A Method for Proving Unlinkability of Stateful Protocols

David Baelde, Stéphanie Delaune, Solène Moreau

A Method for Proving Unlinkability of Stateful Protocols

David Baelde*, Stéphanie Delaune† and Solène Moreau†
* LSV, ENS Paris-Saclay & CNRS, Université Paris-Saclay, France
†Univ Rennes, CNRS, IRISA, France

***Abstract*—The rise of contactless and wireless devices such as mobile phones and RFID chips justifies significant concerns over privacy, and calls for communication protocols that ensure some form of unlinkability. Formally specifying this property is difficult and context-dependent, and analysing it is very complex; as is common with security protocols, several incorrect unlinkability claims can be found in the literature. Formal verification is therefore desirable, but current techniques are not sufficient to directly analyse unlinkability. In [21], two conditions have been identified that imply unlinkability and can be automatically verified. That work, however, only considers a restricted class of protocols. We adapt their formal definition as well as their proof method to the common setting of RFID authentication protocols, where readers access a central database of authorised users. Moreover, we also consider protocols where readers may update their database, and tags may also carry a mutable state. We propose sufficient conditions to ensure unlinkability, find new attacks, and obtain new proofs of unlinkability using Tamarin to establish our sufficient conditions.**

I. Introduction

In our societies, practically everybody carries digital devices whose communications may happen, unnoticed, at any time. While most of these communications rely on cryptography to ensure secrecy or authenticity, little is done to protect the user's privacy. Contactless cards and cell phones reveal identities in clear [30], [31], and several traceability attacks are available even when anonymity is ensured [6], [7], [21], [13]. It has thus become easy to track individuals through their personal devices. To avoid such threats to privacy, we need protocols that ensure *unlinkability* [22]: an outside observer must not be able to tell whether two uses of the protocol are related or not. Privacy concerns and the need for unlinkability are slowly being accepted by the industry, *e.g.* the 5G PPP consortium [4].

Designing unlinkable protocols does not require cutting-edge cryptography, but it is a very difficult task. As is generally true of security protocols, defending against an arbitrary active attacker involves too many details that are easily overlooked. History has told us that formal methods are very useful in that domain, both to find attacks and to obtain security proofs that can be trusted. This is particularly true with unlinkability, for several reasons. First, the informal notion of unlinkability does not translate to a single formal definition. Several definitions of unlinkability have been proposed, *e.g.* [34], [23], [32], [6], [15], [16], [21]: leaving aside the different formal models, these definitions provide varying degrees of security, corresponding to different attack scenarios. Given a protocol with some intended use case, it is not immediately obvious which definition provides strong enough guarantees. Second, most definitions of unlinkability rely on some form of behavioural equivalence, which makes proofs even more difficult — this is a general problem when dealing with privacy notions, which has motivated a recent alternative approach which can avoid equivalences in some cases, not including unlinkability so far [28]. Proofs of equivalences are cumbersome, and we believe that they cannot realistically be carried out in details by hand. Regarding mechanisation, several mature tools are available for analysing trace properties such as secrecy or authentication: for instance, ProVerif [11], [1], the Avantssar platform [9] and Tamarin [26], [2] are very successful. However, even if some of these tools have been extended to support equivalences, they remain limited. Specifically, ProVerif [12] and Tamarin [10] can only prove very restricted forms of equivalences, called diff-equivalences, which are too limiting for unlinkability [21]. All these reasons explain why there are currently only few formal proofs of unlinkability. For instance, we may note the manual but very detailed proof of unlinkability for a variant of AKA by Koutsos [24] and the mechanised proofs of e-passport and RFID protocols by Hirschi *et al.* [21] using ProVerif. Both of these works have lead to the discovery of new attacks on protocols that were previously claimed unlinkable.

Given the current impossibility and certain difficulty of directly verifying unlinkability, [21] adopted an approach based on sufficient conditions. The authors identify two conditions that correspond to two broad classes of attacks on unlinkability, and show that these two conditions imply unlinkability. Moreover, they show that the conditions can be verified directly using *e.g.* ProVerif, which they successfully use on several case studies. Let us briefly describe these conditions. First, *frame opacity* states that, for every execution of the protocol against an active adversary, relations between messages never leak information about the involved agents. This property is expressed using a notion of message idealisation, and can be verified using (an extension of) ProVerif's diff-equivalence. Then, *well-authentication* states that, whenever the outcome of a conditional is positive, the corresponding agent is having an honest interaction with a counterpart of

The research leading to these results has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (grant agreement No 714955-POPSTAR), as well as from the French National Research Agency (ANR) under the project TECAP.

the expected identity. This ensures that the attacker cannot learn anything about identities by (indirectly) observing the outcome of conditionals. In practice, it is easy to verify well-authentication as it is a traditional correspondence property.

The main limitation of the work of [21] is that it does not cover *stateful* protocols, which is important for two reasons. First, many authentication protocols involve a database accessed by readers to check the credentials of an RFID device (aka. tag). In its simplest form, this global state is monotonic, *i.e.* database entries are never modified nor removed. Second, the secret shared by each tag with the readers is often updated to achieve forward privacy. Analysing protocols with such non-monotonic state is notoriously difficult. Whereas Tamarin has been designed to support stateful protocols from the beginning, there have been several attempts to add global states to ProVerif and other tools, e.g. StatVerif [8], Set-Pi [14], AIF-ω [27]. The most recent one, GSVerif [17], enriches ProVerif models to achieve better precision, which leads to several successes on trace properties.

Contributions and outline. In this paper, we revisit the work of [21] with a focus on protocols where the readers rely on a global database and each tag carries a local state. This setting is formally described in Section II. We show in Section III that existing notions of unlinkability are inadequate for our protocols, even when tags do not update their state, and propose a definition that precisely reflects the intended use case. Since the direct verification of unlinkability with available tools remains out of reach, we adapt the method of [21] in Section IV. In particular, we design a third condition to obtain sufficient conditions for our definition of unlinkability. We discuss in Section V how these three conditions can be mechanically verified using Tamarin [26], [2], and present case studies in Section VI.

Before entering into the formal details, let us briefly comment on our sufficient conditions. First, frame opacity is directly inherited from [21], and we also verify it as a diff-equivalence, although using Tamarin rather than ProVerif, with some manual guidance or pre-processing for most cases. Well-authentication is also inherited from [21], with a minor technical difference. However, it is important to observe that, in the stateless setting of [21], the converse of well-authentication was obvious: honest interactions lead to successful conditionals. The situation changes in a setting where the tags' states may evolve, possibly desynchronise with the readers' database, with an impact on the outcome of future honest interactions. This leads to our new condition, *no desynchronisation*, which requires that an honest interaction between a tag and a reader cannot fail, *i.e.* the outcome of its conditionals is always positive.

