translation in Horn clauses is performed by ProVerif, we sometimes push the creation of session parameters (i.e. instructions of the form $\mathsf{new}\ \mathtt{nI}$). Therefore, in order to ensure the existence of at least one session parameter in each event, we systematically introduce a fresh session parameter `sessI` (resp. `sessR`) which is is generated at the beginning of the initiator (resp. responder) role. Such parameters are systematically added in the events, and since they do not occur in the messages exchanged during the protocol execution, there is no need to push them.

```
let SYSTEM = ( !  new k : bitstring; !((
       new nI:bitstring;
       event Iout_1(k,nI,nI); out(ci, nI);
       in(ci, x:bitstring); event Iin_1(k,nI,nI,x)
       let ((=nI,xnr:bitstring)) = dec(x,k) in event Itest_1(k,nI,nI,x);
       event Iout_2(k,nI,nI,x,enc((xnr,nI),k)); out(ci, enc((xnr,nI),k))
      )|(
       new nR: bitstring;
       in(cr, ynI: bitstring); event Rin_1(k,nR,ynI);
       event Rout_1(k,nR,ynI,enc((ynI,nR),k)); out(cr, enc((ynI,nR),k));
       in(cr, y:bitstring); event Rin_2(k,nR,ynI,enc((ynI,nR),k),y);
       let ((=nR,=ynI)) = dec(y,k) in event Rtest_1(k,nR,ynI,enc((ynI,nR),k),y);
       event Rout_2(k,nR,ynI,enc((ynI,nR),k),y,ok); out(cr, ok)
      ))).
```

Fig. 6. Process modelling the Feldhofer protocol with events

```
query x:bitstring,
      y1:bitstring, y2:bitstring,
      z1:bitstring, z2:bitstring;
   (event(Itest_1(x,y1,z1,z2))  ==>
   (event(Iin_1(x,y1,z1,z2))  ==>
   (event(Rout_1(x,y2,z1,z2))  ==>
   (event(Rin_1(x,y2,z1))  ==>
   (event(Iout_1(x,y1,z1))))))).
```

```
query x:bitstring, y1:bitstring,
      y2:bitstring, z1:bitstring,
      z2:bitstring, z3:bitstring;
   (event(Rtest_1(x,y2,z1,z2,z3))  ==>
   (event(Rin_2(x,y2,z1,z2,z3))  ==>
   (event(Iout_2(x,y1,z1,z2,z3))  ==>
   (event(Iin_1(x,y1,z1,z2))  ==>
   (event(Rout_1(x,y2,z1,z2))  ==>
   (event(Rin_1(x,y2,z1))  ==>
   (event(Iout_1(x,y1,z1)))))))))).
```

Fig. 7. ProVerif queries for checking condition (i) on the Feldhofer protocol

Note that, for some examples, we also verified condition (i) of well-authentication using Tamarin by encoding the queries described above as simple lemmas. In our case, one of the most important advantage of Tamarin over ProVerif is its capability to model the repetition operator ! and thus protocols for which a role executes its sessions in sequence. Relying on Tamarin, we were thus able to verify condition (i) for protocols that ensure unlinkability when sessions are running sequentially but not when they are running concurrently, *e.g.* we automatically verified the toy example described in Example 9.

5.2.2. Condition (ii) - shared case

To verify Condition (ii) of well-authentication, we rely on Lemma 1 which provides two sufficient sub-conditions. Condition (a) of Lemma 1 can be checked manually; UKano leaves it to the user. Condition (b) may in general be very difficult to verify. While it is surely possible to reduce the verification of this sub-condition to classical reachability properties verifiable in ProVerif, we prefer to give a more direct verification technique.

Indeed, once frame opacity is known to hold, condition (b) actually follows immediately from simple properties of the idealisation function, since checking that honest outputs cannot be confused in executions of $\mathcal{M}_{\Pi,\overline{id}}$ is equivalent to checking that they cannot be confused in idealised executions. Often, the idealisation function uses only function symbols that do not occur in E and such that at least one session variable $x^{\mathsf{n}} \in \mathcal{X}^{\mathsf{n}}$ occurs in ideal(ℓ) for each honest output

label ℓ. Checking that the idealisation function enjoys these properties is straightforward. Let us now show that it implies condition (b) of Lemma 1.

Proposition 3. *Let* $\Pi = (\overline{k}, \overline{n}_I, \overline{n}_R, \dagger_I, \dagger_R, \mathcal{I}, \mathcal{R})$ *be a protocol such that* $fn(\mathcal{I}) \cap fn(\mathcal{R}) \neq \varnothing$ *(shared case). Consider an idealisation operator* $\mathrm{ideal}(\cdot)$ *such that, for any label* $\ell \in \mathcal{L}$ *occurring in the honest execution of* Π*, some name variable* $x \in \mathcal{X}^{\mathsf{n}}$ *appears in* $\mathrm{ideal}(\ell)$ *in a position only under symbols* $\mathsf{f} \in \Sigma_c$ *that do not occur in equations of* E*. If* Π *satisfies frame opacity for the idealised operator* $\mathrm{ideal}(\cdot)$ *then condition (b) of Lemma 1 holds.*

Proof. Consider an execution ta of $\mathcal{M}_{\Pi,\overline{id}}$ where agent a_1 performs an output with label ℓ and handle w_1, and agent $a_2 \neq a_1$ performs another output with label ℓ and handle w_2. We assume that ℓ occurs in the honest execution of Π and we note ϕ the resulting frame from the above execution. Assume, for the sake of contradiction, that $\phi(w_1) =_{\mathsf{E}} \phi(w_2)$. Since the protocol ensures frame opacity for the idealised operator $\mathrm{ideal}(\cdot)$, we deduce that $\Phi^{\mathrm{fr}}_{\mathrm{ideal}}(\mathsf{ta})(w_1) =_{\mathsf{E}} \Phi^{\mathrm{fr}}_{\mathrm{ideal}}(\mathsf{ta})(w_2)$. By hypothesis, some name variable $x_1 \in \mathcal{X}^{\mathsf{n}}$ occurs in $\mathrm{ideal}(\ell)$ in a position which (even after a substitution) cannot be erased by the equational theory nor the computation relation. In other words we have that $\mathrm{fr}(a_1, x^{\mathsf{n}})$ occurs in $\Phi^{\mathrm{fr}}_{\mathrm{ideal}}(\mathsf{ta})(w_1)$, and similarly $\mathrm{fr}(a_2, x^{\mathsf{n}})$ occurs in $\Phi^{\mathrm{fr}}_{\mathrm{ideal}}(\mathsf{ta})(w_2)$, at the same position under non-malleable constructor symbols only. Since we have assumed (in Section 2.1) that our equational theory is non-degenerate, this implies that $\mathrm{fr}(a_1, x^{\mathsf{n}}) =_{\mathsf{E}} \mathrm{fr}(a_2, x^{\mathsf{n}})$ and contradicts the injectivity of fr. □

5.3. The tool UKano

As mentioned earlier, the tool UKano [51] automatises the encodings described in this section. It takes as input a ProVerif model specifying the protocol to be verified (and the identity names $\overline{id}$) and returns:

1. whether frame opacity could be established or not: in particular, it infers an idealisation operator that, when in the shared case, satisfies the assumptions of Proposition 3;
2. and the list of conditionals for which condition (i) of well-authentication holds.

If frame opacity holds and condition (i) of well-authentication holds for all conditionals — possibly with some exceptions for conditionals the user can identify as safe — then the tool concludes that the protocol given as input ensures unlinkability and anonymity w.r.t. $\overline{id}$. Note that the tool detects whether $fn(\mathcal{I}) \cap fn(\mathcal{R}) = \varnothing$ or not and adapts the queries for verifying item (i) of well-authentication accordingly. Our tool uses heuristics to build idealised operators that always satisfy the assumptions of Proposition 3. Actually, three different heuristics have been implemented.

Syntaxic heuristic. The syntaxic heuristic fully adopts the canonical syntactical construction from Section 4.2 (and displays a warning message when in the shared case, since all requirements are not met in this case). It can be enabled using the option `-ideal-syntaxic`.

Semantic heuristic. The semantic heuristic (enabled with the option `-ideal-semantic`) follows the semantical construction from Section 4.2 with only tuples identified as transparent. Roughly, idealisation of a tuple is a tuple of idealisations of the corresponding sub-terms and idealisation of any other term is a fresh session variable in $\mathcal{X}^{\mathsf{n}}$. Such an idealised operator is much less precise (*i.e.* may lead to more false negatives) but since idealised messages are much simpler, it allows better performance when it works.

Quasi-syntaxic heuristic. This heuristic follows the canonical syntactical construction described in Section 4.2 except that sub-terms having a function symbol at top-level that is involved in the equational theory will be replaced by a fresh session name in order to comply with hypothesis of Proposition 3. This is the default heuristic in UKano.

Finally, the user can also define its own idealisations and the tool UKano will check that assumptions of Proposition 3 are satisfied when in the shared case.

At a technical level, we built UKano on top of ProVerif. We only re-used the lexer, parser and AST of ProVerif and build upon those a generator and translator of ProVerif models implementing our sufficient conditions via the above encodings. This effort represents about 2k OCaml LoC. The official page of the tool UKano with distributed releases of the tool can be found at `http://projects.lsv.ens-cachan.fr/ukano/`. We also distribute ProVerif v1.97 modified for handling extended diff-equivalence (see Section 5.1). The difference between our modified version of ProVerif v1.97 and the original one is about 60 lines of code.

6. Case studies

In this section we apply our verification method to several case studies. We rely on our tool UKano to check whether the protocol under study satisfies frame opacity and well-authentication as defined in Section 4. We also discuss some variations of the protocols to examine how privacy is affected. Remind that if privacy can be established for concurrent sessions (*i.e.* $\dagger_I = \dagger_R = !$) then it implies privacy for all other scenarios as well, *i.e.* when $\dagger_I, \dagger_R \in \{\mathsf{i}, !\}$. We thus model protocols with concurrent sessions and discuss alternative scenarios only when attacks are found. The source code of our tool and material to reproduce results can be found at

`http://projects.lsv.ens-cachan.fr/ukano/`.

All case studies discussed in this section except two (*i.e.* DAA in Section 6.4 and ABCDH in Section 6.5) have been automatically verified using our tool UKano without any manual effort. We discuss little manual efforts needed to conclude for DAA and ABCDH in the dedicated sections. We used UKano `v0.5` based on ProVerif v1.97 on a computer with following specifications:

- OS: Linux 3.10-2-amd64 #1 SMP Debian 3.10.5-1x86_64 GNU/Linux
- CPU / RAM: Intel(R) Xeon(R) CPU X5650 @ 2.67GHz / 47GO

6.1. Hash-Lock protocol

We consider the Hash-Lock protocol as described in [43]. This is an RFID protocol that has been designed to achieve privacy even if no formal proof is given. We suppose that, initially, each tag has his own key k and the reader maintains a database containing those keys. The protocol relies on a hash function, denoted h, and can be informally described as follows.

$$\begin{aligned} \mathsf{Reader} \to \mathsf{Tag} : &\quad n_R \\ \mathsf{Tag} \to \mathsf{Reader} : &\quad n_T,\ \mathsf{h}(n_R, n_T, k) \end{aligned}$$

This protocol falls into our generic class of 2-party protocols in the shared case, and frame opacity and well-authentication can be automatically established in less than 0.01 second. We

can therefore conclude that the protocol preserves unlinkability (note that anonymity does not make sense here). Actually, all implemented heuristics were able to successfully establish frame opacity automatically.

figure

6.2. LAK protocol

We present an RFID protocol first introduced in [44], and we refer to the description given in [52]. To avoid traceability attacks, the main idea is to ask the tag to generate a nonce and to use it to send a different message at each session. We suppose that initially, each tag has his own key k and the reader maintains a database containing those keys. The protocol is informally described below (h models a hash function). In the original version (see *e.g.* [52]), in case of a successful execution, both parties update the key k with $\mathsf{h}(k)$ (they always store the last two keys). Our framework does not allow one to model protocols that rely on a mutable state. Therefore, we consider here a version where the key is not updated at the end of a successful execution allowing the key k to be reused from one session to another. This protocol lies in the shared case since the identity name k is used by the reader and the tag.

$$\begin{aligned}
\mathsf{Reader} \to \mathsf{Tag} &: r_1 \\
\mathsf{Tag} \to \mathsf{Reader} &: r_2,\ \mathsf{h}(r_1 \oplus r_2 \oplus k) \\
\mathsf{Reader} \to \mathsf{Tag} &: \mathsf{h}(\mathsf{h}(r_1 \oplus r_2 \oplus k) \oplus k \oplus r_1)
\end{aligned}$$

Actually, this protocol suffers from an authentication attack. The protocol does not allow the reader to authenticate the tag. This attack can be informally described as follows (and already exists on the original version of this protocol). By using algebraic properties of $\oplus$, an attacker can impersonate a tag by injecting previously eavesdropped messages. Below, $I(\mathsf{A})$ means that the attacker plays the role A.

$$\begin{aligned}
I(\mathsf{Reader}) \to \mathsf{Tag} &: r_1 \\
\mathsf{Tag} \to \mathsf{Reader} &: r_2,\ \mathsf{h}(r_1 \oplus r_2 \oplus k) \\
\mathsf{Reader} \to \mathsf{Tag} &: r_1' \\
I(\mathsf{Tag}) \to \mathsf{Reader} &: r_2^I,\ \mathsf{h}(r_1 \oplus r_2 \oplus k) \\
\mathsf{Reader} \to \mathsf{Tag} &: \mathsf{h}(\mathsf{h}(r_1 \oplus r_2 \oplus k) \oplus k \oplus r_1')
\end{aligned}$$

where $r_2^I = r_1' \oplus r_1 \oplus r_2$, thus $\mathsf{h}(r_1 \oplus r_2 \oplus k) =_{\mathsf{E}} \mathsf{h}(r_1' \oplus r_2^I \oplus k)$.

Due to this, the protocol does not satisfy our well-authentication requirement even with sessions in sequence for Tag and Reader. Indeed, the reader can end a session with a tag whereas the tag has not really participated to this session. In other words, the reader passes a test (which does not correspond to a safe conditional) with success, and therefore performs a τ_{then} action whereas it has not interacted honestly with a tag. Actually, this trace can be turned into an attack against the unlinkability property (for any combination of $\dagger_I, \dagger_R \in \{\mathsf{i}, !\}$). Indeed, by continuing the previous trace, the reader can send a new request to the tag generating a fresh nonce r_1''. The attacker $I(\mathsf{Tag})$ can again answer to this new request choosing his nonce r_2'' accordingly, *i.e.* $r_2'' = r_1'' \oplus r_1 \oplus r_2$. This execution, involving two sessions of the reader talking to the same tag, cannot be mimicked in the single session scenario, and corresponds to an attack trace.

More importantly, this scenario can be seen as a traceability attack on the stateful version of the protocol leading to a practical attack. The attacker will first start a session with the targeted tag by sending it a nonce r_1 and storing its answer. Then, later on, he will interact with the reader as described in the second part of the attack scenario. Two situations may occur: either the interaction is successful meaning that the targeted tag has not been used since its last interaction with the attacker; or the interaction fails meaning that the key has been updated on the reader's side, and thus the targeted tag has performed a session with the reader since its last interaction with the attacker. This attack shows that the reader may be the source of leaks exploited by the attacker to trace a tag. This is why we advocate for the strong notion of unlinkability we used, taking into account the reader and considering it as important as the tag.

We may note that the same protocol was declared untraceable in [52] due to the fact that they have in mind a weaker notion of unlinkability. Actually, their notion captures the intuitive notion that a tag is untraceable if for any execution in which two actions are performed by the same tag, there is another execution indistinguishable from the original one in which the actions have been performed by two different tags. We may note that in the attack scenario described above, the tag in itself does not leak anything but the reader does, explaining why this weak notion of untraceability missed this attack.

Now, to avoid the algebraic attack due to the properties of the xor operator, we may replace it by the pairing operator. The resulting protocol is a 2-party protocol that falls into our class, and for which frame opacity and well-authentication can be established (with concurrent sessions) using UKano (any heuristic is suitable for that). Therefore, Theorem 1 allows us to conclude that it preserves unlinkability.

6.3. BAC protocol and some others

An e-passport is a paper passport with an RFID chip that stores the critical information printed on the passport. The International Civil Aviation Organization (ICAO) standard [1] specifies several protocols through which this information can be accessed. Before executing the Basic Access Control (BAC) protocol, the reader optically scans a weak secret from which it derives two keys k_E and k_M that are then shared between the passport and the reader. Then, the BAC protocol establishes a key seed from which two sessions keys are derived. The session keys are then used to prevent skimming and eavesdropping on subsequent communications.

In [7], two variants of the BAC protocol are described and analysed. We refer below to these two variants as the French version and the United Kingdom (U.K.) version. The U.K. version is claimed unlinkable (with no formal proof) whereas an attack is reported on the French version. We first give an informal description of the BAC protocol using Alice & Bob notation:

$$\begin{aligned}
&\mathsf{Tag} \to \mathsf{Reader} : n_T \\
&\mathsf{Reader} \to \mathsf{Tag} : \{n_R, n_T, k_R\}_{k_E}, \mathsf{mac}(\{n_R, n_T, k_R\}_{k_E}, k_M) \\
&\mathsf{Tag} \to \mathsf{Reader} : \{n_T, n_R, k_T\}_{k_E}, \mathsf{mac}(\{n_T, n_R, k_T\}_{k_E}, k_M)
\end{aligned}$$

Then, to explain the difference between the two versions, we give a description of the passport's role in Figure 8. We do not model the getChallenge constant message that is used to initiate the protocol but it is clear this message does not play any role regarding the security of the protocol. We consider the signature given in Example 1 augmented with a function symbol mac

$$
\begin{aligned}
T(k_E, k_M) \;=\; & \mathsf{new}\, n_T.\mathsf{new}\, k_T.\mathsf{out}(c_T, n_T).\mathsf{in}(c_T, x).\\
& \quad \mathsf{let}\ x_E = \mathsf{proj}_1(x),\ x_M = \mathsf{proj}_2(x),\ z_{\mathsf{test}} = \mathsf{eq}(x_M, \mathsf{mac}(x_E, k_M))\ \mathsf{in}\\
& \qquad \mathsf{let}\ z'_{\mathsf{test}} = \mathsf{eq}(n_T, \mathsf{proj}_1(\mathsf{proj}_2(\mathsf{dec}(x_E, k_E))))\ \mathsf{in}\ \mathsf{out}(c_T, \langle m, \mathsf{mac}(m, k_M)\rangle)\\
& \qquad \mathsf{else}\ \mathsf{out}(\mathsf{error}_{\mathsf{Nonce}})\\
& \quad \mathsf{else}\ \mathsf{out}(\mathsf{error}_{\mathsf{Mac}})
\end{aligned}
$$

$$\text{where } m = \mathsf{enc}(\langle n_T, \langle \mathsf{proj}_1(\mathsf{dec}(x_E, k_E)), k_T\rangle\rangle, k_E).$$

Fig. 8. Description of the passport's role

of arity 2. This is a public constructor whose purpose is to model message authentication code, taking as arguments the message to authenticate and the mac key. There is no rewriting rule and no equation regarding this symbol. We also assume public constants to model error messages. The U.K. version of the protocol does not distinguish the two cases of failure, *i.e.* $\mathsf{error}_{\mathsf{Mac}}$ and $\mathsf{error}_{\mathsf{Nonce}}$ are the same constant, whereas the French version does. The relevant point is the fact that, in case of failure, the French version sends a different error message indicating whether the failure occurs due to a problem when checking the mac, or when checking the nonce. This allows the attacker to exploit this conditional to learn if the mac key of a tag is the one used in a given message $\langle m, \mathsf{mac}(m, k)\rangle$. Using this, he can very easily trace a tag T by first eavesdropping an honest interaction between the tag T and a reader.

The U.K. version of the BAC protocol is a 2-party protocol according to our definition. Note that since the two error messages are actually identical, we can merge the two let instructions, and therefore satisfy our definition of being a responder role. Then, we automatically proved frame opacity and well-authentication using UKano. It took less than 0.1 second independently of the chosen heuristic regarding frame opacity. Therefore, Theorem 1 allows us to conclude that unlinkability is indeed satisfied.

Regarding the French version of this protocol, it happens that the passport's role is neither an initiator role, nor a responder role according to our formal definition. Indeed, our definition of a role, and therefore of a 2-party protocol does not allow to model two sequences of tests that will output different error messages in case of failure. As illustrated by the attack on the French version of the BAC protocol, imposing this syntactic condition is actually a good design principle w.r.t. unlinkability.

Once the BAC protocol has been successfully executed, the reader gains access to the information stored in the RFID tag through the Passive and Active Authentication protocols (PA and AA). They are respectively used to prove authenticity of the stored information and prevent cloning attacks, and may be executed in any order. A formal description of these protocols is available in [6]. These two protocols also fall into our class and our conditions can be checked automatically both for unlinkability and anonymity properties. We can also use our technique to analyse directly the three protocols together (*i.e.* the U.K. version of the BAC together with the PA and AA protocols in any order). We analysed both orders, *i.e.* BAC followed by PA, and then AA, as well as BAC following by AA, and then PA. We establish unlinkability and anonymity w.r.t. all private data stored in the RFID chip (name, picture, etc.). UKano concludes within 1 second to establish both well-authentication and frame opacity (independently of the selected heuristic).

1. $\mathsf{Tag} \to \mathsf{Reader} : \{s_T\}_k$
2. $\mathsf{Reader} \to \mathsf{Tag} : \quad g^{n_R}$
3. $\mathsf{Tag} \to \mathsf{Reader} : g^{n_T}$
4. Both parties compute $G = \mathsf{gen}(s_T, g^{n_R n_T})$.
5. $\mathsf{Reader} \to \mathsf{Tag} : \quad G^{n'_R}$
6. $\mathsf{Tag} \to \mathsf{Reader} : G^{n'_T}$
7. Both parties compute $k' = G^{n'_R n'_T}$
8. $\mathsf{Reader} \to \mathsf{Tag} : \quad \mathsf{mac}(G^{n'_T}, k')$
9. $\mathsf{Tag} \to \mathsf{Reader} : \mathsf{mac}(G^{n'_R}, k')$

Fig. 9. PACE in Alice & Bob notation

6.4. PACE protocol

The Password Authenticated Connection Establishment protocol (PACE) has been proposed by the German Federal Office for Information Security (BSI) to replace the BAC protocol. It has been studied in the literature [18], [17], [27] but to the best of our knowledge, no formal proofs about privacy have been given. Similarly to BAC, its purpose is to establish a secure channel based on an optically-scanned key k. This is done in four main steps (see Figure 9):

- The tag chooses a random number s_T, encrypts it with the symmetric key k shared between the tag and the reader and sends the encrypted random number to the reader (message 1).
- Both the tag and the reader perform a Diffie-Hellman exchange (messages 2 & 3), and derive G from s_T and $g^{n_R n_T}$.
- The tag and the reader perform a Diffie-Hellman exchange based on the parameter G computed at the previous step (messages 5 & 6).
- The tag and the reader derive a session key k' which is confirmed by exchanging and checking the authentication tokens (messages 8 & 9).

Moreover, at step 6, the reader is not supposed to accept as input a message which is equal to the previous message that it has just sent.

To formalise such a protocol, we consider $\Sigma_c = \{\mathsf{enc}, \mathsf{dec}, \mathsf{dh}, \mathsf{mac}, \mathsf{gen}, \mathsf{g}, \mathsf{ok}\}$, and $\Sigma_d = \{\mathsf{neq}\}$. Except g and ok which are public constants, all these function symbols are public constructor symbols of arity 2. The destructor neq has already be defined in Example 4. The symbol dh is used to model modular exponentiation whereas mac will be used to model message authentication code. We consider the equational theory E defined by the following equations:

$$\mathsf{dec}(\mathsf{enc}(x,y),y) = x \qquad \mathsf{dh}(\mathsf{dh}(x,y),z) = \mathsf{dh}(\mathsf{dh}(x,z),y)$$

We consider the process $\mathcal{R}_{\mathsf{PACE}}$ as described in Figure 10. We do not detail the continuation R' and we omit trivial conditionals. The process modelling the role $\mathcal{I}_{\mathsf{PACE}}$ can be obtained in a similar way. Then, we consider $\Pi_{\mathsf{PACE}} = (k, (s_T, n_T, n'_T), (n_R, n'_R), !, !, \mathcal{I}_{\mathsf{PACE}}, \mathcal{R}_{\mathsf{PACE}})$ which falls into our generic class of 2-party protocols. Unfortunately, ProVerif cannot handle the equation above on the dh operator (due to some termination issues). Instead of that single equation, we consider the following equational theory that is more suitable for ProVerif:

$$\mathsf{dh}(\mathsf{dh}(\mathsf{g},y),z) = \mathsf{dh}(\mathsf{dh}(\mathsf{g},z),y) \qquad \mathsf{dh}(\mathsf{dh}(\mathsf{gen}(x_1,x_2),y),z) = \mathsf{dh}(\mathsf{dh}(\mathsf{gen}(x_1,x_2),z),y)$$

$$\begin{aligned}\mathcal{R}_{\mathsf{PACE}} := \; & \mathsf{in}(c_R, y_1).\\ & \mathsf{out}(c_R, \mathsf{dh}(\mathsf{g}, n_R)).\mathsf{in}(c_R, y_2).\\ & \mathsf{out}(c_R, \mathsf{dh}(G, n'_R)).\mathsf{in}(c_R, y_3).\\ & \mathsf{let}\ y_{\mathsf{test}} = \mathsf{eq}(\mathsf{yes}, \mathsf{neq}(y_3, \mathsf{dh}(G, n'_R)))\ \mathsf{in}\\ & \quad \mathsf{out}(c_R, \mathsf{mac}(y_3, k'));\\ & \quad \mathsf{in}(c_R, y_4).\\ & \quad \mathsf{let}\ y_5 = \mathsf{eq}(y_4, \mathsf{mac}(\mathsf{dh}(G, n'_R), k'))\ \mathsf{in}\ R'.\end{aligned}$$

where $G = \mathsf{gen}(\mathsf{dec}(y_1, k), \mathsf{dh}(y_2, n_R))$ and $k' = \mathsf{dh}(y_3, n'_R)$.

Fig. 10. Process $\mathcal{R}_{\mathsf{PACE}}$

This is sufficient for the protocol to work properly but it obviously lacks equations that the attacker may exploit.

First, we would like to highlight an imprecision in the official specification that may lead to practical attacks on unlinkability. As the specification seems to not forbid it, we could have assumed that the decryption operation in $G = \mathsf{gen}(\mathsf{dec}(y_1, k), \mathsf{dh}(y_2, n_R))$ is implemented in such a way that it may fail when the key k does not match with the key of the ciphertext y_1. In that case, an attacker could eavesdrop a first message $c^0 = \mathsf{enc}(s_T^0, k^0)$ of a certain tag T^0 and then, in a future session, it would let the reader optically scan a tag T but replace its challenge $\mathsf{enc}(s_T, k)$ by c^0 and wait for an answer of the reader. If it answers, he learns that the decryption did not fail and thus $k = k^0$: the tag T is actually T^0. We discovered this attack using our method since, in our first attempt to model the protocol, we modelled $\mathsf{dec}(\cdot,\cdot)$ as a destructor (that may fail) and the computation of G as an evaluation:

$$\mathsf{let}\ G = \mathsf{gen}(\mathsf{dec}(y_1, k), \mathsf{dh}(y_2, n_R))\ \mathsf{in}\ [...]$$

In order to declare the protocol well-authenticating, this conditional computing G which is not safe has to satisfy our requirement (see Definition 19). However, as witnessed by the attack scenario described above (the reflection attack), the condition actually fails to hold. Incidentally, the same attack scenario shows that the protocol does not ensure unlinkability (this scenario cannot be observed when interacting with $\mathcal{S}_\Pi$). Similarly to the attack on LAK, we highlight here the importance to take the reader into account and give it as much importance as the tag in the definition of unlinkability. Indeed, it is actually a leakage from the reader that allows an attacker to trace a specific tag.

Second, we now consider that decryption is a constructor, and thus cannot fail, an we report on an attack that we discovered using our method on some models of PACE found in the literature [18],[17],[27]. Indeed, in all those papers, the first conditional of the reader

$$\mathsf{let}\ y_{\mathsf{test}} = \mathsf{eq}(\mathsf{yes}, \mathsf{neq}(y_3, \mathsf{dh}(G, n'_R)))\ \mathsf{in}$$

is omitted. Then the resulting protocol is not well-authenticating. To see this, we simply have to consider a scenario where the attacker will send to the reader the message it has outputted at the previous step. Such an execution will allow the reader to execute its role until the end, and therefore execute τ_{then}, but the resulting trace is not an honest one. Again, this scenario can be turned into an attack against unlinkability as explained next. As before, an attacker could

eavesdrop a first message $c^0 = \mathsf{enc}(s_T^0, k^0)$ of a certain tag T^0. Then, in a future session, it would let the reader optically scan a tag T but replace its challenge $\mathsf{enc}(s_T, k)$ by c^0. Independently of whether k is equal to k^0 or not, the reader answers g^{n_R}. The attacker then plays the two rounds of Diffie-Hellman by reusing messages from the reader (he actually performs a reflection attack). More precisely, he replies with $g^{n_T} = g^{n_R}$, $G^{n'_T} = G^{n'_R}$ and $\mathsf{mac}(G^{n'_R}, k') = \mathsf{mac}(G^{n'_T}, k')$. The crucial point is that the attacker did not prove he knows k (whereas he is supposed to do so to generate G at step 4) thanks to the reflection attack that is not detected. Now, the attacker waits for the reader's answer. If it is positive (the process R' is executed), he learns that $k = k^0$: the tag T is actually the same as T^0.

Third, we turn to PACE as properly understood from the official specification: when the latter test is present and the decryption may not fail. In that case, we report on a new attack. UKano found that the last test of the reader violates well-authentication. This is the case for the following scenario: the message $\mathsf{enc}(s_T, k)$ sent by a tag $T(k, n_T)$ is fed to two readers $R(k, n_R^1), R(k, n_R^2)$ of same identity name. Then, the attacker just forwards messages from one reader to the other. They can thus complete the two rounds of Diffie-Hellman (note that the test avoiding reflection attacks holds). More importantly, the mac-key verification phase (messages 8 and 9 from Figure 9) goes well and the attacker observes that the last conditional of the two readers holds. This violates well-authentication but also unlinkability because the latter scenario cannot be observed at all in $\mathcal{S}_\Pi$: if the attacker makes two readers talk to each other in $\mathcal{S}_\Pi$ they cannot complete a session because they must have different identity names. In practice, this flaw seems hard to exploit but it could be a real privacy concern: if a tag initiates multiple readers, an attacker may learn which ones it had initiated by forwarding messages from one to another. It does not seem to be realistic in the e-passport scenario, but could be harmful in other contexts. It seems that, in the e-passport context, a modelling with sequential sessions would be more realistic. We come back to such a modelling at the end of this section.

Further, we propose a simple fix to the above attack by adding tags avoiding confusions between reader's messages and tag's messages. It suffices to replace messages 8 and 9 from Figure 9 by respectively $\mathsf{mac}(\langle \mathsf{c}_r, G^{n'_T} \rangle, k')$ and $\mathsf{mac}(\langle \mathsf{c}_t, G^{n'_R} \rangle, k')$ where $\mathsf{c}_r, \mathsf{c}_t$ are public constants, and adding the corresponding checks. Well-authentication can be automatically established using UKano in around 1 minute. Frame opacity can be automatically established using any heuristic described in Section 5.3. Heuristics producing more complex idealisations (*i.e.* the syntaxic one) are less efficient. Nevertheless, the tool concludes in at most 16 seconds. We thus conclude that PACE with tags preserves unlinkability in the model considered here.

6.5. Attributed-based authentication scenario using ABCDH protocol

Most authentication protocols are identity-based: the user needs to provide his identity and prove to the service provider he is not trying to impersonate somebody else. However, in many contexts, the service provider just needs to know that the user has some non-identifying attributes (*e.g.* age, gender, country, membership). For instance, a liquor shop just needs to have the proof that the user has the right to buy liquors (*i.e.* that he is old enough) and does not need to know the full identity of the user (*e.g.* as it is currently done when showing ID cards). Attribute-based authentication protocols solve this problem and allow a user to prove to another user, within a secure channel, that he has some attributes without disclosing its identity.

We used our method to automatically establish unlinkability of a typical use case of such a protocol taking part to the IRMA project[6]. We analysed a use case of the protocol ABCDH as defined in [5]. This protocol allows a smartcard C to prove to some Verifier V that he has the required attributes. The protocol aims at fulfilling this goal without revealing the identity of C to V or to anyone else. One of its goal is also to avoid that any other smartcard C' replays those attributes later on. The protocol should also ensure unlinkability of C. To the best of our knowledge, there was no prior formal analysis of that security property for this protocol.