To illustrate how a desynchronisation can lead to an attack on unlinkability, let us consider a simple protocol between tags and readers. Each tag carries a state k_T, initialised with some k that is also stored as a new entry in the readers' database. The protocol consists of a single message from the tag to the reader, which is a hash of the tag's state, written $\mathsf{g}(k_T)$. When receiving an input, the reader checks that it matches $\mathsf{g}(k_R)$ for a k_R in its database. At the end of the session, tag and reader update their state by applying another hash function h to it: $k_T \leftarrow \mathsf{h}(k_T)$ and $k_R \leftarrow \mathsf{h}(k_R)$. This protocol is easily desynchronised: an attacker could intercept the first output of a tag, which nevertheless updates its state, rendering all future interactions with readers unsuccessful since no state in the database will match the updated state of the tag. Once a tag has been desynchronised, an honest session fails if, and only if, it involves that tag: this is a failure of unlinkability.

II. Model for protocols

We model security protocols in the symbolic model with a process algebra inspired from the applied pi calculus [5]. Participants are represented by processes while messages exchanged between participants are represented by terms.

We consider a specific class of protocols: stateful 2-party protocols between a tag and a reader. In order to model the stateful nature of these protocols, we assume that each tag has a memory cell to store the successive values of its state, whereas the readers have access to a global database.

A. Term algebra

We assume an infinite set $\mathcal{N}$ of *names* used to represent *atomic private data* such as keys, or nonces; and two infinite and disjoint sets of *variables* $\mathcal{X}$ and $\mathcal{W}$. Variables in $\mathcal{X}$ are used to refer *e.g.* to input messages and variables in $\mathcal{W}$, called *handles*, are used as pointers to messages learned by the attacker. We assume a *signature* Σ, *i.e.* a set of function symbols, split into *constructors* and *destructors*: $\Sigma = \Sigma_c \sqcup \Sigma_d$.

We note $\mathcal{T}(\mathcal{F}, \mathcal{D})$ the set of terms built from elements of the set of initial data $\mathcal{D}$ by applying function symbols in the signature $\mathcal{F}$. We refer to elements of $\mathcal{T}(\Sigma_c, \mathcal{N} \cup \mathcal{X})$ as *constructor terms*. We define $vars(t)$ as the set of variables that occur in a term $t \in \mathcal{T}(\Sigma, \mathcal{N} \cup \mathcal{X})$ and call *message* a constructor term u that is *ground*, *i.e.* such that $vars(u) = \varnothing$. We use the vector notation (for example $\overline{x}$) to denote a (possibly empty) sequence. The *domain* of a substitution σ is noted $\mathsf{dom}(\sigma)$, and its application to a term t is noted $t\sigma$. The *positions* of a term are defined as usual.

We split the signature Σ into two sets, Σ_{pub} and Σ_{priv}, to distinguish function symbols that are *public* (available to the attacker) from others that are *private*. A computation performed by the attacker, modelled by a term in $\mathcal{T}(\Sigma_{\mathsf{pub}}, \mathcal{W})$, is called a *recipe*. Our attacker cannot generate fresh names, but we may assume instead an infinite set of public constants available to the attacker in the set $\Sigma_c \cap \Sigma_{\mathsf{pub}}$.

Example 1: Let $\Sigma^{\mathsf{BH}} = \{\mathsf{h}, \langle\rangle, \mathsf{proj}_1, \mathsf{proj}_2, \mathsf{eq}, \mathsf{ok}, \mathsf{error}\}$. The function symbol h represents a hash function of arity 2. We model the pairing function with symbol $\langle\rangle$ of arity 2, and projections with $\mathsf{proj}_1/\mathsf{proj}_2$ of arity 1. The binary symbol eq is used to model equality tests. Finally, symbols ok and error of arity 0 model public constants. All these function symbols are public. Constructors are given by $\Sigma_c^{\mathsf{BH}} = \{\mathsf{h}, \langle\rangle, \mathsf{ok}, \mathsf{error}\}$, and destructors by $\Sigma_d^{\mathsf{BH}} = \{\mathsf{proj}_1, \mathsf{proj}_2, \mathsf{eq}\}$.

In order to provide a meaning to constructor symbols, we equip constructor terms with an equational theory. We assume a set of equations E over $\mathcal{T}(\Sigma_c, \mathcal{X})$ and define $=_{\mathsf{E}}$ as the smallest congruence containing E that is closed under substitutions and under bijective renaming.

Example 2: Consider $\Sigma^{\oplus} = \Sigma_c^{\oplus} = \{\oplus, 0\}$ where $\oplus$ is binary and 0 is a constant. The standard equational theory $\mathsf{E}_{\oplus}$ modelling the exclusive or operator is the following:

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

For instance, with $a, b \in \mathcal{N}$, we have that $a \oplus (a \oplus b) =_{\mathsf{E}_{\oplus}} b$.

Then, we give a meaning to destructors through an *ordered rewrite system*, *i.e.* an ordered set of rules $\mathsf{g}(t_1, \ldots, t_n) \to t$ where g is a destructor and $t, t_1, \ldots, t_n$ are constructor terms. A ground term u can be rewritten into v if there is a position p in u, a rewrite rule $\mathsf{g}(t_1, \ldots, t_n) \to t$ and a substitution θ from variables to messages such that $u|_p =_{\mathsf{E}} \mathsf{g}(t'_1, \ldots, t'_n)$, $t'_1 =_{\mathsf{E}} t_1\theta, \ldots, t'_n =_{\mathsf{E}} t_n\theta$ and $v =_{\mathsf{E}} u[t\theta]_p$ (*i.e.* u in which the subterm at position p has been replaced by $t\theta$). In the case where more than one rule may be applied at position p, only the first such rule can be effectively used. Given a ground term u, it may be possible to rewrite it (in an arbitrary number of steps) into a message: in that case, this message is noted $u{\Downarrow}$. We write $u{\Downarrow}$ when no such message exists, and say that the computation *fails*.

Example 3: Going back to Example 1, we consider the following rules to represent the properties of symbols in Σ_d^{BH}:

$$\mathsf{proj}_1(\langle x_1, x_2\rangle) \to x_1 \quad \mathsf{proj}_2(\langle x_1, x_2\rangle) \to x_2 \quad \mathsf{eq}(x, x) \to \mathsf{ok}$$

When u, v are messages, $\mathsf{eq}(u, v){\Downarrow}$ if, and only if, $u =_{\mathsf{E}} v$.

B. Process algebra

We consider a set $\mathcal{C}$ of *channel names* assumed to be public. We also consider an infinite set $\mathcal{R}$ of *references* to represent memory cell addresses of tags. We assume that $\mathcal{R}$ and $\mathcal{N}$ (the set of names defined in Section II-A) are disjoint.