The key ingredient of this protocol is *attribute-based credential* (ABC). It is a cryptographic container for attributes. In ABC, attributes are signed by some issuers and allow for *selective disclosure* (SD): it is possible to produce a zero-knowledge (ZK) proof revealing a subset of attributes signed by the issuer along with a proof that the selected disclosed attributes are actually in the credential. This non-interactive proof protocol can be bound to some fresh data to avoid replay attacks. We shall use the notation $\mathsf{SD}(\overline{a_i}; n)$ to denote the selective disclosure of attributes $\overline{a_i}$ bound to n. Note that $\mathsf{SD}(\varnothing; n)$ (no attribute is disclosed) still proves the existence of a credential. There are two majors ABC schemes: Microsoft U-Prove [47] and IBM's Idemix [25]. We decided to model IBM's scheme (since it is the one that is used in IRMA) following the formal model given in [26]. We may note that we consider here some privacy issues whereas the security analysis presented in [26] is dedicated to the analysis of some reachability properties. It involves complex cryptographic primitives (*e.g.* commitments, blind signature, ZK proofs) but ProVerif can deal with them all. In this scheme, each user has a master secret never revealed to other parties. Issuers issue credentials bound to the master secret of users (note that users are known to issuers under pseudonyms). A SD consists in a ZK proof bound to n proving some knowledge: knowledge of the master secret, knowledge of a credential bound to the master secret, knowledge that the credential has been signed by the given organisation, knowledge that the credential contains some given attributes.

We analyse the ABCDH [5] using the model of SD from [26] used in the following scenario:

- an organisation O_{age} issues credentials about the age of majority;
- an organisation $\mathsf{O}_{\text{check}}$ issues credentials giving the right to check the age of majority;
- a user C wants to watch a movie rated adult-only due to its violent contents; his has a credential from O_{age} with the attribute `adult`;
- a movie theatre V wants to verify whether the user has the right to watch this movie; it has a credential from $\mathsf{O}_{\text{check}}$ with the attribute `canCheckAdult`.

The scheme is informally given in Figure 11.

n_V, n_C and n are fresh nonces. Functions $\mathsf{f}_1/1$,$\mathsf{f}_2/1$ and $\mathsf{f}_3/2$ are independent hash functions; we thus model them as free constructor symbols. The construction $\mathsf{SD}(\cdot;\cdot)$ is not modelled atomically and follows [26] but we do not describe here its details. We note however that when V (respectively C) sends a $\mathsf{SD}(\cdot;\cdot)$, the corresponding message we do not detail here contains an identity-parameter $user_V$ (respectively $user_C$).

This is a 2-party protocol that falls into our class. Actually, we have that $user_V \in fn(\mathcal{I}) \cap \overline{k} \neq \varnothing$ and $user_C \in fn(\mathcal{R}) \cap \overline{k} \neq \varnothing$, but $fn(\mathcal{I}) \cap fn(\mathcal{R}) = \varnothing$ (non-shared case). The complete model of this protocol is quite complex and can be found in [51]. Frame Opacity can be automatically established using the syntaxic heuristic (see Section 5.3) in less than 40 seconds. The other

[6]For more information about IRMA ("I Reveal My Attributes"), see `https://www.irmacard.org`.

1. $V \to C : \mathsf{dh}(g, n_V), \mathsf{SD}(\mathtt{canCheckAdult}; \mathsf{f}_1(\mathsf{dh}(g, n_V)))$
2. $C \to V : \mathsf{dh}(g, n_C), \mathsf{SD}(\varnothing; \mathsf{f}_1(\mathsf{dh}(g, n_V)), \mathsf{dh}(g, n_C))$
3. $V \to C : \mathsf{enc}(\langle \mathtt{0x00}, \mathsf{ok} \rangle, k)$
4. $C \to V : \mathsf{enc}(\langle \mathtt{0x01}, \mathsf{ok} \rangle, k)$
5. $V \to C : \mathsf{enc}(\langle n, \mathtt{requestAdult} \rangle, k)$
6. $C \to V : \mathsf{enc}(\langle \mathtt{adult}, \mathsf{SD}(\mathtt{adult}; \mathsf{f}_3(n, \mathtt{seed})) \rangle, k)$

Fig. 11. ABCDH (where $seed = \mathsf{dh}(\mathsf{dh}(g, n_C), n_V)$ and $k = \mathsf{f}_2(seed)$)

heuristics were not enough precise to conclude (yielding negative results) showing the importance of having the choice between heuristics that are precise but less efficient and ones that are more efficient but less precise. Regarding well-authentication, due to the length of the protocol, the queries are also quite long. Because of the latter and the high complexity of the underlying term algebra, it required too much time for ProVerif to terminate. We addressed this performance issue by soundly splitting up big queries into smaller ones. This way, we successfully established well-authentitcation for this protocol within 3 hours.

6.6. DAA join & DAA sign

A Trusted Platform Module (TPM) is a hardware device aiming at protecting cryptographic keys and at performing some cryptographic operations. Typically, a user may authenticate himself to a service provider relying on such a TPM. The main advantage is to physically separate the very sensitive data from the rest of the system. On the downside however, such devices may be used by malicious agents to breach users' privacy by exploiting their TPMs. Direct Anonymous Attestation (DAA) protocols have been designed to let TPMs authenticate themselves whilst providing accountability and privacy.

In a nutshell, some issuers issue credentials representing membership to a group to the TPM using group signatures via the *DAA join* protocol. Those credentials are bound to the internal secret of the TPM that must remain unknown to the service provider. Then, when a TPM is willing to prove to a verifier its membership to a group, it uses the *DAA sign* protocol. We analysed the RSA-based DAA join and sign protocols as described in [50]. Both protocols rely on complex cryptographic primitives (*e.g.* blind signatures, commitments, and Zero Knowledge proofs) but ProVerif can deal with them all. Note that the authors of [50] have automatically established a game-based version of unlinkability of the combination of DAA Join and DAA Sign using ProVerif. We only provide an analysis of each protocol in isolation since the combination of the two protocols is a 3-party protocol.

6.6.1. DAA join

In the RSA-based DAA join protocol, the TPM starts by sending a credential request in the form of a commitment containing its internal secret, some session nonce and the public key of the issuer. The issuer then challenges the TPM with some fresh nonces encrypted asymmetrically with the public key of the TPM. After having received the expected TPM's answer, the issuer sends a new nonce as second challenge. To this second challenge, the TPM needs to provide a ZK proof bound to this challenge proving that he knows the internal secret on which the previous commitment was bound. Finally, after verifying this proof, the issuer blindly signs the commitment allowing the TPM to extract the required credential.

1. $\mathsf{TPM} \to \mathsf{Issuer} : N_I, U$
2. $\mathsf{Issuer} \to \mathsf{TPM} : \mathsf{penc}(n_e, n, \mathsf{pk}(sk_{\mathsf{TPM}}))$
3. $\mathsf{TPM} \to \mathsf{Issuer} : \mathsf{h}((U, n_e))$
4. $\mathsf{Issuer} \to \mathsf{TPM} : n_i$
5. $\mathsf{TPM} \to \mathsf{Issuer} : n_t, \mathsf{ZK}_{\mathsf{join}}((tsk, n_v), (zeta_I, N_I, U, (n_t, n_i)))$
6. $\mathsf{Issuer} \to \mathsf{TPM} : \mathsf{clsign}((U, r), sk_I)$

Fig. 12. DAA Join

We give in Figure 12 an Alice & Bob description of the protocol between the TPM and the issuer. The message $zeta_I = \mathsf{h}((0, bsnI))$ relies on $bsnI$: using a fresh $bsnI$ allows to ensure that the session of DAA Join will be unlinkable from previous ones. The message $tsk = \mathsf{h}((\mathsf{h}((\mathrm{DAAseed}, \mathsf{h}(KI))), cnt, 0))$ combines the internal secret of the TPM (*i.e.* DAAseed) with the public long-term key of the issuer (*i.e.* KI). The commit message $N_I = \mathsf{commit}(zeta_I, tsk)$ binds $zeta_I$ with the internal secret while the commit message $U = \mathsf{clcommit}(\mathsf{pk}(sk_I), n_v, tsk)$ expresses a credential request for a signature key sk_I. The goal of the TPM will be to get the message U signed by the issuer. More precisely, the issuer will blindly sign the message U with the signature key sk_I after making sure that the TPM can decrypt challenges encrypted with its public key (step 2.) and that he can provide a fresh ZK proof showing he knows its internal secret binds in U and N_I (step 5.). Finally, if all checks are successful, the issuer will blindly sign the credential request U (step 6.). We note $\mathsf{clsign}((U, r), sk_I)$ the blind signature of a commitment U with signature key sk_I and some random r. The function penc denotes a randomized asymmetric encryption scheme. Note that $\mathsf{ZK}(\cdot, \cdot)$ has two arguments: the first one should contain private data and the second one should contain public data. One can always extract public data from ZK proofs and one can check if both public and private data match as expected.

This protocol falls in our class and lies in the shared case[7] (*i.e.* $fn(\mathcal{I}) \cap fn(\mathcal{R}) = \{sk_{\mathsf{TPM}}\}$). UKano automatically established frame opacity in less than 30 seconds using the syntaxic idealisation, and in less that 3 seconds when using the quasi-syntaxic heuristic. Note that the semantic one is not precise enough to allow one to conclude. Regarding well-authentication, we had to leverage the same splitting technique explained in Section 6.5 so that UKano could conclude in a reasonable amount of time (around 30 seconds).

6.6.2. DAA sign

Once a TPM has obtained such a credential, it may prove its membership using the DAA sign protocol. This protocol is played by a TPM and a verifier: the verifier starts by challenging the TPM with a fresh nonce (step 1.), the latter then sends a complex ZK proof bound to this nonce (step 2.). The latter ZK proof also proves that the TPM knows a credential from the expected issuer bound to a secret he knows (essentially a message $\mathsf{clsign}((U, r), sk_I)$ received in a previous session of DAA join). The verifier accepts only if the ZK proof can be successfully checked (step 3.).

We give in Figure 13 an Alice & Bob description of the protocol between a verifier and the TPM willing to sign a message m using its credential $cred = \mathsf{clsign}((U, r), sk_I)$ he received from a past

[7]Both roles share the identity name sk_{TPM} (but note that the Issuer only uses $\mathsf{pk}(sk_{\mathsf{TPM}})$). Indeed, before executing the join protocol, the TPM and the issuer should establish a one-way authenticated channel that is not specified by the DAA scheme. Therefore, an Issuer session is associated to a single TPM's identity it is expected to communicate with.

DAA join session. From its credential $cred$, the TPM will compute a new credential dedicated to the current sign session: $cred' = \mathsf{clcommit}((\mathsf{pk}(sk_I), cred), n_c)$. Indeed, if the TPM had directly used $cred$ then two sessions of DAA sign would have been trivially linkable. The TPM also computes a commit of tsk that was used to obtain the credential: $N_V = \mathsf{commit}(tsk, zeta_V)$ where $zeta_V$ is a fresh nonce[8].

$$
\begin{array}{ll}
1.\ \mathsf{Verifier} \to \mathsf{TPM}: & n_v \\
2.\ \ \mathsf{TPM} \to \mathsf{Verifier}: & (zeta_V, \mathsf{pk}(sk_I), N_V, cred', n_t, \\
 & \quad \mathsf{ZK}_{\mathsf{sign}}((tsk, n_c), (zeta_V, \mathsf{pk}(sk_I), N_V, cred', (n_t, n_v, m)))) \\
3.\ \mathsf{Verifier} \to \mathsf{TPM}: & \mathsf{accept/reject}
\end{array}
$$

Fig. 13. DAA Sign

Similarly to Examples 10,11, we distinguish two cases whether sk_I is considered as a private constant or as an identity parameter. We recall that this choice critically impacts the privacy property that is modeled. Indeed, the privacy set [48] is considered to be (a) the set of users who obtained a credential from a given issuer in the former case, or, (b) the set of all users in the latter case.

(a) sk_I as a private constant. This 2-party protocol falls in our class and lies in the non-shared case. Indeed, we model infinitely many different TPMs that may take part to the DAA sign protocol with any verifier whose role is always the same (he has no proper identity). We automatically analysed this protocol with UKano and established both frame opacity and well-authentication in less than 4 seconds. Frame opacity has been established using a well-chosen idealisation adapted from the syntaxic heuristic.

(b) sk_I as an identity parameter. This 2-party protocol falls in our class and lies in the shared case. Indeed, we model infinitely many different TPMs with credentials signed by pairwise different issuers that may take part to the DAA sign protocol with a verifier who is checking credential from the corresponding issuer[9]. We automatically analysed this protocol with UKano and we found that frame opacity is violated for any of UKano heuristic (note that the idealisation for the case (a) is not conform for (b)). By inspecting the attack trace returned by UKano, one can quickly rebuild an attack against unlinkability and anonymity. Indeed, the attack on frame opacity shows that an attacker can exploit the fact that the ZK proof contains in its public part the public key of the issuer (*i.e.* $\mathsf{pk}(sk_I)$)). A passive eavesdropper is thus able to learn the issuer that has signed the credential used in a ZK proof sent by a prover, hence breaking anonymity and unlinkability. This is not surprising as the privacy mechanism of DAA sign was intended to protect users' privacy inside a certain group (associated with an issuer), which is a property we have checked, and which holds, with the variant (a).

6.6.3. Summary

We now summarise our results in Table 1. We only summarize results obtained regarding unlinkability and considering concurrent sessions. For each protocol, we mention the identity parameters of each role. Most of our case studies fall into the shared case with $fn(\mathcal{I}) \cap fn(\mathcal{R}) \neq \emptyset$.

[8]The protocol also specifies a mode that makes different signatures linkable by construction using $zeta_V = \mathsf{h}((0, bsnV))$. We focus on the other mode for which unlinkability is expected to hold.

[9]We discuss a more precise modelling and why it cannot be analyzed in our framework in 7.1.2.

We indicate the verification time in seconds to verify both conditions. When there is an attack, we give the time ProVerif takes to show that one of the condition fails to hold. We note ✓ for a condition automatically checked using our tool UKano and ✕ when the condition does not hold. Note that all positive results were established automatically using our tool UKano (which is based on ProVerif) without any manual effort (except for the cases indicated by ✓* for which little manual efforts were needed).

Protocol	Identity parameters		Frame opacity	Well-auth.	Unlink.	Verification time
	in role $\mathcal{I}$	in role $\mathcal{R}$				
Hash-Lock	k	k	✓	✓	**safe**	$<$ 1s
Fixed LAK	k	k	✓	✓	**safe**	$<$ 1s
BAC	k_E, k_M	k_E, k_M	✓	✓	**safe**	$<$ 1s
BAC/PA/AA	k_E, k_M	k_E, k_M	✓	✓	**safe**	$<$ 1s
BAC/AA/PA	k_E, k_M	k_E, k_M	✓	✓	**safe**	$<$ 1s
PACE (faillible dec)	k	k	–	✕	attack	$<$ 30s
PACE (as in [18])	k	k	–	✕	attack	$<$ 1m
PACE	k	k	–	✕	attack	$<$ 2m
PACE with tags	k	k	✓	✓	**safe**	$<$ 2m
ABCDH (irma)	$user_V$	$user_C$	✓	✓*	**safe**	$<$ 3h
DAA join	DAAseed, sk_{TPM}	sk_{TPM}	✓	✓*	**safe**	$<$ 5s
DAA sign (a)	$\varnothing$	DAAseed, cnt, r	✓*	✓	**safe**	$<$ 5s
DAA sign (b)	sk_I	sk_I, DAAseed, cnt, r	✕	✓	attack	$<$ 1s

Table 1
Summary of our case studies regarding unlinkability with concurrent sessions

7. Limitations of our approach

In this section, we would like to discuss some further limitations of our approach. We first explain some limitations that come from the approach itself in Section 7.1. In Section 7.2, we then discuss some limitations of our tool UKano which inherits some of the limitations of the ProVerif tool on which it is based.

7.1. Limitations of our Theorem 1

Our approach consists of providing two sufficient conditions under which anonymity (see Definition 12) and unlinkability (see Definition 10) are satisfied. These condtions, even if they are satisfied by many concrete examples, may not be fullfilled by some protocols that are nevertheless anonymous and unlinkable (see examples described in Section 7.1.1). We then discuss in Section 7.1.2 some limitations that come from the class of protocols we consider.

7.1.1. Tightness of our conditions

As illustrated by the toy protocols given in Examples 30 and 31, our conditions are sufficient but not necessary to ensure unlinkability or anonymity.