Protocols will be modelled as *processes* given by the grammar in Figure 1. We will not comment on the standard constructs of this grammar (*i.e.* name restriction, input, output, parallel). The process $P; Q$ represents the sequential composition of processes P and Q and will only have a meaningful semantics under some conditions on P (this will be detailed later on). The *replication* $!P$ can be understood as the infinite parallel composition $P \mid (P \mid (P \mid \ldots))$ while the *repetition* $¡P$ corresponds to an infinite sequential composition $P; (P; (P; \ldots))$. The process $\mathsf{let}\ \overline{x} = \overline{t}\ \mathsf{in}\ P\ \mathsf{else}\ Q$ (where $\overline{x}$ and $\overline{t}$ are two sequences of the same length, respectively containing variables and terms) combines computations with a conditional: it attempts to evaluate $\overline{t}$, executes P with $\overline{x}$ bound to the resulting messages upon success, and Q otherwise[1]. Next, we have constructs for dealing with memory cells, used by tags: $\mathsf{new}\, r.P$ introduces a new cell with undefined contents before executing P, where get and set constructs may be used to respectively lookup or update the contents of r. Finally, two constructs model the database shared by all readers: insert to add a value to the database, and lookup for both looking up and updating the database. More specifically, $\mathsf{lookup}\ y\ \mathsf{such\ that}\ \overline{x} = \overline{t}, y' = t'\ \mathsf{in}\ P\ \mathsf{else}\ Q$ attempts to find some value v in the database such that $\overline{t}\{y \mapsto v\}$ and $t'\{y \mapsto v\}$ evaluate successfully to $\overline{w}$ and v' respectively; upon success it replaces v with v' in the database and executes P with $y, y', \overline{x}$ bound to $v, v', \overline{w}$; otherwise, it executes Q.

The constructs in and get (resp. let) bind variable x (resp. $\overline{x}$) in their first sub-process. The construct lookup binds y in $\overline{t}$, t' and $y, y', \overline{x}$ in its first sub-process. The free variables $fv(P)$ of a process P are defined accordingly. We also define $fn(P)$ as the set of names occurring in P not bound by a new construct, and similarly for free references $\mathit{fref}(P)$. Processes are identified modulo α-renaming for bound variables, names and references. A process P is closed when $fv(P) = \varnothing$.

Example 4: As a running example, we consider the Basic Hash protocol as described in [15]. Each tag stores a secret key that is never updated, and the readers have access to a database containing all these keys. The protocol is as follows:

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

where n_T is a fresh nonce and k is the secret key.

When receiving a message, the reader checks that it is a pair whose second element is a hash of one of the keys from the database, using the first element of the pair as hashing key. The protocol can be modelled in our syntax by:

$$\begin{aligned} P_{\mathsf{BH}} &= (!\, R) \mid (!\, \mathsf{new}\, k.\mathsf{new}\, r.\mathsf{set}(r, k).\mathsf{insert}(k).\ ¡\, \mathsf{new}\, n_T.T) \\ T &= \mathsf{get}(r, y).\mathsf{out}(c_T, \langle n_T, \mathsf{h}(y, n_T)\rangle) \\ R &= \mathsf{in}(c_T, z). \\ &\quad \mathsf{lookup}\ y\ \mathsf{such\ that}\ \overline{x} = \mathsf{eq}(\mathsf{proj}_2(z), \mathsf{h}(y, \mathsf{proj}_1(z)))), \\ &\qquad\qquad y' = y\ \mathsf{in}\ \mathsf{out}(c_R, \mathsf{ok}) \\ &\qquad\qquad\qquad \mathsf{else}\ \mathsf{out}(c_R, \mathsf{error}). \end{aligned}$$

C. Semantics

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

- $\mathcal{P}$ is a multiset of closed processes where null processes are implicitly removed;
- $\phi = \{w_1 \mapsto u_1, \ldots, w_n \mapsto u_n\}$ is a *frame* representing the knowledge of the adversary, *i.e.* a substitution where $w_1, \ldots, w_n$ are handles and $u_1, \ldots, u_n$ are messages;
- $\mathcal{S} = (\mathcal{S}_T, \mathcal{S}_R)$ is a *store* split into two parts:
  - $\mathcal{S}_T$ is a substitution mapping references to messages, representing the current contents of memory cells, such that $\mathit{fref}(\mathcal{P}) \subseteq \mathsf{dom}(\mathcal{S}_T)$;
  - $\mathcal{S}_R$ is a set of messages representing the global database shared by the readers.

We note $\mathsf{store}(K)$ the store of a configuration K. We sometimes use $\mathcal{P}$ as a configuration; it stands for $(\mathcal{P}; \varnothing; \varnothing)$.

The operational semantics of processes is given by labelled transitions $K \xrightarrow{\alpha} K'$, where K and K' are configurations and α is an action of the form $\mathsf{in}(c, R)$, $\mathsf{out}(c, w)$, τ, τ_{abort},

[1]The use of sequences $\overline{x}$ and $\overline{t}$ here is superfluous in terms of expressiveness but allows for a more convenient definition of our notion of well-authentication.

$P, Q := 0$	null process	
$\mid \mathsf{new}\, n.P$	name restriction	$n \in \mathcal{N}$
$\mid \mathsf{new}\, r.P$	reference restriction	$r \in \mathcal{R}$
$\mid \mathsf{in}(c, x).P$	input	$c \in \mathcal{C}$, $x \in \mathcal{X}$
$\mid \mathsf{out}(c, u).P$	output	$c \in \mathcal{C}$, $u \in \mathcal{T}(\Sigma_c, \mathcal{N} \cup \mathcal{X})$
$\mid \mathsf{let}\ \overline{x} = \overline{t}\ \mathsf{in}\ P\ \mathsf{else}\ Q$	evaluation	$\overline{x} \in \mathcal{X}^k$, $\overline{t} \in \mathcal{T}(\Sigma, \mathcal{N} \cup \mathcal{X})^k$
$\mid \mathsf{get}(r, x).P$	memory cell lookup	$r \in \mathcal{R}$, $x \in \mathcal{X}$
$\mid \mathsf{set}(r, v).P$	memory cell insertion	$r \in \mathcal{R}$, $v \in \mathcal{T}(\Sigma_c, \mathcal{N} \cup \mathcal{X})$
$\mid \mathsf{lookup}\ y\ \mathsf{such\ that}\ \overline{x} = \overline{t}, y' = t'\ \mathsf{in}\ P\ \mathsf{else}\ Q$	database test and update	$(y, y', \overline{x}) \in \mathcal{X}^{k+2}$, and $(t', \overline{t}) \in \mathcal{T}(\Sigma, \mathcal{N} \cup \mathcal{X})^{k+1}$
$\mid \mathsf{insert}(v).P$	database insertion	$v \in \mathcal{T}(\Sigma_c, \mathcal{N} \cup \mathcal{X})$
$\mid (P \mid Q)$	parallel composition	
$\mid\ !P$	replication	
$\mid\ P; Q$	sequence	
$\mid\ ¡P$	repetition	

Fig. 1. Syntax of processes

τ_{else} or τ_{then}. The definition of these transitions is given in Figure 2. In that figure we refer to the free names of frames and stores, which are defined as expected: $fn(\phi)$ (resp. $fn(\mathcal{S})$) is the set of names appearing in the messages of ϕ (resp. $\mathcal{S}$).