Example 30. *We suppose that, initially, each tag has its own key k and the reader maintains a database containing those keys. The protocol relies on symmetric encryption, and can be informally described as follows.*

$$\mathsf{Tag} \to \mathsf{Reader} : \{id\}_k$$

Once the reader receives the encryption, it opens it and checks the identity of the tag before accepting (or not) to grant access to the tag. This protocol falls into our generic class of 2-party protocols (shared case). Anonymity w.r.t. id is satisfied but unlinkability is not: a given tag always sends the same message.

Regarding our conditions, frame opacity does not *hold. Consider*

$$\phi = \{w_1 \mapsto \{id_1\}_{k_1},\ w_2 \mapsto \{id_1\}_{k_1},\ w_3 \mapsto \{id_2\}_{k_2}\}.$$

Such a frame can be obtained when executing the $\mathcal{M}_\Pi$ process. The syntactical idealisation will rename both occurrences of id_1 (resp. k_1) using different names whereas the semantical idealisation will idealise each output using a fresh names. In both cases, the resulting idealised frame is not statically equivalent to ϕ. Thus, frame opacity cannot be established using these idealisations. Actually, no idealisation will be able to idealise the two first outputs in the same way and the two last outputs in different way at the same time. This illustrates that frame opacity is a too strong condition when considering anonymity.

Regarding well-authentication, we can establish that such a condition does not *hold as well. Still considering the execution leading to the frame ϕ above, we can then consider a reader that starts two sessions accepting twice w_1 as an input. It will then continue by executing its conditionals positively. The annotations of these two conditionals will be respectively $R(\{k_1, id_1\}, sid)$ and $R(\{k_1, id_1\}, sid')$. Therefore, condition (ii) of Definition 19 is not satisfied. These two conditionals are not safe and they are both associated to the same annotation (the one carried out by the output w_1). This does not break anonymity but simply shows that replaying messages is a scenario that allows an attacker to fool one party (here the reader) up to some point (here until the end).*

Example 31. *In order to ensure unlinkability, we now suppose that the tag sends its identity accompanied with a freshly generated random number r. Therefore, we have that:*

$$\mathsf{Tag} \to \mathsf{Reader} : \{\langle r, id\rangle\}_k.$$

This protocol falls into our generic class of 2-party protocols (shared case). The identity parameters of both roles are id and k whereas r is the session parameter of role $\mathcal{I}$. As in the previous example, anonymity w.r.t. id holds. Unlinkability should hold, assuming that the reader does not output any message indicating whether the test has been passed with success or not.

Actually, frame opacity can be established relying on either the syntaxical idealisation or the semantical one. The fresh random number inside each encryption allows one to ensure that all

the ciphertexts are different. However, for the same reason as the one explained in the previous example, well-authentication does not hold: condition (ii) is not satisfied.

We recall that this can be considered as a false attack only if the protocol and the use case both enforce that the continuation of the protocol in case the test passes is always indistinguishable from the continuation in the other case; this is a strong assumption.

7.1.2. Class of protocols

Among the limitations coming from our definition of protocols, we first reconsider the DAA sign and PACE protocols to highlight some limitations of our approach.

Two parties only. Our notion of protocols only covers 2-party protocols. This obviously excludes important protocols with more than 2 parties such as secure group communication protocols [45], e-voting protocols [37], make mobile communication protocols [14], the combination of DAA join and DAA sign [37] that features 3 parties, etc. This also excludes scenarios where privacy is considered between *group of entities*. For instance for DAA sign (see Example 10 or Section 6.6.2), verifiers and clients may be associated to different issuers. Using our framework, one can analyze privacy between users in a single group (as in Example 10), or between groups but where each goup has only one user (as in Example 11). In the latter case, one would rather want to model privacy between groups where each group contains an unbounded number of users, but this is out of the scope of our approach. This is not surprinsing since such a scenario actually features three parties: clients, verifiers, issuers (forming groups); all with unbounded number of entities.

Honest trace. Now, we want to report on a potential limitation we discovered when analysing the PACE protocol using Tamarin. Our initial aim was to investigate the scenario where sessions can be executed only sequentially. We have turned to Tamarin since ProVerif is not able to faithfully model such scenarios. We wrote a Tamarin model encoding well-authentication and found surprisingly that this condition does not hold, even with the tagged version. This contrasts with the positive result obtained with ProVerif. Actually, this comes from the fact that Tamarin models Diffie-Hellman exponentiation in a more faithful way than ProVerif. Some behaviours that were not possible in the ProVerif model become possible, and it happens that well-authentication is not satisfied in such a model.

Indeed, the attacker can alter the Diffie-Hellman shares, as informally depicted in Figure 14, without impacting the successive conditionals. This is problematic because successful tests will

$$
\begin{array}{rl}
1. & \mathsf{Tag} \to \mathsf{Reader} : \ \{s_T\}_k \\
2. & \mathsf{Reader} \to \mathsf{Attacker} : g^{n_R} \\
2'. & \mathsf{Attacker} \to \mathsf{Tag} : \ (g^{n_R})^X \\
3. & \mathsf{Tag} \to \mathsf{Attacker} : g^{n_T} \\
3'. & \mathsf{Attacker} \to \mathsf{Reader} : \ (g^{n_T})^X
\end{array}
$$

Fig. 14. Example of successful but dishonest interaction (X can be any message)

pass (independently of the message X) while such interactions are not honest according to our current definition of honest trace (see Definition 4). This problematic interaction is however not detected in ProVerif, due to the lack of equations in the underlying equational theory: the final keys computed by both parties will be different, $((g^{n_R})^X)^{n_T}$ for the tag and $((g^{n_T})^X)^{n_R}$ for the

reader. Therefore such an interaction cannot be completed sucessfully, and well-authentication will be established.

The failure of well-authentication described above does not yield a failure of unlinkability. It is thus a case where one might want to make well-authentication less restrictive. One direction for weakening it is to extend the notion of "honest trace associated to a protocol" (Definition 5): instead of a single honest trace we would associate to a protocol a set of symbolic traces that are, roughly, traces with (possibly) variables in recipes. For PACE, one may for instance use $\mathsf{tr}_h = \mathsf{out}(c_I, w_1).\mathsf{in}(c_R, \mathsf{dh}(w_1, X)).\mathsf{out}(c_R, w_2).\mathsf{in}(c_I, \mathsf{dh}(w_2, X)).\ldots$ in addition to the standard trace. However, in order to adapt our proof technique, we need to make sure that whatever the recipes chosen to fill in the variables (*e.g.* X in tr_h), the resulting concrete trace can be executed by the protocol and the produced frame always has the same idealisation. Remark that this is the case for tr_h in the case of the PACE protocol.

In practice, this limitation on the well-authentication condition does not seem very important, as the issue is tied to the peculiar use of two Diffie-Hellman rounds in PACE and, expanding the discussion beyond privacy, the attack on well-authentication shows a potential weakness in that protocol. Indeed, Tag and Reader fail to establish an agreement on each other's Diffie-Hellman shares from the first round (*i.e.* g^{n_R} and g^{n_T}). Therefore, an attacker is able to manipulate those shares without being detected, which goes against best practices in protocol design. In contrast, the MAC messages 8 and 9 (see Figure 9) allow the Tag and Reader to agree on each other's Diffie-Hellman shares from the second round (*i.e.* $G^{n'_T}$ and $G^{n'_R}$) and on the shared key resulting from the first round (*i.e.* $g^{n_R n_T}$ in G). Adding g^{n_T} (respectively g^{n_R}) to the first (respectively second) MAC message would fix this lack of agreement. We have formally verified with Tamarin that PACE with this modification indeed satisfies well-authentication, thus providing an agreement on the full protocol transcript.

Stateless only. Our framework and our theorem only applies to stateless protocols (*i.e.* no mutable states persistent across sessions). This immediately excludes numerous real-world protocols such as secure messaging protocols [31], mobile communication protocols [14], etc.

7.2. Limitations of our tool UKano

Our tool UKano also suffers from some limitations. In particular, the heuristics we propose to build idealisation could be improved. For instance, in case of DAA sign, we were unable to establish frame opacity fully automatically: we had to propose a well-chosen idealisation adapted from the syntaxic heuristic. Note that, in general the syntaxic heuristic is a good choice but yields large messages that may cause some efficiency issues to ProVerif. The semantical heuristic is less precise but much more efficient. In case of DAA sign, we make a trade-off betweeen these two choices to establish frame opacity. Regarding well-authentication, the resulting queries happen to be quite big and may also cause some troubles to ProVerif. This can be adressed by soundly splitting the query (see Section 6.5) but this feature has not been implemented in UKano.

Besides the limitations of our tool UKano, we also inherit some of the limitations of the backend tool, i.e. ProVerif. For instance, in terms of cryptographic primitives, we have seen that ProVerif only consider a very abstract model for modular exponentiation, and does not allow one to model all the algebraic properties of the exclusive-or operator. We are also unable to faithfully model scenarios involving sequential compositions. This feature could be modeled in ProVerif relying on

private chanels but abstractions performed by ProVerif when modeling private channels will not allow us to benefit from the extra information of that encoding. Of course, this limits the scope of our approach but progress made on existing verification tools will directly benefit to our approach as well. In particular, a natural extension for our tool UKano would be to consider Tamarin as a backend. This would bring precise support for sequential composition, a more faithful model for Diffie-Hellman exchange, and also the recent extension to deal with the exclusive-or operator [39].

8. Conclusion

We have identified two conditions, namely well-authentication and frame opacity, which imply anonymity and unlinkability for a wide class of protocols. Additionally, we have shown that these two conditions can be checked automatically using the tool ProVerif, and we have mechanised their verification in our tool UKano. This yields a new verification technique to check anonymity and unlinkability for an unbounded number of sessions. It has proved quite effective on various case studies. In particular, it has brought first-time unlinkability proofs for the BAC protocol (e-passport) and ABCDH protocol. Our case studies also illustrated that our methodology is useful to discover attacks against unlinkability and anonymity as illustrated by the new attacks we found on PACE and LAK.

In the future, we plan to improve the way our sufficient conditions are checked. For instance, we would like to let UKano build idealisations in a cleverer way; notably in the choice of heuristics to adopt, and we are interested in other tools we may leverage as back-ends (*e.g.* Tamarin). We are also interested in simplifying further the verification of our conditions towards completely reducing the verification of equivalence-based properties to pure reachability verifications, which are known to be much simpler. Specifically, we believe that frame opacity could be verified via reachability and syntactical checks only. Obtaining such a result would certainly be useful, as it would allow us to use a richer toolset to verify case studies.

Based on limitations discussed in Section 7.1.2, we also identify a number of research problems aimed at generalizing the impact of our technique. We would like to investigate the extension of our main theorem to the case of protocols with states. This is certainly technically challenging, but would make it possible to model more protocols, or at least model them more faithfully. We are also interested in extending our method to protocols with more than 2 parties which are commonplace (*e.g.* the combination of DAA join and DAA sign is essentially a 3-party protocol).

Finally, we believe that the overall methodology developed in this paper (*i.e.* privacy via sufficient conditions) could be applied in other contexts where privacy is critical: *e.g.* e-voting, attribute-based credentials, blockchain technologies, transparent certificate authorities. In our opinion, the privacy via sufficient conditions approach also sheds light on the privacy notions themselves. Indeed, each sufficient condition helps to get a better grasp of necessary ingredients for preserving privacy. It might thus be interesting to translate such conditions into more comprehensive guidelines helping the design of new privacy-enhancing protocols.

Acknowledgement. We would like to thank Bruno Blanchet for his valuable help regarding the mechanisation in ProVerif of our frame opacity condition. The extension of bi-processes in Section 5.1 is due to him. We also thank Solène Moreau for her useful feedback on earlier versions of this paper.

References

[1] PKI for machine readable travel documents offering ICC read-only access. Technical report, International Civil Aviation Organization, 2004.

[2] Iso 15408-2: Common criteria for information technology security evaluation - part 2: Security functional components, July 2009.

[3] M. Abadi and B. Blanchet. Computer-assisted verification of a protocol for certified email. In *Static Analysis*, pages 316–335. Springer, 2003.

[4] M. Abadi and C. Fournet. Mobile values, new names, and secure communication. In *Proceedings of POPL'01*. ACM Press, 2001.

[5] G. Alpár and J.-H. Hoepman. A secure channel for attribute-based credentials:[short paper]. In *Proceedings of the 2013 ACM workshop on Digital identity management*, pages 13–18. ACM, 2013.

[6] M. Arapinis, V. Cheval, and S. Delaune. Verifying privacy-type properties in a modular way. In *Proceedings of the 25th IEEE Computer Security Foundations Symposium (CSF'12)*, pages 95–109, Cambridge Massachusetts, USA, June 2012. IEEE Computer Society Press.

[7] M. Arapinis, T. Chothia, E. Ritter, and M. Ryan. Analysing unlinkability and anonymity using the applied pi calculus. In *Proceedings of CSF'10*. IEEE Comp. Soc. Press, 2010.

[8] M. Arapinis, L. Mancini, E. Ritter, M. Ryan, N. Golde, K. Redon, and R. Borgaonkar. New privacy issues in mobile telephony: fix and verification. In *Proceedings of the ACM conference on Computer and communications security*, pages 205–216. ACM, 2012.

[9] M. Arapinis, L. I. Mancini, E. Ritter, and M. Ryan. Privacy through pseudonymity in mobile telephony systems. In *NDSS*, 2014.

[10] A. Armando, R. Carbone, L. Compagna, J. Cuéllar, and M. L. Tobarra. Formal analysis of SAML 2.0 web browser single sign-on: breaking the SAML-based single sign-on for Google apps. In *Proc. 6th ACM Workshop on Formal Methods in Security Engineering (FMSE'08)*, pages 1–10. ACM, 2008.

[11] A. Armando et al. The AVANTSSAR platform for the automated validation of trust and security of service-oriented architectures. In *Proc. 18th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'12)*, volume 7214, pages 267–282. Springer, 2012.

[12] M. Backes, C. Hritcu, and M. Maffei. Automated verification of remote electronic voting protocols in the applied pi-calculus. In *Proceedings of the 21st IEEE Computer Security Foundations Symposium, CSF 2008, Pittsburgh, Pennsylvania, 23-25 June 2008*, pages 195–209. IEEE Computer Society, 2008.

[13] M. Backes, M. Maffei, and D. Unruh. Zero-knowledge in the applied pi-calculus and automated verification of the direct anonymous attestation protocol. In *Security and Privacy, 2008. SP 2008. IEEE Symposium on*, pages 202–215. IEEE, 2008.

[14] D. Basin, J. Dreier, L. Hirschi, S. Radomirović, R. Sasse, and V. Stettler. Formal analysis of 5G authentication. *arXiv preprint arXiv:1806.10360*, 2018.

[15] D. Basin, J. Dreier, and R. Sasse. Automated symbolic proofs of observational equivalence. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 1144–1155. ACM, 2015.

[16] M. Baudet. Deciding security of protocols against off-line guessing attacks. In *Proc. 12th Conference on Computer and Communications Security*. ACM, 2005.

[17] J. Bender, Ö. Dagdelen, M. Fischlin, and D. Kügler. The pace aa protocol for machine readable travel documents, and its security. In *Financial Cryptography and Data Security*, pages 344–358. Springer, 2012.

[18] J. Bender, M. Fischlin, and D. Kügler. Security analysis of the pace key-agreement protocol. In *Information Security*, pages 33–48. Springer, 2009.

[19] B. Blanchet. An Efficient Cryptographic Protocol Verifier Based on Prolog Rules. In *Proceedings of CSFW'01*, pages 82–96. IEEE Comp. Soc. Press, 2001.

[20] B. Blanchet, M. Abadi, and C. Fournet. Automated verification of selected equivalences for security protocols. *Journal of Logic and Algebraic Programming*, 2008.

[21] B. Blanchet and B. Smyth. Automated reasoning for equivalences in the applied pi calculus with barriers. In *Proc. 29th Computer Security Foundations Symposium*, 2016.

[22] M. Brusó. *Dissecting Unlinkability*. PhD thesis, Technische Universiteit Eindhoven, 2014.

[23] M. Brusó, K. Chatzikokolakis, and J. den Hartog. Formal verification of privacy for RFID systems. In *Proceedings of CSF'10*, 2010.

[24] M. Brusó, K. Chatzikokolakis, S. Etalle, and J. Den Hartog. Linking unlinkability. In *Trustworthy Global Computing*, pages 129–144. Springer, 2012.

[25] J. Camenisch, A. Lehmann, and G. Neven. Electronic identities need private credentials. *IEEE Security & Privacy*, 10(1):80–83, 2012.

[26] J. Camenisch, S. Mödersheim, and D. Sommer. A formal model of identity mixer. In *Formal Methods for Industrial Critical Systems*, pages 198–214. Springer, 2010.
[27] L. Cheikhrouhou, W. Stephan, Ö. Dagdelen, M. Fischlin, and M. Ullmann. Merging the cryptographic security analysis and the algebraic-logic security proof of pace. In *Sicherheit*, pages 83–94, 2012.
[28] V. Cheval and B. Blanchet. Proving more observational equivalences with ProVerif. In *Proc. 2nd Conference on Principles of Security and Trust*, volume 7796 of *LNCS*, pages 226–246. Springer, 2013.
[29] V. Cheval, H. Comon-Lundh, and S. Delaune. Trace equivalence decision: Negative tests and non-determinism. In *Proceedings of CCS'11*. ACM Press, 2011.
[30] R. Chrétien, V. Cortier, and S. Delaune. From security protocols to pushdown automata. *ACM Transactions on Computational Logic*, 17(1:3), Sept. 2015.
[31] K. Cohn-Gordon, C. Cremers, B. Dowling, L. Garratt, and D. Stebila. A formal security analysis of the signal messaging protocol. In *Security and Privacy (EuroS&P), 2017 IEEE European Symposium on*, pages 451–466. IEEE, 2017.
[32] H. Comon and A. Koutsos. Formal computational unlinkability proofs of rfid protocols. In *2017 IEEE 30th Computer Security Foundations Symposium (CSF)*, pages 100–114, Aug 2017.
[33] V. Cortier, S. Delaune, and P. Lafourcade. A survey of algebraic properties used in cryptographic protocols. *Journal of Computer Security*, 14(1):1–43, 2006.
[34] V. Cortier and B. Smyth. Attacking and fixing Helios: An analysis of ballot secrecy. *Journal of Computer Security*, 21(1):89–148, 2013.
[35] S. Delaune and L. Hirschi. A survey of symbolic methods for establishing equivalence-based properties in cryptographic protocols. *Journal of Logical and Algebraic Methods in Programming*, 2016.
[36] S. Delaune, S. Kremer, and M. D. Ryan. Verifying privacy-type properties of electronic voting protocols. *Journal of Computer Security*, (4), 2008.
[37] S. Delaune, M. D. Ryan, and B. Smyth. Automatic verification of privacy properties in the applied pi-calculus. In *Proceedings of the 2nd Joint iTrust and PST Conferences on Privacy, Trust Management and Security (IFIPTM'08)*, volume 263 of *IFIP Conference Proceedings*. Springer, 2008.
[38] N. Dong, H. Jonker, and J. Pang. Formal analysis of privacy in an ehealth protocol. In *Computer Security–ESORICS 2012*, pages 325–342. Springer, 2012.
[39] J. Dreier, L. Hirschi, S. Radomirovic, and R. Sasse. Automated unbounded verification of stateful cryptographic protocols with exclusive or. In *31st IEEE Computer Security Foundations Symposium (CSF'2018)*, 2018.
[40] M. Feldhofer, S. Dominikus, and J. Wolkerstorfer. Strong authentication for RFID systems using the AES algorithm. In *Cryptographic Hardware and Embedded Systems-CHES 2004*, pages 357–370. Springer, 2004.
[41] L. Hirschi. *Automated Verification of Privacy in Security Protocols: Back and Forth Between Theory & Practice*. PhD thesis, École Normale Supérieure Paris-Saclay, April 2017.
[42] L. Hirschi, D. Baelde, and S. Delaune. A method for verifying privacy-type properties: the unbounded case. In *Proceedings of the 37th IEEE Symposium on Security and Privacy (S&P'16)*, San Jose, California, USA, May 2016. IEEE Computer Society Press.
[43] A. Juels and S. A. Weis. Defining strong privacy for RFID. *ACM Transactions on Information and System Security (TISSEC)*, 13(1):7, 2009.
[44] S. Lee, T. Asano, and K. Kim. RFID mutual authentication scheme based on synchronized secret information. In *Symposium on cryptography and information security*, 2006.
[45] C. Meadows. Formal methods for cryptographic protocol analysis: Emerging issues and trends. *IEEE journal on selected areas in communications*, 21(1):44–54, 2003.
[46] S. Meier, B. Schmidt, C. Cremers, and D. Basin. The Tamarin Prover for the Symbolic Analysis of Security Protocols. In *Proc. 25th International Conference on Computer Aided Verification (CAV'13)*, volume 8044 of *LNCS*, pages 696–701. Springer, 2013.
[47] C. Paquin and G. Zaverucha. U-prove cryptographic specification v1.1 (revision 3), December 2013.
[48] A. Pfitzmann and M. Köhntopp. Anonymity, unobservability, and pseudonymity—a proposal for terminology. In *Designing privacy enhancing technologies*, pages 1–9. Springer, 2001.
[49] S. Santiago, S. Escobar, C. Meadows, and J. Meseguer. A formal definition of protocol indistinguishability and its verification using maude-npa. In *Security and Trust Management*, pages 162–177. Springer, 2014.
[50] B. Smyth, M. D. Ryan, and L. Chen. Formal analysis of privacy in direct anonymous attestation schemes. *Science of Computer Programming*, 111:300–317, 2015.
[51] UKano tool and case studies. `http://projects.lsv.ens-cachan.fr/ukano/`. Accessed: 12-11-2018.
[52] T. Van Deursen and S. Radomirovic. Attacks on RFID protocols. *IACR Cryptology ePrint Archive*, 2008:310, 2008.

Appendix

We provide in this appendix the proof of our main result (Theorem 1). Our main argument consists in showing that, for any execution of $\mathcal{M}_{\Pi,\overline{id}}$, there is an indistinguishable execution of $\mathcal{S}_\Pi$ (the other direction being easy). This indistinguishable execution will be obtained by a modification of the involved agents. We will proceed via a renaming of agents applied to an abstraction of the given execution of $\mathcal{M}_{\Pi,\overline{id}}$. We then prove that the renamed executions can still be executed and produce an indistinguishable frame.

We fix a protocol $\Pi = (\overline{k}, \overline{n}_I, \overline{n}_R, \dagger_I, \dagger_R, \mathcal{I}, \mathcal{R})$, some identity names $\overline{id}$ and some fresh constants $\overline{\mathsf{id}_0}$ yielding a process $\mathcal{M}_{\Pi,\overline{id}}$ as defined in Section 3. We denote id_{0i} (resp id_i, k_i, and k'_i) the i^{th} element of the sequence $\overline{\mathsf{id}_0}$ (resp. $\overline{id}$, $\overline{k}$, and $\overline{k'}$). The construction of the proof will slightly differ depending on $\dagger_I, \dagger_R$ (sequential vs concurrent sessions) and whether $fn(\mathcal{I}) \cap fn(\mathcal{R}) = \varnothing$ or not (non-shared case vs shared case).

A. Abstraction of configurations

Instead of working with $\mathcal{M}_{\Pi,\overline{id}}$, $\mathcal{M}_\Pi$, and $\mathcal{S}_\Pi$, it will be more convenient to work with *ground configurations*. Intuitively, we will associate to each execution of $\mathcal{M}_{\Pi,\overline{id}}$, $\mathcal{M}_\Pi$, and $\mathcal{S}_\Pi$, a ground configuration that contains all agents involved in that execution, already correctly instantiated. By doing so, we are able to get rid of technical details such as unfolding replications and repetitions a necessary number of times, create necessary identity and session parameters, etc. These ground configurations are generated from sequences of annotations satisfying some requirements.

A.1. Sequences of annotations

The sequence of annotations from which we will build ground configurations shall satisfy some requirements that we list below. Essentially, the goal is to make sure that no freshness condition over session and identity names is violated.

Definition 21. *A sequence S of annotations is* well-formed *if the following conditions hold.*

- *In all annotations $A(\overline{k'}, \overline{n'})$ with $A \in \{I, R\}$, the session parameters $\overline{n'}$ are names, and the identity parameters $\overline{k'}$ are made of names or constants $\overline{\mathsf{id}_0}$. We also have that $|\overline{n'}| = |\overline{n_A}|$ and $|\overline{k'}| = |\overline{k}|$. Moreover, when $\overline{\mathsf{id}_0} \cap \overline{k'} \neq \varnothing$, $k'_i = \mathsf{id}_{0j}$ if, and only if, $k_i = id_j$.*
- *No name appears both as identity and session parameter in any two annotations.*
- *Two different annotations never share a session parameter.*
- *Two annotations either have the same identity parameters, or do not share any identity parameter at all.*

We say that a sequence of annotations S is single-session *when two different annotations of the same role never share an identity parameter and no annotation contains a constant in $\overline{\mathsf{id}_0}$.*

We straightforwardly lift those definitions to annotated traces by only keeping the underlying sequence of annotations and dropping actions. A well-formed (resp. well-formed, single-session) sequence of annotations contains annotations of agents that can be instantiated from $\mathcal{M}_{\Pi,\overline{id}}$ (resp. $\mathcal{S}_\Pi$). Conversely, we may note that given an annotated trace ta such that $\mathcal{M}_{\Pi,\overline{id}} \xrightarrow{\mathsf{ta}} K'$, we have that ta is well-formed.

A.2. Ground configurations

After the introduction of some notations, we explain how *ground configurations* are obtained from well-formed sequences of annotations. Given a well-formed sequence of annotations S, and some role $A \in \{I, R\}$, we introduce the following notations:

- $\mathrm{id}_A(S)$ is the set of identity parameters of agents of role A occurring in S, *i.e.*

$$\mathrm{id}_A(S) = \{\overline{k} \mid A(\overline{k}, \overline{n}) \in S\}.$$

- for $\bar{l} \in \mathrm{id}_A(S)$, we note $\mathrm{sess}_A(S, \bar{l})$ the set of session parameters of agents of role A and identity parameters $\bar{l}$ occurring in S, *i.e.* $\mathrm{sess}_A(S, \bar{l}) = \{\overline{n} \mid A(\bar{l}, \overline{n}) \in S\}$.
- for $\bar{l} \in \mathrm{id}_A(S)$, we note $\mathrm{sess}_A^{\mathsf{seq}}(S, \bar{l})$ the sequence made of elements from $\mathrm{sess}_A(S, \bar{l})$, without repetition, in order of first occurrence in S.

Finally, for some sequence of elements $L = e_1, e_2, e_3, \ldots$ and a process $P(e)$ parametrized by e, we denote by $\coprod_{e \in L} P(e)$ the process $P(e_1); (P(e_2); (P(e_3); \ldots))$.

Definition 22. *Let S be a well-formed sequence of annotations. The* ground configuration associated to S, *denoted by $\mathcal{K}(S)$, is the multiset $\mathcal{P}_I \sqcup \mathcal{P}_R$ where $\mathcal{P}_I$ is defined as follows depending on $\dagger_I$:*

- *if $\dagger_I = !$ then $\mathcal{P}_I = \left\{ \left(\mathcal{I}\{\overline{k} \mapsto \bar{l}, \overline{n}_I \mapsto \overline{m}\}\right)[I(\bar{l}, \overline{m})] \mid I(\bar{l}, \overline{m}) \in S \right\}$;*
- *if $\dagger_I = \mathsf{i}$ then $\mathcal{P}_I = \left\{ \coprod_{\overline{m} \in S_{\bar{l}}} \left(\mathcal{I}\{\overline{k} \mapsto \bar{l}, \overline{n}_I \mapsto \overline{m}\}\right)[I(\bar{l}, \overline{m})] \mid \bar{l} \in \mathrm{id}_I(S) \text{ and } S_{\bar{l}} = \mathrm{sess}_I^{\mathsf{seq}}(S, \bar{l}) \right\}$.*

The multiset $\mathcal{P}_R$ is computed in a similar way, replacing $\mathcal{I}$, $\overline{n}_I$ and I by $\mathcal{R}$, $\overline{n}_R$ and R respectively.

Example 32. *Consider the following toy protocol $\Pi_{\mathsf{toy}} := (k, n_I, n_R, \mathsf{i}, \mathsf{i}, \mathcal{I}, \mathcal{R})$ where $\mathcal{I} = \mathsf{out}(c_I, \mathsf{enc}(n_I, k))$ and $\mathcal{R} = \mathsf{in}(c_R, x)$. We have that*

$$\mathcal{M}_{\Pi_{\mathsf{toy}}} =! \,\mathsf{new}\, k.(\mathsf{i}\,\mathsf{new}\, n_I.\mathcal{I} \mid \mathsf{i}\,\mathsf{new}\, n_R.\mathcal{R}) \xrightarrow{\mathsf{ta}} (\mathcal{Q}; \phi)$$

for $\mathsf{ta} = \tau.\tau.\tau.\tau.\tau.\ \ell : \mathsf{out}(c_I, w_0)[I(k, n_I)].\tau.\tau.\tau.\ \ell : \mathsf{out}(c_I, w_1)[I(k, n'_I)]$. The ground configuration associated to ta is the following multiset with one element:

$$\mathcal{K}(\mathsf{ta}) = \{(\mathsf{out}(c_I, \mathsf{enc}(n_I, k))[I(k, n_I)]); (\mathsf{out}(c_I, \mathsf{enc}(n'_I, k))[I(k, n'_I)])\}.$$

Note that $\mathcal{K}(\mathsf{ta})$ is also able to produce the annotated trace ta up to some τ actions.

We lift those definitions to annotated traces as before. A ground configuration associated to a well-formed sequence of annotations is essentially an "unfolding" of $\mathcal{M}_{\Pi, \overline{id}}$. Therefore, there is a strong relationship between the original process and the one obtained through $\mathcal{K}(\cdot)$ as established in the following proposition.

Proposition 4. *Let ta be a well-formed annotated trace. We have that:*

(1) *If $\mathcal{M}_{\Pi, \overline{id}} \overset{\mathsf{ta}}{\Longrightarrow} K$ (resp. $\mathcal{S}_\Pi \overset{\mathsf{ta}}{\Longrightarrow} K$), then $\mathcal{K}(\mathsf{ta}) \overset{\mathsf{ta}}{\Longrightarrow} K'$ for some K' such that $\phi(K) = \phi(K')$.*

(2) *If $\mathcal{K}(\mathsf{ta}) \overset{\mathsf{ta}}{\Longrightarrow} K$, then $\mathcal{M}_{\Pi, \overline{id}} \overset{\mathsf{ta}}{\Longrightarrow} K'$ for some K' such that $\phi(K) = \phi(K')$.*

(3) *If* $\mathcal{K}(\mathsf{ta}) \overset{\mathsf{ta}}{\Longrightarrow} K$ *and* ta *is single-session, then* $\mathcal{S}_\Pi \overset{\mathsf{ta}}{\Longrightarrow} K'$ *for some* K' *such that* $\phi(K) = \phi(K')$.

(4) *If* $\mathsf{ta} = \mathsf{ta}_1.\mathsf{ta}_2$ *and* $\mathcal{K}(\mathsf{ta}_1.\mathsf{ta}_2) \xrightarrow{\mathsf{ta}_1} K$ *then* $\mathcal{K}(\mathsf{ta}_1) \xrightarrow{\mathsf{ta}_1} K'$ *for some* K' *such that* $\phi(K) = \phi(K')$.

Proof. Item (1) holds by construction of the operator $\mathcal{K}(\cdot)$, which has been built by closely mimicking how $\mathcal{M}_{\Pi,\overline{id}}$ and $\mathcal{S}_\Pi$ create agents. We thus have that when an agent is at top-level in a configuration in the execution of $\mathcal{M}_{\Pi,\overline{id}}$ (or $\mathcal{S}_\Pi$) then it is also available in the execution of $\mathcal{K}(\mathsf{ta})$. In general, less τ actions are necessary for the execution starting with $\mathcal{K}(\mathsf{ta})$ than for the executions starting with $\mathcal{M}_{\Pi,\overline{id}}$ or $\mathcal{S}_\Pi$. Indeed, there is no need, in ground configurations, to spawn agents by unfolding replications or repetitions or creating fresh names. This is because agents are (more) immediately available in $\mathcal{K}(\mathsf{ta})$.