Rule IN allows the attacker to send on channel c a message u, provided that it is the result of a computation, modelled by R, that applies public function symbols to messages from the frame, which models the attacker's knowledge. Rule OUT corresponds to the output of a term, that is added to the frame (and thus to the attacker's knowledge). Rules NEW-N and NEW-R are standard rules for restrictions; note that alpha-renaming can always be applied to obtain the associated freshness conditions. Rules LET-THEN and LET-ELSE correspond to the evaluation of a sequence of terms $\overline{t}$. In case of success, *i.e.* if there exists some sequence of messages $\overline{u}$ such that $\overline{u} = \overline{t}\Downarrow$, then variables $\overline{x}$ are bound to those messages and the process P is executed. In case of failure, the process Q is executed.

Rules INSERT, UPDATE-THEN and UPDATE-ELSE define transitions for operations on the store. By definition of configurations, $fref(\mathcal{P}) \subseteq \mathsf{dom}(\mathcal{S}_T)$, hence get and set can only be performed on a memory cell r after it has been created with the rule NEW-R. Note that we require v to be a message in rule SET whereas, in the syntax used to model protocols, we only require that terms occurring at these positions are (not necessarily ground) constructor terms. This difference is due to the fact that we consider closed processes in the operational semantics: by the time a $\mathsf{set}(r, v)$ is executed, all variables of v will have been bound to messages, turning the constructor term into a message itself.

We finally describe rules dealing with sequential composition and repetition. As in [21], we make use in rule SEQ of a simplification relation $\rightsquigarrow$ to move sequential compositions above most other constructs, so as to be able to execute them. This simplification relation is defined in fig. 3. It ensures that, whenever P is not a parallel composition, replication or repetition, we have $P; Q \rightsquigarrow R$ for some R that is not a sequence. Thus sequences and repetitions are only meaningful for this class of processes, which is limited but sufficient for our purposes. We also consider a rule ABORT to model the possibility for the attacker to interrupt a tag session at any time to start a new one.

Example 5: Following Example 4 we have:

$$(P_{\mathsf{BH}}; \varnothing; \varnothing) \xrightarrow{\mathsf{tr}} (\mathcal{P}_{\mathsf{BH}} \cup \{¡\,\mathsf{new}\, n_T.T\{r \mapsto r'\}\}; \phi_0; \mathcal{S}_0)$$

with $\mathcal{P}_{\mathsf{BH}} = \{!\,R, !\,\mathsf{new}\, k.\mathsf{new}\, r.\mathsf{set}(r, k).\mathsf{insert}(k).\ ¡\,\mathsf{new}\, n_T.T\}$, and for arbitrary names $k' \neq n'_T$, handles $w_0 \neq w_1$ and reference r':

- $\mathsf{tr} = \tau^9.\mathsf{out}(c_T, w_0).\mathsf{in}(c_T, w_0).\tau_{\mathsf{then}}.\mathsf{out}(c_R, w_1)$;
- $\phi_0 = \{w_0 \mapsto \langle n'_T, \mathsf{h}(k', n'_T)\rangle, w_1 \mapsto \mathsf{ok}\}$;
- $\mathcal{S}_0 = (\{r' \mapsto k'\}, \{k'\})$.

D. Trace equivalence

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

Definition 1: A frame ϕ is *statically included* in ϕ' when $\mathsf{dom}(\phi) = \mathsf{dom}(\phi')$ and:

- for any recipe R such that $R\phi\Downarrow$ is a message, we have that $R\phi'\Downarrow$ is a message;
- for any recipes R_1, R_2 such that $R_1\phi\Downarrow$, $R_2\phi\Downarrow$ are messages, and $R_1\phi\Downarrow =_{\mathsf{E}} R_2\phi\Downarrow$, we have $R_1\phi'\Downarrow =_{\mathsf{E}} R_2\phi'\Downarrow$.

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

Example 6: As an illustrative example, we consider the signature given in Example 2 and the following frames:

- $\phi_{\mathsf{diff}} = \{w_1 \mapsto id_1 \oplus n_1, w_2 \mapsto id_2 \oplus n_2\}$, and
- $\phi_{\mathsf{same}} = \{w_1 \mapsto id_0 \oplus n_1, w_2 \mapsto id_0 \oplus n_2\}$

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

$$\textsc{Out}\quad (\mathsf{out}(c,u).P \cup \mathcal{P};\phi;\mathcal{S}) \xrightarrow{\mathsf{out}(c,w)} (P \cup \mathcal{P};\phi \cup \{w \mapsto u\};\mathcal{S})$$
with u a message and w a fresh handle from $\mathcal{W}$

$$\textsc{New-N}\quad (\mathsf{new}\, n.P \cup \mathcal{P};\phi;\mathcal{S}) \xrightarrow{\tau} (P \cup \mathcal{P};\phi;\mathcal{S}) \quad \text{where w.l.o.g. } n \notin fn(\mathcal{P},\phi,\mathcal{S})$$

$$\textsc{New-R}\quad (\mathsf{new}\, r.P \cup \mathcal{P};\phi;(\mathcal{S}_T,\mathcal{S}_R)) \xrightarrow{\tau} (P \cup \mathcal{P};\phi;(\mathcal{S}_T \cup \{r \mapsto \bot\},\mathcal{S}_R)) \text{ where w.l.o.g. } r \notin \mathsf{dom}(\mathcal{S}_T)$$

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

$$\textsc{Let-Else}\quad (\mathsf{let}\ \overline{x} = \overline{t}\ \mathsf{in}\ P\ \mathsf{else}\ Q \cup \mathcal{P};\phi;\mathcal{S}) \xrightarrow{\tau_{\mathsf{else}}} (Q \cup \mathcal{P};\phi;\mathcal{S}) \quad \text{when } \overline{t}\Downarrow$$

$$\textsc{Get}\quad (\mathsf{get}(r,y).P \cup \mathcal{P};\phi;(\mathcal{S}_T,\mathcal{S}_R)) \xrightarrow{\tau} (P\{y \mapsto \mathcal{S}_T(r)\} \cup \mathcal{P};\phi;(\mathcal{S}_T,\mathcal{S}_R))$$

$$\textsc{Set}\quad (\mathsf{set}(r,v).P \cup \mathcal{P};\phi;(\mathcal{S}_T,\mathcal{S}_R)) \xrightarrow{\tau} (P \cup \mathcal{P};\phi;(\mathcal{S}_T[r \leftarrow v],\mathcal{S}_R)) \quad \text{with } v \text{ a message}$$

$$\textsc{Insert}\quad (\mathsf{insert}(v).P \cup \mathcal{P};\phi;(\mathcal{S}_T,\mathcal{S}_R)) \xrightarrow{\tau} (P \cup \mathcal{P};\phi;(\mathcal{S}_T,\mathcal{S}_R \cup \{v\})) \quad \text{with } v \text{ a message}$$