Item (2) heavily relies on the well-formedness of ta. One can thus prove that all agents in $\mathcal{K}(\mathsf{ta})$ can be created along the execution by choosing appropriate names when triggering rules New. For instance, the first item of Definition 21 makes sure that the arity of parameters in agents matches the number of names to be created. The second and third items of Definition 21 ensure that the freshness guard conditions of the rule New holds for names to be created. Finally, the fourth item of Definition 21 implies that when an agent $a = A(\overline{k}, \overline{n})$ must be created then either (i) names in $\overline{k}$ are completely fresh and this identity $\overline{k}$ can be created from $\mathcal{M}_{\Pi,\overline{id}}$ by unfolding ! and create names $\overline{k}$ or (ii) names in $\overline{k}$ have already been created and thus the agent a can be created from the last replicated process used to create identity $\overline{k}$ in the first place.

Item (3) is similar to (2). The single-session hypothesis provides exactly what is needed to mimic the execution using $\mathcal{S}_\Pi$ rather than $\mathcal{M}_{\Pi,\overline{id}}$.

Finally, item (4) stems from a simple observation. Compared to $\mathcal{K}(\mathsf{ta}_1)$, the multiset of processes $\mathcal{K}(\mathsf{ta}_1.\mathsf{ta}_2)$ adds processes in parallel and in sequence after some processes of $\mathcal{K}(\mathsf{ta}_1)$. However, these extra processes are unused when executing ta_1, thus $\mathcal{K}(\mathsf{ta}_1)$ can perform the same execution. □

A.3. Renamings of annotations

As mentioned before, we shall prove that for any execution of $\mathcal{M}_{\Pi,\overline{id}}$, there is an indistinguishable execution of $\mathcal{S}_\Pi$. This indistinguishable execution that $\mathcal{S}_\Pi$ can perform will be obtained by a renaming of annotations. We define next a generic notion of such renamings of annotations. However, the crux of the final proof is to find a *good* renaming that implies: (i) the executability by $\mathcal{S}_\Pi$ of the renamed trace, and (ii) the static indistinguishability of the resulting frames (before and after the renaming).

Definition 23. *A* renaming of annotations *(denoted by ρ) is an injective mapping from annotations to annotations such that:*

- *for any well-formed sequence of annotations S, $S\rho$ is well-formed;*
- *ρ is role-preserving:* i.e. *initiator (resp. responder) annotations are mapped to initiator (resp. responder) annotations;*
- *for any two annotations $a_1 = A_1(\overline{k_1}, \overline{n_1})$, $a_2 = A_2(\overline{k_2}, \overline{n_2})$, if $\rho(a_1)$ and $\rho(a_2)$ have the same identity parameters, then $\overline{k_1} = \overline{k_2}$.*

The two first conditions are expected: renaming of annotations shall only modify session and identity parameters whilst preserving well-formedness. The final condition ensures that renamings do not create more "sequential dependencies" between agents (*i.e.* agents sharing the same identity and whose role can execute sessions only sequentially): after renaming, less pairs of agents have same identity.

Next, we define $\mathsf{ta}\rho$ as the annotated trace obtained from ta by applying ρ to annotations only. Note that, by definition of renamings, the resulting $\mathsf{ta}\rho$ is well-formed as well.

One can also define the effect of renamings on ground configurations. If $\rho(A(\overline{k},\overline{n})) = A(\overline{k'},\overline{n'})$, the renaming σ induced by ρ on $A(\overline{k},\overline{n})$ is the (injective) mapping such that $\overline{k}\sigma = \overline{k'}$ and $\overline{n}\sigma = \overline{n'}$. Given a ground configuration $\mathcal{P} = \{\coprod_j P_j^i[a_j^i]\}_i$, we define $\mathcal{P}\rho = \{\coprod_j P_j^i\sigma_j^i[\rho(a_j^i)]\}_i$ where σ_j^i is the renaming induced by ρ on a_j^i. Note that the renaming on parameters induced by a renaming of annotations may conflict: this happens, for example, when $\rho(A(\overline{k},\overline{n})) = A(\overline{k_1},\overline{n})$ and $\rho(A(\overline{k},\overline{m})) = A(\overline{k_2},\overline{m})$.

A renaming of annotations can break executability. Even when executability is preserved, it is not obvious to relate processes before and after the renaming, as messages can be affected in complex ways and conditionals may not evaluate to the same outcome. Fortunately, frame opacity and well-authentication will provide us with strong properties to reason over executions, as seen in the next subsections.

Example 33. *Consider the annotated trace* ta *from Example 18 that* $\mathcal{M}_{\Pi,\overline{id}}$ *can execute. The ground configuration* $\mathcal{K}(\mathsf{ta})$ *can execute it as well, using* $\Longrightarrow$*. We now define* ρ *as follows:* $\rho(I(k',n'_I)) = I(k_1,n'_I)$ *and* $\rho(R(k',n'_R)) = R(k_2,n'_R)$ *for some fresh names* k_1,k_2*. The trace* $\mathsf{ta}\rho$ *can no longer be executed by* $\mathcal{M}_{\Pi,\overline{id}}$ *nor by* $\mathcal{K}(\mathsf{ta})\rho$ *(even using* $\Longrightarrow$*) because the first output sent by* $R(k_2,n'_R)$ *(*i.e. $\mathsf{enc}(\langle n'_I,n'_R\rangle,k_2)$*) will not be accepted by* $I(k_1,n'_I)$ *since* $k_1 \neq k_2$*.*

B. Control is determined by associations

We show in that section that the outcome of tests is entirely determined by associations. This will be useful to show that, if we modify an execution (by renaming agents) while preserving enough associations, then the control flow is left unchanged.

Proposition 5. *We assume that* Π *satisfies item (i) of the well-authentication condition. Let* $\mathsf{ta} = \mathsf{ta}_0.\tau_x[a_1]$ *with* $\tau_x \in \{\tau_{\mathsf{then}},\tau_{\mathsf{else}}\}$ *be a well-formed annotated trace such that*

$$\mathcal{K}(\mathsf{ta}) \xrightarrow{\mathsf{ta}_0.\tau_x[a_1]} (\mathcal{P};\phi)$$

*and the last action (*i.e. $\tau_x[a_1]$*) is performed by an unsafe conditional. We have that* $\tau_x = \tau_{\mathsf{then}}$ *if, and only if, there exists* $a_2 \in \mathcal{A}$ *such that* a_1 *and* a_2 *are associated in* (ta_0,ϕ)*.*

Proof. ($\Rightarrow$) We start by applying Proposition 4 to obtain an execution

$$\mathcal{M}_{\Pi,\overline{id}} \xRightarrow{\mathsf{ta}_0.\tau_{\mathsf{then}}[a_1]} (\mathcal{P}';\phi) \text{ and thus } \mathcal{M}_{\Pi,\overline{id}} \xrightarrow{\mathsf{ta}_0^*.\tau_{\mathsf{then}}[a_1]} (\mathcal{P}'';\phi).$$

for some $\mathsf{ta}_0^* \stackrel{\tau}{=} \mathsf{ta}_0$. As a consequence of well-authentication, item (i) applied on the above execution, we obtain that for some $a_2 \in \mathcal{A}$, a_1 and a_2 are associated in (ta_0^*, ϕ). Since $\mathsf{ta}_0 \stackrel{\tau}{=} \mathsf{ta}_0^*$, they are also associated in (ta_0, ϕ).

($\Leftarrow$) For this other direction, we observe that (up to changes of recipes that do not affect the resulting messages) if two agents are associated in the above execution (starting with $\mathcal{K}(\mathsf{ta})$), then they are executing *the* honest trace of Π modulo a renaming of parameters, thus the considered test must be successful. We thus assume that $a_1 = A_1(\overline{k_1}, \overline{n_1})$ and $a_2 = A_2(\overline{k_2}, \overline{n_2})$ are associated in (ta_0, ϕ) we shall prove that $\tau_x = \tau_{\mathsf{then}}$. By association, $\mathsf{ta}_0|_{a_1,a_2}$ is honest: its observable actions are of the form $\mathsf{out}(c_1, w_1).\mathsf{in}(c_1', M_1) \dots \mathsf{out}(c_n, w_n).\mathsf{in}(c_n', M_n)$ with possibly an extra output at the end, and are such that $M_i\phi \Downarrow_{=\mathsf{E}} w_i\phi$ for all $1 \le i \le n$. Consider ta' obtained from ta_0 by replacing each recipe M_i by w_i. Since this change of recipes does not affect the resulting messages, the modified trace can still be executed by $\mathcal{K}(\mathsf{ta})$ and yields the same configuration $(\mathcal{P}; \phi)$. But now $\mathsf{ta}'|_{a_1,a_2}$ is a self-contained execution, *i.e.* if P and Q are the processes (possibly sub-processes) respectively annotated a_1 and a_2 in $\mathcal{K}(\mathsf{ta})$, we have:

$$(\{P[a_1], Q[a_2]\}; \varnothing) \xrightarrow{\mathsf{ta}'|_{a_1,a_2}} (\{P'[a_1], Q'[a_2]\}; \phi') \xrightarrow{\tau_x[a_1]} (\{P''[a_1], Q'[a_2]\}; \phi').$$

In the shared case (*i.e.* $fn(\mathcal{I}) \cap fn(\mathcal{R}) \neq \varnothing$), by definition of association, the identity parameters of a_1 are equal to those of a_2. Otherwise, it holds that $fn(\mathcal{I}) \cap fn(\mathcal{R}) = \varnothing$. In both cases, we thus have:

$$\begin{aligned}(\{\mathsf{new}\ \overline{k}.(\mathsf{new}\ \overline{n}_I.\mathcal{I} \mid \mathsf{new}\ \overline{n}_R.\mathcal{R})\}; \varnothing) &\xrightarrow{\tau^*} (\{P[a_1], Q[a_2]\}; \varnothing) \\ &\xrightarrow{\mathsf{ta}'|_{a_1,a_2}} (\{P'[a_1], Q'[a_2]\}; \phi') \\ &\xrightarrow{\tau_x[a_1]} (\{P''[a_1], Q'[a_2]\}; \phi').\end{aligned}$$

In that execution, everything is deterministic (up to the equational theory) and thus the execution is actually a prefix of *the* honest execution of Π (from the process P_Π defined in Definition 5), up to a bijective renaming of parameters (note that P and Q do not share session parameters). Remind that all tests must be positive in the honest execution (*i.e.* τ_{else} does not occur in the honest execution). Therefore, $\tau_x = \tau_{\mathsf{then}}$ concluding the proof. □

C. Invariance of frame idealisations

In general, a renaming of annotations can break executability: as illustrated in Example 33, mapping two dual annotations to annotations with different identities breaks the ability of the two underlying agents to communicate successfully. Moreover, even when executability is preserved, parameters change (so do names) and thus frames are modified. However, as stated next in Proposition 6, such renamings do not change idealised frames. We obtain the latter since we made sure that idealised frames only depend on what is already observable and not on specific identity or session parameters. In combination with frame opacity, this will imply (Proposition 7) that a renaming of annotations has no observable effect on the resulting *real* frames.

Proposition 6. *Let* ta *be an annotated trace such that* $\Phi_{\text{ideal}}(\mathsf{ta})$ *is well-defined. Let* ρ *be a renaming of annotations. We have that* $\Phi_{\text{ideal}}(\mathsf{ta}) \sim \Phi_{\text{ideal}}(\mathsf{ta}\rho)$.

Proof. Let ta be an annotated trace such that $\Phi_{\text{ideal}}(\mathsf{ta})$ is well-defined. Let ρ be a renaming of annotations. Let fr_1 be an arbitrary name assignment, and fr_2 be an injective function satisfying $\text{fr}_2(a\rho, x) = \text{fr}_1(a, x)$. First, we show, by induction on ta, that $\Phi^{\text{fr}_1}_{\text{ideal}}(\mathsf{ta}) = \Phi^{\text{fr}_2}_{\text{ideal}}(\mathsf{ta}\rho)$. The only interesting case is when $\mathsf{ta} = \mathsf{ta}_0.(\ell : \mathsf{out}(c, w)[a])$. In such a case, we have that:

$$\Phi^{\text{fr}_1}_{\text{ideal}}(\mathsf{ta}_0.(\ell : \mathsf{out}(c, w)[a])) = \Phi^{\text{fr}_1}_{\text{ideal}}(\mathsf{ta}_0) \cup \{w \mapsto \text{ideal}(\ell)\sigma^{\mathsf{i}}_1\sigma^{\mathsf{n}}_1\Downarrow\}$$

with $\sigma^{\mathsf{n}}_1(x^{\mathsf{n}}_j) = \text{fr}_1(a, x^{\mathsf{n}}_j)$ and $\sigma^{\mathsf{i}}_1(x^{\mathsf{i}}_j) = R_j\Phi^{\text{fr}_1}_{\text{ideal}}(\mathsf{ta}_0)$ where R_j is the j-th input of a in ta_0. The idealised frame $\Phi^{\text{fr}_2}_{\text{ideal}}(\mathsf{ta}\rho)$ is defined similarly, *i.e.*

$$\Phi^{\text{fr}_2}_{\text{ideal}}(\mathsf{ta}_0\rho.(\ell : \mathsf{out}(c, w)[a\rho])) = \Phi^{\text{fr}_2}_{\text{ideal}}(\mathsf{ta}_0\rho) \cup \{w \mapsto \text{ideal}(\ell)\sigma^{\mathsf{i}}_2\sigma^{\mathsf{n}}_2\Downarrow\}$$

with $\sigma^{\mathsf{n}}_2(x^{\mathsf{n}}_j) = \text{fr}_2(a\rho, x^{\mathsf{n}}_j)$ and $\sigma^{\mathsf{i}}_2(x^{\mathsf{i}}_j) = R^{\rho}_j\Phi^{\text{fr}_2}_{\text{ideal}}(\mathsf{ta}_0\rho)$ where R^{ρ}_j is the j-th input of $a\rho$ in $\mathsf{ta}_0\rho$. By induction hypothesis we know that $\Phi^{\text{fr}_1}_{\text{ideal}}(\mathsf{ta}_0) = \Phi^{\text{fr}_2}_{\text{ideal}}(\mathsf{ta}_0\rho)$. Therefore, to conclude, it remains to show that $\text{ideal}(\ell)\sigma^{\mathsf{i}}_1\sigma^{\mathsf{n}}_1 = \text{ideal}(\ell)\sigma^{\mathsf{i}}_2\sigma^{\mathsf{n}}_2$. Actually, we have that $R^{\rho}_j = R_j$, thus $\sigma^{\mathsf{i}}_1(x^{\mathsf{i}}_j) = \sigma^{\mathsf{i}}_2(x^{\mathsf{i}}_j)$, and $\sigma^{\mathsf{n}}_2(x^{\mathsf{n}}_j) = \text{fr}_2(a\rho, x^{\mathsf{n}}_j) = \text{fr}_1(a, x^{\mathsf{n}}_j) = \sigma^{\mathsf{n}}_1(x^{\mathsf{n}}_j)$.

We have shown that $\Phi^{\text{fr}_1}_{\text{ideal}}(\mathsf{ta}) = \Phi^{\text{fr}_2}_{\text{ideal}}(\mathsf{ta}\rho)$ and relying on Proposition 1, we easily deduce that $\Phi_{\text{ideal}}(\mathsf{ta}) \sim \Phi_{\text{ideal}}(\mathsf{ta}\rho)$. □

Proposition 7. *We assume that* Π *satisfies frame opacity. Let* ρ *be a renaming of annotations and* ta *be a well-formed annotated trace. If* $\mathcal{K}(\mathsf{ta}) \overset{\mathsf{ta}}{\Longrightarrow} (\mathcal{P}_1; \phi_1)$ *and* $\mathcal{K}(\mathsf{ta}\rho) \overset{\mathsf{ta}\rho}{\Longrightarrow} (\mathcal{P}_2; \phi_2)$*, then we have that* $\phi_1 \sim \phi_2$*.*

Proof. We start by applying Proposition 4 on the two given executions to obtain two executions starting from $\mathcal{M}_{\Pi,\overline{id}}$:

- $\mathcal{M}_{\Pi,\overline{id}} \xrightarrow{\mathsf{ta}^*} (\mathcal{P}'_1; \phi_1)$ with $\mathsf{ta}^* \overset{\tau}{=} \mathsf{ta}$; and
- $\mathcal{M}_{\Pi,\overline{id}} \xrightarrow{\mathsf{ta}^*_\rho} (\mathcal{P}'_2; \phi_2)$ with $\mathsf{ta}^*_\rho \overset{\tau}{=} \mathsf{ta}\rho$.

Relying on frame opacity, we know that $\Phi_{\text{ideal}}(\mathsf{ta}^*)$ (resp. $\Phi_{\text{ideal}}(\mathsf{ta}^*_\rho)$) is well-defined and $\Phi_{\text{ideal}}(\mathsf{ta}^*) \sim \phi_1$ (resp. $\Phi_{\text{ideal}}(\mathsf{ta}^*_\rho) \sim \phi_2$).