$$\textsc{Update-Then}\quad (\mathsf{lookup}\ y\ \mathsf{such\ that}\ \overline{x} = \overline{t}, y' = t'\ \mathsf{in}\ P\ \mathsf{else}\ Q \cup \mathcal{P};\phi;(\mathcal{S}_T,\mathcal{S}_R)) \xrightarrow{\tau_{\mathsf{then}}} (P\{y \mapsto v, y' \mapsto v', \overline{x} \mapsto \overline{w}\} \cup \mathcal{P};\phi;(\mathcal{S}_T,(\mathcal{S}_R \setminus \{v\}) \cup \{v'\}))$$
when $v \in \mathcal{S}_R$ is such that $\overline{w} = \overline{t}\{y \mapsto v\}\Downarrow$ and $v' = t'\{y \mapsto v\}\Downarrow$

$$\textsc{Update-Else}\quad (\mathsf{lookup}\ y\ \mathsf{such\ that}\ \overline{x} = \overline{t}, y' = t'\ \mathsf{in}\ P\ \mathsf{else}\ Q \cup \mathcal{P};\phi;\mathcal{S}) \xrightarrow{\tau_{\mathsf{else}}} (Q \cup \mathcal{P};\phi;\mathcal{S})$$
when for all $v \in \mathcal{S}_R$, we have that $\overline{t}\{y \mapsto v\}\Downarrow$ or $t'\{y \mapsto v\}\Downarrow$

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

$$\textsc{Replication}\quad (!P \cup \mathcal{P};\phi;\mathcal{S}) \xrightarrow{\tau} (P \cup\, !P \cup \mathcal{P};\phi;\mathcal{S})$$

$$\textsc{Repetition}\quad (\mathsf{i}P \cup \mathcal{P};\phi;\mathcal{S}) \xrightarrow{\tau} (\{P;\mathsf{i}P\} \cup \mathcal{P};\phi;\mathcal{S})$$

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

$$\textsc{Abort}\quad ((P;Q) \cup \mathcal{P};\phi;\mathcal{S}) \xrightarrow{\tau_{\mathsf{abort}}} (Q \cup \mathcal{P};\phi;\mathcal{S})$$

Fig. 2. Semantics for processes

$$\begin{array}{rcll}
0;Q & \rightsquigarrow & Q & \\
(\mathsf{in}(c,x).P);Q & \rightsquigarrow & \mathsf{in}(c,x).(P;Q) & \text{when } x \notin fv(Q) \\
(\mathsf{out}(c,u).P);Q & \rightsquigarrow & \mathsf{out}(c,u).(P;Q) & \\
(\mathsf{new}\, n.P);Q & \rightsquigarrow & \mathsf{new}\, n.(P;Q) & \text{when } n \notin fn(Q) \\
(\mathsf{new}\, r.P);Q & \rightsquigarrow & \mathsf{new}\, r.(P;Q) & \text{when } r \notin fref(Q) \\
(\mathsf{let}\ \overline{x} = \overline{t}\ \mathsf{in}\ P'\ \mathsf{else}\ P'');Q & \rightsquigarrow & \mathsf{let}\ \overline{x} = \overline{t}\ \mathsf{in}\ (P';Q)\ \mathsf{else}\ (P'';Q) & \text{when } \overline{x} \cap fv(Q) = \varnothing \\
(\mathsf{lookup}\ y\ \mathsf{such\ that}\ y', \overline{x} = \overline{t}\ \mathsf{in}\ P'\ \mathsf{else}\ P'');Q & \rightsquigarrow & \mathsf{lookup}\ y\ \mathsf{such\ that}\ y', \overline{x} = \overline{t}\ \mathsf{in}\ (P';Q)\ \mathsf{else}\ (P'';Q) & \\
 & & & \text{when } \{y, y', \overline{x}\} \cap fv(Q) = \varnothing \\
(\mathsf{insert}(v).P);Q & \rightsquigarrow & \mathsf{insert}(v).(P;Q) & \\
(\mathsf{get}(r,y).P);Q & \rightsquigarrow & \mathsf{get}(r,y).(P;Q) & \text{when } y \notin fv(Q) \\
(\mathsf{set}(r,v).P);Q & \rightsquigarrow & \mathsf{set}(r,v).(P;Q) &
\end{array}$$

Fig. 3. Sequence simplification rules

where $n_1, n_2, id_0, id_1, id_2 \in \mathcal{N}$. We have that $\phi_{\mathsf{diff}} \sim \phi_{\mathsf{same}}$. Assuming now that n_1 and n_2 are given to the attacker through $\phi_0 = \{w_3 \mapsto n_1, w_4 \mapsto n_2\}$, we have that: $\phi_0 \cup \phi_{\mathsf{same}} \not\sim \phi_0 \cup \phi_{\mathsf{diff}}$. Indeed, consider $R_1 = w_1 \oplus w_3$ and $R_2 = w_2 \oplus w_4$. The test $R_1 \stackrel{?}{=} R_2$ holds in $\phi_0 \cup \phi_{\mathsf{same}}$ whereas it does not hold in the frame $\phi_0 \cup \phi_{\mathsf{diff}}$. Knowing n_1 and n_2, the attacker is able to tell whether w_1 and w_2 are issued from the same tag.

In order to define trace equivalence, we first define $\mathsf{trace}(K)$ for a configuration $K = (\mathcal{P};\phi;\mathcal{S})$:

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

Given a trace tr, $\mathsf{obs}(\mathsf{tr})$ is the subsequence of tr obtained by erasing *unobservable* actions, *i.e.* τ, τ_{abort}, τ_{then} and τ_{else}.

Definition 2: 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 $K' \sqsubseteq K$.

E. Our class of protocols

In this section, we formally define the class of protocols we consider in this paper, *i.e.* stateful 2-party protocols where each role is a sequence of actions, without any parallel composition.

We consider a set $\mathcal{L}$ of labels (denoted by ℓ) used to decorate output actions, in order to identify which outputs come from a same syntactic action. We fix two channels $c_T \neq c_R$ in order to identify the role underlying a given action.

Definition 3: A *tag role* manipulating reference r_0 is a closed process T obtained using the following grammar and such that $\mathit{fref}(T) \subseteq \{r_0\}$:

$$\begin{array}{rcl} T, T_1, T_2 & ::= & 0 \mid \mathsf{in}(c_R, x).T \mid \ell : \mathsf{out}(c_T, u).T \\ & & \mid \mathsf{let}\ \overline{x} = \overline{t}\ \mathsf{in}\ T_1\ \mathsf{else}\ T_2 \\ & & \mid \mathsf{get}(r_0, y).T \mid \mathsf{set}(r_0, v).T \end{array}$$

A *reader role* is a closed process R obtained from the grammar below:

$$\begin{array}{rcl} R, R_1, R_2 & ::= & 0 \mid \mathsf{in}(c_T, x).R \mid \ell : \mathsf{out}(c_R, u).R \\ & & \mid \mathsf{let}\ \overline{x} = \overline{t}\ \mathsf{in}\ R_1\ \mathsf{else}\ R_2 \\ & & \mid \mathsf{lookup}\ y\ \mathsf{such\ that}\ \overline{x} = \overline{t}, y' = t'\ \mathsf{in}\ R_1 \\ & & \quad \mathsf{else}\ R_2 \end{array}$$

A tag role is allowed to access its memory cell r_0, whereas a reader role manipulates the database through the lookup construct. A protocol is then given by tag and reader roles, together with some data to initialise their memory.

Definition 4: A protocol Π is a tuple $(\mathit{init}_T, \mathit{init}_R, T, R)$ where init_T, init_R are messages, T is a tag role manipulating reference r, and R is a reader role. We assume that no output label occurs twice in T and R. We define $\mathcal{M}_\Pi$ as

$$\begin{array}{ll} & (!\ \mathsf{new}\,\overline{k}.\mathsf{new}\,r.\mathsf{set}(r, \mathit{init}_T).\mathsf{insert}(\mathit{init}_R).\ \mathsf{i}\ \mathsf{new}\,\overline{n}_T.T) \\ \mid & (!\ \mathsf{new}\,\overline{n}_R.R) \end{array}$$

where $\overline{k} = \mathit{fn}(\mathit{init}_T, \mathit{init}_R)$, $\overline{n}_T = \mathit{fn}(T)$, and $\overline{n}_R = \mathit{fn}(R)$.

The *session parameters* $\overline{n}_T$ (resp. $\overline{n}_R$) occurring in T (resp. R) correspond to secrets that are known only by the tag (resp. reader) and are generated fresh for each session. This is without loss of generality since arbitrary secrets can be shared via the *initialisation parameters* init_T and init_R. The process $\mathcal{M}_\Pi$ corresponds to a real system using protocol Π, where an arbitrary number of tags can be spawned. For each new tag, fresh copies of r and $\overline{k}$ are generated to initialise the reference and an entry is inserted in the database. Note that we consider sequential sessions for a given tag in order to avoid concurrent accesses to memory cells.

Example 7: Continuing Example 4, T and R correspond respectively to tag and reader roles, up to the additon of distinct labels ℓ_0, ℓ_1 and ℓ_2 to decorate their outputs. Then $\Pi_{\mathsf{BH}} = (k, k, T, R)$ is a protocol according to our definition and the process P_{BH} of Example 4 is $\mathcal{M}_{\Pi_{\mathsf{BH}}}$.

III. Modelling unlinkability

We first recall some earlier definitions of unlinkability before giving ours and discussing how and why our definition differs from earlier ones.

Intuitively, unlinkability attempts to express that an attacker cannot learn *anything* about the relationships between several uses of the protocols, in the spirit of the ISO definition [22].

A. Related work

a) Weak and strong unlinkability: A formal notion of *weak unlinkability* has been proposed in [6] as a very general model of unlinkability in the framework of the applied pi-calculus. It is similar in spirit to the *untraceability* notion of [32]. Intuitively, it requires that if two actions are played by the same tag in a trace, there exists an indistinguishable trace where the two actions are played by different tags.

In [6], another notion is introduced, *strong unlinkability*, which the authors view as being unrealistically strong but more amenable to mechanised verification. Importantly, strong unlinkability is defined in terms of *observational equivalence*, a very strong equivalence on processes whose failure often does not correspond to attacks.

b) Taking into account the reader in the model: Brusò et al. [15] investigate unlinkability for a simple class of protocols. In their model, the reader process is explicitly set to 0, which is possible because tag roles consist of a single output. Even in such a simple situation, this is abusive: we will illustrate this on the OSK protocol [29] in Example 9. This protocol is unlinkable according to [15] but we show a legitimate linkability attack that breaks our notion of unlinkability.

c) Different ways of modelling the reader: A more recent definition of unlinkability, introduced in [21], involves identity-specific readers, *i.e.* each reader session can only interact successfully with sessions of a specific tag identity. This can be realistic for some protocols, *e.g.* e-passport protocols. However, for the protocols considered here, where readers access a global database, considering a generic reader role makes more sense. In fact, we will show that it avoids false attacks in Example 10.

B. Our definition of unlikability

On top of this historical buildup, we now propose a definition of unlinkability that is relevant in our context, before discussing it in light of previous definitions.

We define unlinkability as the impossibility for an outside observer to distinguish between two systems: the real system where each tag can play an arbitrary number of sessions for each identity, and an ideal system where each tag can play only one session for each identity. We express this indistinguishability in terms of trace equivalence.

Definition 5: Let $\Pi = (\mathit{init}_T, \mathit{init}_R, T, R)$ be a protocol. We define $\mathcal{S}_\Pi$ as the process obtained from $\mathcal{M}_\Pi$ by removing the repetition operator i. We say that Π is *unlinkable* if $\mathcal{M}_\Pi \approx \mathcal{S}_\Pi$.

Note that, even in the single session system $\mathcal{S}_\Pi$, multiple reader sessions may access the database entry corresponding to a same identity.

Example 8: Continuing Example 7, unlinkability is expressed through the following equivalence:

$$\left(!\,\mathsf{new}\,k.\mathsf{new}\,r.\mathsf{set}(r,k).\mathsf{insert}(k).\,\mathsf{i}\,\mathsf{new}\,n_T.T\right)\mid !R$$
$$\approx \left(!\,\mathsf{new}\,k.\mathsf{new}\,r.\mathsf{set}(r,k).\mathsf{insert}(k).\mathsf{new}\,n_T.T\right)\mid !R$$

Showing that such an equivalence holds is non trivial and is actually the main purpose of this work.

C. Comparison with previous definitions

a) Weak and strong unlinkability: It can be shown that the *weak unlinkability* of [6] is strictly weaker than our notion of unlinkability, and misses potential attacks; see Appendix A-A for details. In fact, our definition of unlinkability is closer to the notion of *strong unlinkability* introduced in [6], which the authors view as being unrealistically strong but more amenable to mechanised verification.

In contrast, our definition relies on the more realistic notion of trace equivalence, as is now common [21], [18]. Unlike [6], we view our definition of unlinkability as being realistic (it precisely captures linkability attacks) but not amenable to verification (there are no tools for proving trace equivalences with unbounded sessions).

b) Taking into account the reader in the model: We illustrate with the OSK protocol [29] that our definition of unlinkability is able to capture a legitimate linkability attack that would be missed by the notion defined in [15].

Example 9 (OSK protocol): Let $\Pi = (k, k, T, R)$ where:

$$\begin{array}{rcl} T & = & \mathsf{get}(r,y).\mathsf{set}(r,\mathsf{hash}(y)).\ell_0 : \mathsf{out}(c_T,\mathsf{g}(y)) \\ R & = & \mathsf{in}(c_T,x).\mathsf{lookup}\ y\ \mathsf{such\ that} \\ & & \quad x_{\mathsf{test}} = \mathsf{TestOSK}(y,x), y' = \mathsf{UpdateOSK}(x)\ \mathsf{in} \\ & & \quad \ell_1 : \mathsf{out}(c_R,\mathsf{ok})\ \mathsf{else}\ \ell_2 : \mathsf{out}(c_R,\mathsf{error}) \end{array}$$

with the following infinite set of rewriting rules:

- $\mathsf{TestOSK}(x, \mathsf{g}(\mathsf{hash}^n(x))) \to \mathsf{ok}$ with $n \geq 0$; and
- $\mathsf{UpdateOSK}(\mathsf{g}(y)) \to \mathsf{hash}(y)$.