Note that, if ta_1 and ta_2 are two annotated traces such that $\mathsf{ta}_1 \overset{\tau}{=} \mathsf{ta}_2$ and fr_1 is a name assignment, then $\Phi^{\text{fr}_1}_{\text{ideal}}(\mathsf{ta}_1) = \Phi^{\text{fr}_1}_{\text{ideal}}(\mathsf{ta}_2)$. Thanks to this remark, we easily deduce that $\Phi_{\text{ideal}}(\mathsf{ta}) \sim \Phi_{\text{ideal}}(\mathsf{ta}^*)$ and $\Phi_{\text{ideal}}(\mathsf{ta}\rho) = \Phi_{\text{ideal}}(\mathsf{ta}^*_\rho)$. Thanks to Proposition 6, we know that $\Phi_{\text{ideal}}(\mathsf{ta}) \sim \Phi_{\text{ideal}}(\mathsf{ta}\rho)$ and by transitivity of $\sim$, we conclude that $\phi_1 \sim \phi_2$. □

D. A sufficient condition for preserving executability

We can now state a key lemma (Lemma 2), identifying a class of renamings which yield indistinguishable executions. More precisely, this lemma shows that for renamings satisfying some requirements, if $\mathcal{K}(\mathsf{ta})$ can execute an annotated trace ta then $\mathcal{K}(\mathsf{ta})\rho$ has an indistinguishable execution following the annotated trace $\mathsf{ta}\rho$. Remark that, in the conclusion, the renaming is applied after building the ground configuration $(\mathcal{K}(\mathsf{ta})\rho)$ instead of building the ground configuration

of the renamed trace ($\mathcal{K}(\mathsf{ta}\rho)$). Both variants are a priori different. However, in the final proof and in order to leverage previous propositions, we will need to relate executions of $\mathcal{K}(\mathsf{ta})$ with executions of $\mathcal{K}(\mathsf{ta}\rho)$. The following easy proposition bridges this gap. We also state and prove a variant of Proposition 4, item (4) when ρ is applied after building the ground configuration.

Proposition 8. *Let* ta *be a well-formed annotated trace and* ρ *be a renaming of annotations.*
If $\mathcal{K}(\mathsf{ta})\rho \xrightarrow{\mathsf{ta}\rho} (\mathcal{P}';\phi)$ *then* $\mathcal{K}(\mathsf{ta}\rho) \overset{\mathsf{ta}\rho}{\Longrightarrow} (\mathcal{P}'';\phi)$.

Proof. Essentially, the proposition follows from the fact that there are less agents in sequence in $\mathcal{K}(\mathsf{ta}\rho)$ than in $\mathcal{K}(\mathsf{ta})\rho$, thanks to the third item of Definition 23.

More formally, by considering $\mathcal{K}(\mathsf{ta}\rho)$ and $\mathcal{K}(\mathsf{ta})\rho$ as multiset of processes without sequence (by removing all sequences and taking the union of processes), we obtain the same multisets. Next, it suffices to prove that no execution is blocked by a sequence in $\mathcal{K}(\mathsf{ta}\rho)$. By definition of the renaming ρ (third requirement in Definition 23), if an agent $P[\rho(a)]$ occurring in $\mathcal{K}(\mathsf{ta}\rho)$ is in sequence with an agent $\rho(a')$ before him, then $P[a\rho]$ occurring in $(\mathcal{K}(\mathsf{ta})\rho)$ must be in sequence with a before him as well. Hence, when a process $P[\rho(a)];Q$ is available (*i.e.* at top-level) at some point in the execution from $\mathcal{K}(\mathsf{ta})\rho$, then a similar process $P[\rho(a)];Q'$ is also available at the same point in the execution from $\mathcal{K}(\mathsf{ta}\rho)$. However, a process may become available in the execution from $\mathcal{K}(\mathsf{ta})\rho$ only after having performed rule Seq, while the same process may be immediately available in the multiset in the execution from $\mathcal{K}(\mathsf{ta}\rho)$. This is why we only obtain a weak execution $\mathcal{K}(\mathsf{ta}\rho) \overset{\mathsf{ta}\rho}{\Longrightarrow} (\mathcal{P}'';\phi)$. □

Proposition 9. *If* $\mathsf{ta} = \mathsf{ta}_1.\mathsf{ta}_2$ *is a well-formed annotated trace and* $\mathcal{K}(\mathsf{ta}_1.\mathsf{ta}_2)\rho \xrightarrow{\mathsf{ta}_1\rho} K$ *then* $\mathcal{K}(\mathsf{ta}_1)\rho \xrightarrow{\mathsf{ta}_1\rho} K'$ *with* $\phi(K) = \phi(K')$.

Proof. The argument is the same as for Proposition 4, item (4): the processes that are added to $\mathcal{K}(\mathsf{ta}_1)\rho$ when considering $\mathcal{K}(\mathsf{ta}_1.\mathsf{ta}_2)\rho$ are unused in the execution of $\mathsf{ta}_1\rho$; moreover, the effect of ρ on the processes of $\mathcal{K}(\mathsf{ta}_1)$ is obviously the same as in $\mathcal{K}(\mathsf{ta}_1.\mathsf{ta}_2)$. □

Finally, after having defined the notion of *connection* between agents, we can state our key lemma.

Definition 24. *Annotations* a *and* a' *are* connected *in* (ta,ϕ) *if they are associated in* (ta_0,ϕ) *for some prefix* ta_0 *of* ta *that contains at least one* τ_{then} *action of an unsafe conditional annotated with either* a *or* a'.

Lemma 2. *We assume that* Π *satisfies frame opacity and item (i) of well-authentication. Let* ta *be a well-formed annotated trace such that* $\mathcal{K}(\mathsf{ta}) \xrightarrow{\mathsf{ta}} (\mathcal{P};\phi)$. *Let* ρ *be a renaming of annotations. Moreover, when* $fn(\mathcal{I}) \cap fn(\mathcal{R}) \neq \varnothing$ *(shared case), we assume that for any annotations* a, a', *it holds that* a *and* a' *are connected in* (ta,ϕ), *if, and only if,* $\rho(a)$ *and* $\rho(a')$ *are dual.*
We have that $\mathcal{K}(\mathsf{ta})\rho \xrightarrow{\mathsf{ta}\rho} (\mathcal{Q};\psi)$ *for some* ψ *such that* $\phi \sim \psi$.

Proof. The ground configurations $\mathcal{K}(\mathsf{ta})$ and $\mathcal{K}(\mathsf{ta})\rho$ have the same shape: these processes only differ by their terms. Thus, we can put them together to form a bi-process[10], *i.e.* a process

[10]The bi-process considered here does not make use of the extension of diff-equivalence presented before: its inputs are of the form $\mathsf{in}(c,x)$ and not $\mathsf{in}(c,\mathsf{choice}[x,y])$.

in which terms are bi-terms of the form $\mathsf{choice}(t_1, t_2)$. Given a bi-process B, we denote $\mathsf{fst}(B)$ (resp. $\mathsf{snd}(B)$) the process obtained from B by replacing any bi-term $\mathsf{choice}[t_1, t_2]$ by t_1 (resp. t_2). Moreover, we write $B \xrightarrow{\alpha[\mathsf{choice}[a,a']]} B'$ to indicate that $\mathsf{fst}(B)$ executes α with annotation a and $\mathsf{snd}(B)$ performs α in the same way but with annotation a'. More generally, we write $B \xrightarrow{\mathsf{choice}[\mathsf{ta},\mathsf{ta}']} B'$ to indicate that the bi-process B executes the trace ta on the left and ta' on the right, where the two traces differ only in their annotations.

For the sake of simplicity, we decorate outputs of the biprocess obtained from $\mathcal{K}(\mathsf{ta})$ and $\mathcal{K}(\mathsf{ta})\rho$ with pairwise distinct handles from $\mathcal{W}$, and we assume that the handle that decorates an output will be used to store the output message when it will be executed. Lastly, we associate a vector of terms in $\mathcal{T}(\Sigma_{\mathsf{pub}}, \mathcal{W} \cup \mathcal{X})$ to each safe conditional of the protocol: recall that $\mathsf{let}\ \overline{z} = \overline{t}\ \mathsf{in}\ P\ \mathsf{else}\ Q$ is identified as a safe conditional only if there exists a sequence $\overline{R}$ of terms in $\mathcal{T}(\Sigma_{\mathsf{pub}}, \mathcal{W} \cup \mathcal{X})$ such that $\overline{R}\{w_1 \mapsto u_1, \ldots, w_m \mapsto u_m\} = \overline{t}$ where u_i are the messages used in outputs occurring before the conditional and w_i is the handle associated to u_i; $\overline{R}$ is the vector of terms associated to the safe conditional. Note that $\overline{R}$ may contain variables from $\mathcal{X}$ corresponding to inputs performed before the conditional, which will be instantiated by ground terms before the execution of the conditional. When executing a process with labels on conditionals, we assume that the execution of an input $\mathsf{in}(c, x)$ with recipe R_{in} will instantiate the variable x occurring in the label of the conditional with R_{in}.

For any prefix $\mathcal{K}(\mathsf{ta}) \xrightarrow{\mathsf{ta}_0} (\mathcal{P}_0; \phi_0)$ of the execution $\mathcal{K}(\mathsf{ta}) \xrightarrow{\mathsf{ta}} (\mathcal{P}; \phi)$, we prove that there exists an execution $\mathcal{K}(\mathsf{ta})\rho \xrightarrow{\mathsf{ta}_0\rho} (\mathcal{Q}_0; \psi_0)$ such that:

(a) $B \xrightarrow{\mathsf{choice}[\mathsf{ta}_0,\mathsf{ta}_0\rho]} B_0$ with $\mathsf{fst}(B_0) = (\mathcal{P}_0; \phi_0)$ and $\mathsf{snd}(B_0) = (\mathcal{Q}_0; \psi_0)$.

(a') Any bi-conditional $\mathsf{let}\ \overline{z} = \mathsf{choice}[\overline{t}_l, \overline{t}_r]\ \mathsf{in}\ B_P\ \mathsf{else}\ B_Q$ labeled with $\overline{R}$ is such that $\overline{R} = \overline{C}[R_1, \ldots, R_k]$ with $\overline{C}$ a sequence of contexts built on Σ_{pub}, and $R_i \in \mathcal{T}(\Sigma_{\mathsf{pub}}, \mathcal{W} \cup \mathcal{X})$ is either a variable in $\mathcal{X}$ or a $w \notin dom(\phi_0)$ or a recipe i.e. a term in $\mathcal{T}(\Sigma_{\mathsf{pub}}, dom(\phi_0))$. Moreover, assuming that $u \Downarrow u$ for any constructor term (even if it contains some variables), we have that $\overline{C}[R_1\phi_0^+ \Downarrow, \ldots, R_k\phi_0^+ \Downarrow] = \overline{t}_l$ and $\overline{C}[R_1\psi_0^+ \Downarrow, \ldots, R_k\psi_0^+ \Downarrow] = \overline{t}_r$ where ϕ_0^+ (resp. ψ_0^+) is ϕ_0 (resp. ψ_0) extended with $w \mapsto u$ for each output $\mathsf{out}(c, u)$ decorated with w preceding the conditional in $\mathsf{fst}(B_0)$ (resp. $\mathsf{snd}(B_0)$).

(b) $\phi_0 \sim \psi_0$;

(c_1) when $fn(\mathcal{I}) \cap fn(\mathcal{R}) = \varnothing$ (non-shared case), $\rho(a)$ and $\rho(a')$ are associated in $(\mathsf{ta}_0\rho, \psi_0)$ if, and only if, a and a' are associated in (ta_0, ϕ_0);

(c_2) when $fn(\mathcal{I}) \cap fn(\mathcal{R}) \neq \varnothing$ (shared case), $\rho(a)$ and $\rho(a')$ are associated in $(\mathsf{ta}_0\rho, \psi_0)$ if, and only if, a and a' are associated in (ta_0, ϕ_0) and connected in (ta, ϕ).

We proceed by induction on the prefix ta_0 of ta.

Case ta_0 *is empty.* In such a case, $\mathsf{ta}_0\rho$ can obviously be executed. Condition (b) is trivial since both frames are empty. In order to check conditions (c_1) and (c_2), note that association coincides with duality for empty traces. We start by establishing condition (c_1). Being dual simply means being distinct roles, hence one obviously has that $\rho(a)$ and $\rho(a')$ are dual if, and only if, a and a' are. This allows us to conclude for condition (c_1). Now, we establish condition (c_2). By hypothesis, we have that a and a' are connected in (ta, ϕ) if, and only if, $\rho(a)$ and $\rho(a')$ are dual, this allows us to conclude regarding one direction. Now, if $\rho(a)$ and $\rho(a')$ are dual, then a and a'

are dual too by definition of an agent renaming. Hence, we have the other direction. Condition (a) holds and condition (a') holds since by definition of being safe, we have the expected $\overline{R}$.

Case the prefix of ta *is of the form* $\mathsf{ta}_0.\alpha$. The action α may be an input, an output, a conditional (*i.e.* τ_{then} or τ_{else}). By (a), we know that there is a process in $\mathcal{Q}_0$ which is able to perform an action of the same nature. We consider separately the case where α is an input, an output or a conditional. In those cases α is necessarily annotated, say by a, and has been produced by a process annotated a in $\mathcal{P}_0$. By induction hypothesis we have $(\mathcal{P}_0;\phi_0)$ and $(\mathcal{Q}_0;\psi_0)$ and the following executions

$$\mathcal{K}(\mathsf{ta}) \xrightarrow{\mathsf{ta}_0} (\mathcal{P}_0;\phi_0) \xrightarrow{\alpha[a]} (\mathcal{P}_0';\phi_0') \text{ and } \mathcal{K}(\mathsf{ta})\rho \xrightarrow{\mathsf{ta}_0\rho} (\mathcal{Q}_0;\psi_0)$$

satisfying all our invariants. Note that one has $(\mathsf{ta}_0.\alpha[a])\rho = (\mathsf{ta}_0\rho).\alpha[\rho(a)]$. Moreover, we have that $B \xrightarrow{\mathsf{choice}[\mathsf{ta}_0,\mathsf{ta}_0\rho]} B_0$ with $\mathsf{fst}(B_0) = (\mathcal{P}_0;\phi_0)$ and $\mathsf{snd}(B_0) = (\mathcal{Q}_0;\psi_0)$. Now, we have to prove that there exists B_0' such that $B_0 \xrightarrow{\mathsf{choice}[\alpha,\alpha\rho]} B_0'$ with $\mathsf{fst}(B_0') = (\mathcal{P}_0';\phi_0')$.

Case where α *is an output.* We immediately have that $\mathcal{Q}_0$ can perform $\alpha[\rho(a)]$, on the same channel and with the same handle. We now have to check our invariants for $\mathsf{ta}_0.\alpha[a]$. Let $\mathsf{out}(c,w)[a]$ be α. Conditions (a) is obviously preserved. Regarding condition (a'), it is easy to see that the result holds. The term that was added in ϕ_0^+ (resp. ψ_0^+) is now present in ϕ_0' (resp. ψ_0').

Conditions (c_1) and (c_2) follow from the fact that association is not affected by the execution of an output: $\rho(a)$ and $\rho(a')$ are associated in $(\mathsf{ta}_0.\alpha[a])\rho$ if, and only if, they are associated in $\mathsf{ta}_0\rho$, and similarly without ρ. Finally, we shall prove (b): $\phi_0' \sim \psi_0'$ where ϕ_0' (resp. ψ_0') is the resulting frame after the action $\alpha[a]$ (resp. $\alpha[a\rho]$). Applying Proposition 4 item (4) on the execution before renaming and Proposition 9 on the execution after renaming, one obtains

$$\mathcal{K}(\mathsf{ta}_0.\alpha[a]) \xrightarrow{\mathsf{ta}_0.\alpha[a]} (\mathcal{P}_0'';\phi_0') \text{ and } \mathcal{K}(\mathsf{ta}_0.\alpha[a])\rho \xrightarrow{(\mathsf{ta}_0\rho).\alpha[a\rho]} (\mathcal{Q}_0';\psi_0').$$

We finally conclude $\phi_0' \sim \psi_0'$ using Proposition 8 on the execution on the right and then Proposition 7.

Case where α *is a conditional.* We first need to make sure that the outcome of the test is the same for a and $a\rho$. We let τ_x (resp. τ_y) be the action produced by evaluating the conditional of a (resp. $a\rho$) and shall prove that $\tau_x = \tau_y$. We distinguish two cases depending on whether the conditional has a label (*i.e.* has been identified as safe) or not.