Each tag has a secret state, whose initial value is stored in the reader's database. The tag's state is updated by applying a hash function hash at each session. The reader expects a message of the form $\mathsf{g}(\mathsf{hash}^n(x))$ for some database entry (to allow resynchronisations) and updates that entry with $\mathsf{hash}^{n+1}(x)$ (to avoid replay attacks). This step is modelled relying on the private function symbol UpdateOSK. We explicitly model the reader's response using public constants ok and error: this is important because in actual access control scenarios, one often observes the outcome of the authentication protocol, *e.g.* by observing whether a door opens.

The scenario described in Figure 4 is a linkability attack w.r.t. our definition. The first tag's output is intercepted by the attacker in order to be replayed after a successful interaction between the tag and the reader. This replayed message is not accepted by the reader because the state contained in the message is too old. This scenario shows that it is possible to link two sessions of a same tag: a reader replies ok after receiving a previously intercepted message if and only if this tag has not interacted with the reader meanwhile.

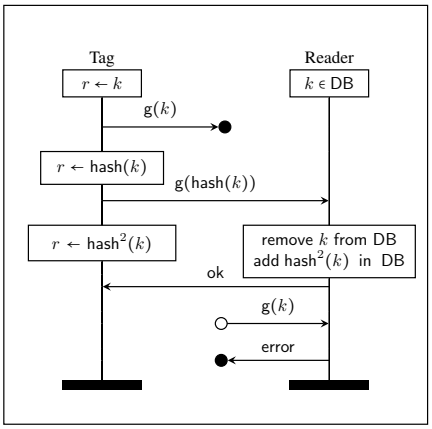

Fig. 4. Linkability attack for OSK protocol

c) Different ways of modelling the reader: The next example protocol is linkable in the setting of [21] but unlinkable according to our definition.

Example 10 (Basic Hash protocol): We go back to our running example. Since the state is never updated, it is possible to analyse the protocol in the setting of [21]. In that setting, readers are identity-specific, and the scenario described in Figure 5 is a linkability attack.

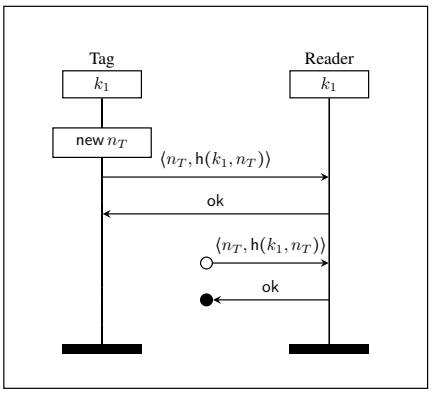

Fig. 5. Scenario with identity-specific readers of the Basic Hash protocol

In this scenario, the attacker replays a message and observes the answer from the reader. That answer may be ok in the multiple sessions system, where two readers can interact with tags of identity k_1, but not in the single session system. This scenario does not correspond to an attack when considering generic readers that would perform a database lookup to check the tag's output. Our notion of unlinkability closely models this behaviour and does not suffer from this false attack.

Therefore, for protocols that can be modelled in the setting of [21] (*i.e.* protocols without updates) our notion of unlinkability does not imply the one of [21]. The converse implication, however, holds; see Appendix A-B for details. Our notion is thus strictly weaker and, in our opinion, more realistic for the class of protocols that we consider.

IV. A METHOD FOR ESTABLISHING UNLINKABILITY

As already mentionned, existing tools for verifying unlinkability in the unbounded case (ProVerif, Tamarin) are based on a notion of *diff-equivalence* and are unable to conclude even for very simple protocols. We illustrate this systematic issue with our running example, the Basic Hash protocol. We have tried to check unlinkability on this example using ProVerif and Tamarin and in both cases, the tool has returned a trace which does not correspond to an attack.

In ProVerif, we model the database with a table and since the tag performs only one action (an output), we can consider tags in parallel. Then, in order to check the equivalence given in Example 8, we write the following bi-process:

```
table keys(bitstring).

let T (k:bitstring,nT:bitstring) =
  out(cT,(nT,h((k,nT)))).

let R =
  in(cT, x:bitstring);
  get keys(k) suchthat snd(x)=h((k,fst(x))) in
  out(cR,ok).

process
  (! R) |
  ! new k:bitstring;
  ! new kk:bitstring; insert keys(choice[k,kk]);
    new nT:bitstring; T(choice[k,kk],nT)
```

On the left, an arbitrary number of tags with identity k can each play many sessions. On the right, each new session of T is launched with a fresh identity kk, *i.e.* each tag plays a single session. The tool concludes that the equivalence "cannot be proved" and shows an attack trace. This trace involves a reader R and two tags $T(\mathsf{choice}[k, kk])$ and $T(\mathsf{choice}[k, kk'])$. Assuming that the first tag performs its output, the bi-frame will contain:

$$w_1 \to \langle n_T, \mathsf{h}(\mathsf{choice}[k, kk], n_T)\rangle.$$

Then, the reader performs an input with w_1 and looks up in the table to find a key matching the received message. On the left side of the bi-process, the table contains the keys k at line 1 and k at line 2. On the right side of the bi-process, the table contains the keys kk at line 1 and kk' at line 2. Therefore, the test will succeed on the left side independently of the chosen line, while the test fails on the right side when the reader looks up at line 2. This false attack relies on the incorrect assumption that the attacker is able to identify the line of the database that is involved in the underlying execution.

The experiments we conducted in Tamarin lead to a similar conclusion; more details may be found in our source code [3].

We now revisit the work of [21] in the context of stateful protocols. We define next our three conditions: *frame opacity*, *well-authentication* and *no desynchronisation*. In order to do so, we first have to introduce annotations in our semantics.

A. Annotations

We distinguish two types of actions in protocol executions:

1) τ actions are used to create new agents (rules NEW-N, NEW-R, PAR, REPLICATION, REPETITION, SEQ, INSERT and SET for the initialisation) and will not be annotated;
2) other actions are performed by already created agents (rules IN, OUT, LET-THEN, LET-ELSE, UPDATE-THEN, UPDATE-ELSE, ABORT, GET and SET when it corresponds to an action from the tag) and will be annotated accordingly.

Considering a protocol Π, we define a set $\mathcal{A}$ of annotations for identitying the protocol's agents. A tag is annotated $T(r, \overline{n}_T)$ where r is the reference and $\overline{n}_T$ the session parameters for this tag, and a reader is annotated $R(\overline{n}_R)$ where $\overline{n}_R$ are the session parameters for this reader. The goal of those annotations is to explicitly associate each action to the agent responsible for it. We assume that $\overline{n}_T \neq \varnothing$ and $\overline{n}_R \neq \varnothing$ in order to uniquely identify agents. Note that these session nonces do not have to be useful in the role, so the assumption is not restrictive.