- If the conditional has a label $\overline{R}$, since this conditional is now at toplevel, we know that $\phi_0^+ = \phi_0$ and $\psi_0^+ = \psi_0$, and $\overline{R} = \overline{C}[R_1,\ldots,R_k]$ only contains variables from $dom(\phi_0) = dom(\psi_0)$. On the left, we have that the conditional will be evaluated to true iff $\bar{t}_l \Downarrow$ is a message, i.e. iff $\overline{C}[R_1\phi_0^+\Downarrow,\ldots,R_k\phi_0^+\Downarrow]\Downarrow$ is a message, i.e. iff $\overline{C}[R_1,\ldots,R_k]\phi_0\Downarrow$ is a message and similarly on the right. Since $\phi_0 \sim \psi_0$, this allows us to conclude that the two conditionals have the same outcome.
- If the conditional is unsafe, we make use of Proposition 5 to show that the outcome of the conditional is the same on both sides. First, we deduce the following executions from

Proposition 4 item (4) applied on the execution before renaming and Proposition 9 applied on the execution after renaming:

$$\mathcal{K}(\mathsf{ta}_0.\tau_x[a]) \xrightarrow{\mathsf{ta}_0.\tau_x[a]} (\mathcal{P}_0''; \phi_0) \text{ and } \mathcal{K}(\mathsf{ta}_0.\tau_y[a])\rho \xrightarrow{(\mathsf{ta}_0\rho).\tau_y[a\rho]} (\mathcal{Q}_0'; \psi_0).$$

To infer $\tau_x = \tau_y$ from Proposition 5, it remains to prove that a and a' are associated in (ta_0, ϕ_0) if, and only if, $\rho(a)$ and $\rho(a')$ are associated in $(\mathsf{ta}_0\rho, \psi_0)$. When $fn(\mathcal{I}) \cap fn(\mathcal{R}) = \varnothing$ (non-shared case), this is given by the invariant (c_1). Otherwise, when $fn(\mathcal{I}) \cap fn(\mathcal{R}) \neq \varnothing$ (shared case), (c_2) gives us that $\rho(a)$ and $\rho(a')$ are associated in $(\mathsf{ta}_0\rho, \psi_0)$ if, and only if, a and a' are associated in (ta_0, ϕ_0) and connected in (ta, ϕ). Therefore, to conclude, it is actually sufficient to show that if a and a' are associated in (ta_0, ϕ_0) then $\rho(a)$ and $\rho(a')$ are associated in $(\mathsf{ta}_0\rho, \psi_0)$. Since a and a' are associated in (ta_0, ϕ_0), thanks to Proposition 5, we know that the outcome of the test will be positive (*i.e.* $\tau_x = \tau_{\mathsf{then}}$), and thus a and a' are connected in $(\mathsf{ta}_0.\tau_{\tau_{\mathsf{then}}}, \phi_0)$, and therefore a and a' are also connected in (ta, ϕ). Thanks to (c_2) we have that $\rho(a)$ and $\rho(a')$ are associated in $(\mathsf{ta}_0\rho, \psi_0)$, and we are done.

After the execution of this conditional producing $\tau_x = \tau_y$, condition (a) and (a') obviously still hold since $\tau_x = \tau_y$ implying that both agents go to the same branch of the conditional. Invariant (b) is trivial since frames have not changed. Conditions (c_1) and (c_2) are preserved because the association between a and a' is preserved if, and only if, the outcome of the test is positive, which is the same before and after the renaming.

Case where $\alpha = \mathsf{in}(c, R_{in})$ is an input. We immediately have that $\mathcal{Q}_0$ can perform $\alpha[\rho(a)]$ on the same channel and with the same recipe R_{in} (since $dom(\phi_0) = dom(\psi_0)$ follows from $\phi_0 \sim \psi_0$). Conditions (a) and (b) are obviously preserved. Let us establish Condition (a'). We consider a bi-conditional $\mathsf{let}\ \overline{z} = \mathsf{choice}[\overline{t}_l, \overline{t}_r]\ \mathsf{in}\ B_P\ \mathsf{else}\ B_Q$ labeled with $\overline{R}$ before executing α. Thus, we know that (a') holds, i.e. $\overline{R} = \overline{C}[R_1, \ldots, R_k]$ such that $\overline{C}[R_1\phi_0^+ \Downarrow, \ldots, R_k\phi_0^+ \Downarrow] = \overline{t}_l$ and $\overline{C}[R_1\psi_0^+ \Downarrow, \ldots, R_k\psi_0^+ \Downarrow] = \overline{t}_r$. After executing α, either this conditional is kept unchanged and the result trivially holds, or $\overline{t}_l$ becomes $\overline{t}_l\{x \mapsto R_{in}\phi_0 \Downarrow\}$ and we have that the label of this conditional is now $\overline{R}' = \overline{R}\{x \mapsto R_{in}\}$, and we have that $R_{in}\phi_0 \Downarrow$ (and thus $R_{in}\psi_0 \Downarrow$) is a message. To conclude, it remains to show that $\overline{C}[R_1\{x \mapsto R_{in}\}\phi_0^+ \Downarrow, \ldots, R_k\{x \mapsto R_{in}\}\phi_0^+ \Downarrow] = \overline{t}_l\{x \mapsto R_{in}\phi_0 \Downarrow\}$ (and similarly for ψ_0). Either $R_i = x$ and we have that $R_i\{x \mapsto R_{in}\}\phi_0^+ \Downarrow = R_{in}\phi_0 \Downarrow = (R_i\phi_0^+ \Downarrow)\{x \mapsto R_{in}\phi_0 \Downarrow\}$. Otherwise, R_i does not contain x, and we have that $R_i\{x \mapsto R_{in}\}\phi_0^+ \Downarrow = R_i\phi_0^+ \Downarrow = (R_i\phi_0^+ \Downarrow)\{x \mapsto R_{in}\phi_0 \Downarrow\}$.

Thus, we have that $\overline{C}[R_1\{x \mapsto R_{in}\}\phi_0^+ \Downarrow, \ldots, R_k\{x \mapsto R_{in}\}\phi_0^+ \Downarrow] = \overline{C}[(R_1\phi_0^+ \Downarrow)\{x \mapsto R_{in}\phi_0\}, \ldots, (R_k\phi_0^+ \Downarrow)\{x \mapsto R_{in}\phi_0\}] = \overline{C}[R_1\phi_0^+ \Downarrow, \ldots, R_1\phi_0^+ \Downarrow]\{x \mapsto R_{in}\phi_0 \Downarrow\} = \overline{t}_l\{x \mapsto R_{in}\phi_0 \Downarrow\}$. This allows us to conclude.

Conditions (c_1) and (c_2) are preserved because honest interactions are preserved by the renaming, since $\phi_0 \sim \psi_0$ by invariant (b). We only detail one direction of (c_1), the other cases being similar. Assume that $\rho(a)$ and $\rho(a')$ are associated in $((\mathsf{ta}_0.\alpha[a])\rho, \psi_0)$. The renamed agents $\rho(a)$ and $\rho(a')$ are also associated in $(\mathsf{ta}_0\rho, \phi')$, thus a and a' are associated in (ta_0, ϕ'). Now, because α did not break the association of $\rho(a)$ and $\rho(a')$ in $(\mathsf{ta}_0\rho, \psi_0)$, it must be that the input message in $\alpha = \mathsf{in}(c, R_{in})$ corresponds to the last output of $\rho(a')$ in $\mathsf{ta}_0\rho$. Formally, if that last output corresponds to the handle w in ψ_0, we have $R_{in}\psi_0 \Downarrow_{=_{\mathsf{E}}} w\psi_0$. But, because $\phi_0 \sim \psi_0$ by invariant (b), we then also have $M\phi_0 \Downarrow_{=_{\mathsf{E}}} w\phi_0$ and the association of a and a' in (ta_0, ϕ_0) carries over to $(\mathsf{ta}_0.\alpha[a], \phi_0)$. □

E. Final proof

Thanks to Lemma 2, we can transform any execution of $\mathcal{M}_{\Pi,\overline{id}}$ into an indistinguishable execution of $\mathcal{S}_\Pi$, provided that an appropriate renaming of annotations exists. In order to prove that such a renaming exists in Proposition 11, we show below that in the shared case, agents cannot be connected multiple times.

Proposition 10. *Assume that* Π *satisfies item (ii) of well-authentication and that* $fn(\mathcal{I}) \cap fn(\mathcal{R}) \neq \varnothing$ *(shared case). Consider a well-formed annotated trace* ta *and an execution* $\mathcal{K}(\mathsf{ta}) \xrightarrow{\mathsf{ta}} (\mathcal{P};\phi)$. *Let* a, a_1, *and* a_2 *be three annotations such that* a *and* a_1 *(resp.* a *and* a_2*) are connected in* (ta,ϕ), *then we have that* $a_1 = a_2$.

Proof. Consider the first unsafe conditional performed in ta by one of the three agents a, a_1 or a_2. This conditional exists and must be successful, otherwise the agents would not be connected in ta. In other words, we have $\mathsf{ta} = \mathsf{ta}'.\tau_{\mathsf{then}}[x].\mathsf{ta}''$ where $x \in \{a, a_1, a_2\}$ and ta' does not contain any unsafe conditional performed by one of our three agents. Since a is connected with both a_1 and a_2 in ta, it is associated with both a_1 and a_2 in ta'. This contradicts condition (ii) of well-authentication applied to $\mathsf{ta}'.\tau_{\mathsf{then}}[x]$. □

Proposition 11. *Let* Π *be a protocol satisfying item (ii) of well-authentication, and* ta *be a well-formed annotated trace such that* $\mathcal{K}(\mathsf{ta}) \xrightarrow{\mathsf{ta}} K$. *There exists a renaming of annotations* ρ *such that:*

- $\mathsf{ta}\rho$ *is single-session;*
- *Moreover, when* $fn(\mathcal{I}) \cap fn(\mathcal{R}) \neq \varnothing$ *(shared case), for any annotations* a *and* a', *we have that* a *and* a' *are connected in* (ta,ϕ), *if, and only if,* $\rho(a)$ *and* $\rho(a')$ *are dual.*

Proof. For $\overline{k} \in \mathrm{id}_I(\mathsf{ta}) \cup \mathrm{id}_R(\mathsf{ta})$, we define $\mathrm{Co}(\overline{k})$ as follows:

- when $fn(\mathcal{I}) \cap fn(\mathcal{R}) = \varnothing$ (non-shared case), we let $\mathrm{Co}(\overline{k})$ be the empty set.
- when $fn(\mathcal{I}) \cap fn(\mathcal{R}) \neq \varnothing$ (shared case), we let $\mathrm{Co}(\overline{k})$ be the set of all $(\overline{n_1},\overline{n_2})$ such that $I(\overline{k},\overline{n_1})$ and $R(\overline{k},\overline{n_2})$ are connected in $(\mathsf{ta},\phi(K))$;

Essentially, $\mathrm{Co}(\overline{k})$ denotes the set of pairs of (dual) sessions that the renaming to be defined should keep on the same identity. Applying Proposition 10, we deduce that for any $\overline{k} \in \mathrm{id}_A(\mathsf{ta})$ and $(\overline{n_1},\overline{n_2}),(\overline{n_3},\overline{n_4}) \in \mathrm{Co}(\overline{k})$, then either (i) $\overline{n_1} = \overline{n_3}$ and $\overline{n_2} = \overline{n_4}$ or (ii) $\overline{n_1} \neq \overline{n_3}$ and $\overline{n_2} \neq \overline{n_4}$.

Next, we assume the existence of a function $k^c : \mathcal{N}^* \times \mathcal{N}^* \times^* \mathcal{N}^* \mapsto \mathcal{N}^*$ that associates to any sequence of names $(\overline{k},\overline{n_1},\overline{n_2})$ a vector of names of the length of identity parameters of Π: $\overline{k'} = k^c(\overline{k},\overline{n_1},\overline{n_2})$. These name vectors are assumed to be all disjoint and not containing any name already occurring in the annotations of ta. This gives us a mean to pick fresh identity parameters for each combination of $\overline{k},\overline{n_1},\overline{n_2}$ taken from the annotations of ta. We also assume a function k^1 such that the vectors $k^1(\overline{k},\overline{n_1})$ are again disjoint and not overlapping with annotations of ta and any $k^c(\overline{k'},\overline{n_1'},\overline{n_2'})$, and similarly for $k^2(\overline{k},\overline{n_2})$ which should also not overlap with k^1 vectors. These last two collections of identity parameters will be used to give fresh identities to initiator

and responder agents, independently. We then define ρ as follows:

$$\begin{aligned} I(\overline{k},\overline{n}_1) &\mapsto I(k^c(\overline{k},\overline{n_1},\overline{n_2}),\overline{n_1}) && \text{if } (\overline{n_1},\overline{n_2}) \in \mathrm{Co}(\overline{k}) \\ &\mapsto I(k^1(\overline{k},\overline{n_1}),\overline{n_1}) && \text{otherwise} \\ R(\overline{k},\overline{n}_2) &\mapsto R(k^c(\overline{k},\overline{n_1},\overline{n_2}),\overline{n_2}) && \text{if } (\overline{n_1},\overline{n_2}) \in \mathrm{Co}(\overline{k}) \\ &\mapsto R(k^2(\overline{k},\overline{n_2}),\overline{n_2}) && \text{otherwise} \end{aligned}$$

We now prove that the renaming ρ defined above satisfies all requirements.

The mapping ρ is a renaming. First, for any well-formed ta', the fact that $\mathsf{ta}'\rho$ is well-formed follows from the following: (i) session names are not modified and (ii) identity names are all pairwise distinct and never intersect except for agents $\rho(I(\overline{k},\overline{n_1}))$ and $\rho(R(\overline{k},\overline{n_2}))$ such that $(\overline{n_1},\overline{n_2}) \in \mathrm{Co}(\overline{k})$ but there cannot be a third agent sharing the same identity names according to the result obtained above. The mapping ρ is obviously role-preserving. Finally, if $\rho(a)$ and $\rho(a')$ share the same identity parameters then (a) $fn(\mathcal{I}) \cap fn(\mathcal{R}) \neq \varnothing$ and (b) a and a' are connected in (ta,ϕ) and are thus dual implying that a and a' share the same identity parameters as well.

The renaming ρ is single-session. First, id_0 never occurs in the image of ρ. Second, all agents are mapped to agents having fresh, distinct identity parameters except for agents a, a' that were connected in (ta,ϕ). However, as already discussed, in such a case, there is no third agent sharing those identity parameters and a and a' are necessarily dual.

To conclude, we shall prove that, in the shared case, for any annotations a, a', it holds that a and a' are connected in (ta,ϕ), if, and only if, $\rho(a)$ and $\rho(a')$ are dual. This is actually a direct consequence of the definition of the renaming ρ. □

We are now able to prove our main theorem.

Proof of Theorem 1. We have that $\mathcal{S}_\Pi \sqsubseteq \mathcal{M}_\Pi \sqsubseteq \mathcal{M}_{\Pi,\overline{id}}$, and we have to establish that $\mathcal{M}_{\Pi,\overline{id}} \sqsubseteq \mathcal{S}_\Pi$. Consider an execution $\mathcal{M}_{\Pi,\overline{id}} \xrightarrow{\mathsf{ta}} (\mathcal{P};\phi)$. First, thanks to Proposition 4, there is an annotated trace $\mathsf{ta}' \overset{\tau}{=} \mathsf{ta}$ such that $\mathcal{K}(\mathsf{ta}) \xrightarrow{\mathsf{ta}'} (\mathcal{P}';\phi)$. Let ρ be the renaming obtained in Proposition 11 for ta'. By Lemma 2, we have that $\mathcal{K}(\mathsf{ta})\rho \xrightarrow{\mathsf{ta}'\rho} (\mathcal{Q};\phi_\rho)$ for some frame ϕ_ρ such that $\phi_\rho \sim \phi$. We then deduce from Proposition 8 that $\mathcal{K}(\mathsf{ta}\rho) \xRightarrow{\mathsf{ta}'\rho} (\mathcal{Q}';\phi_\rho)$, and thus $\mathcal{K}(\mathsf{ta}\rho) \overset{\mathsf{ta}\rho}{\Longrightarrow} (Q';\phi_\rho)$. Since $\mathsf{ta}\rho$ is single-session, Proposition 4 implies that $(\mathcal{S}_\Pi;\varnothing) \overset{\mathsf{ta}\rho}{\Longrightarrow} (\mathcal{Q}'';\phi_\rho)$, and this allows us to conclude. □