We are now able to define an *annotated semantics*, where agents are annotated as explained above and actions coming from the tag/reader roles are annotated with the annotation of the agent performing this action. We will write ta to refer to traces of the annotated semantics.

Example 11: We go back to our running example, considering the protocol of Example 5 with a session parameter n_R added for the reader role. The multiset $\mathcal{P}_{\mathsf{BH}}$ thus becomes:

$$\{!\,\mathsf{new}\, n_R.R,\ !\,\mathsf{new}\, k.\mathsf{new}\, r.\mathsf{set}(r,k).\mathsf{insert}(k).\ \mathsf{i}\,\mathsf{new}\, n_T.T\}.$$

The annotated version of the execution given in Example 5 yields the following annotated trace ta, where k', n'_T and n'_R are fresh names, $a_T = T(r', n'_T)$ and $a_R = R(n'_R)$:

$$\begin{aligned}\mathsf{ta} \;=\;& \tau^{10}.\ell_0 : \mathsf{out}(c_T, w_0)[a_T].\\ & \mathsf{in}(c_T, w_0)[a_R].\tau_{\mathsf{then}}[a_R].\ell_1 : \mathsf{out}(c_R, w_1)[a_R]\end{aligned}$$

After the first τ actions, the annotated configuration is $(\mathcal{P}_{\mathsf{BH}} \cup \{R\sigma_R[a_R], T\sigma_T[a_T], \mathsf{i}\,\mathsf{new}\, n_T.T\{r \mapsto r'\}\}; \varnothing; \mathcal{S}_0)$ where:

- $\sigma_T = \{r \mapsto r', n_T \mapsto n'_T\}$; $\sigma_R = \{n_R \mapsto n'_R\}$;
- $\mathcal{S}_0 = (\{r' \mapsto k'\}, \{k'\})$.

After ta, the configuration is the same as in Example 5: no annotation remains since all agents have terminated.

B. Frame opacity

Intuitively, this condition aims to prevent attacks in which, for some possible behaviour of the attacker, there exists a relation between messages that leaks information about the involved agents. Practically speaking, this condition requires that any reachable frame must be statically equivalent to an idealised frame that does not depend on identity parameters.

A very simple way to obtain an idealisation of a frame is to replace each output message by a fresh nonce. In that case, if the real frame and the idealised frame are statically equivalent, it is obvious that the attacker cannot learn anything by analysing the relations between the messages, since there is no relation between disctinct fresh nonces. Nevertheless, it is not satisfying because too restrictive as *e.g.* a pair is distinguishable from a nonce.

We will obtain idealised frames by replacing each output message by a message built using constructors and fresh nonces. The precise construction will depend on the specific output, identified by its label ℓ. Formally, we consider a set of *name variables* $\mathcal{X}^{\mathrm{n}} = \{x_1^{\mathrm{n}}, x_2^{\mathrm{n}}, \ldots\} \subseteq \mathcal{X}$ that will refer to fresh names used in the idealised frame. We also assume a fixed but arbitrary *idealisation operator* $\mathrm{ideal}(\cdot) : \mathcal{L} \mapsto \mathcal{T}(\Sigma_c, \mathcal{X}^{\mathrm{n}})$. This idealisation operator will have to be chosen as part of the modelling such that frame opacity holds.

Definition 6: Let $\mathrm{fr} : \mathcal{A} \times \mathcal{X}^{\mathrm{n}} \mapsto \mathcal{N}$ be an injective function mapping, for each agent, name variables to names. We define the idealised frame $\Phi^{\mathrm{fr}}_{\mathrm{ideal}}(\mathsf{ta})$ associated to ta, as follows:

$$\{w \mapsto \mathrm{ideal}(\ell)\sigma^{\mathrm{n}}\} \in \Phi^{\mathrm{fr}}_{\mathrm{ideal}}(\mathsf{ta}) \;\Leftrightarrow\; \ell : \mathsf{out}(c, w)[a] \in \mathsf{ta}$$

where $\sigma^{\mathrm{n}}(x_j^{\mathrm{n}}) = \mathrm{fr}(a, x_j^{\mathrm{n}})$.

Example 12: Continuing Example 11, we consider the following idealisation operator:

$$\ell_0 \mapsto \langle x_1^{\mathrm{n}}, x_2^{\mathrm{n}} \rangle,\ \ell_1 \mapsto \mathsf{ok},\ \ell_2 \mapsto \mathsf{error}.$$

Let fr be an injective function such that $\mathrm{fr}(a_T, x_i^{\mathrm{n}}) = n_i^T$ with $i \in \{1, 2\}$. We have $\Phi^{\mathrm{fr}}_{\mathrm{ideal}}(\mathsf{ta}) = \{w_0 \mapsto \langle n_1^T, n_2^T \rangle, w_1 \mapsto \mathsf{ok}\}$.

The choice of fr in $\Phi^{\mathrm{fr}}_{\mathrm{ideal}}(\mathsf{ta})$ is not important with respect to static equivalence: we have $\Phi^{\mathrm{fr}_1}_{\mathrm{ideal}}(\mathsf{ta}) \sim \Phi^{\mathrm{fr}_2}_{\mathrm{ideal}}(\mathsf{ta})$ for any fr_1 and fr_2 [21]. We can thus often write $\Phi_{\mathrm{ideal}}(\mathsf{ta})$ instead of $\Phi^{\mathrm{fr}}_{\mathrm{ideal}}(\mathsf{ta})$, as in the definition of frame opacity given next.

Definition 7: The protocol Π ensures *frame opacity* w.r.t. the idealisation operator ideal if for any execution $(\mathcal{M}_\Pi; \varnothing; \mathcal{S}) \xrightarrow{\mathsf{ta}} (Q; \phi; \mathcal{S}')$ we have that $\Phi_{\mathrm{ideal}}(\mathsf{ta}) \sim \phi$.

Example 13: With the idealisation operator given in Example 12, frame opacity holds for the Basic Hash protocol. This can be established using the diff-equivalence of either ProVerif or Tamarin. The intuition is that messages outputted by tags are all different since a fresh nonce is used each time, and an outside observer cannot distinguish a hash from a nonce. More practical details will be given in Section VI.

As in [21], we can actually consider more complex and powerful idealisation operators (see also Appendix B-C). However, they are not needed to conclude on our case studies presented in Section VI and we therefore chose to simplify this notion for the sake of readability.

C. Well-authentication

The idea behind this second condition is to avoid that the outcome of conditionals leaks information about identities to the attacker. To do so, we require that whenever a conditional (let or lookup) is positively evaluated, the corresponding agent is having an *honest* interaction with another participant. In practice, protocols often have some conditionals for which the attacker already knows the outcome: these *safe* conditionals can (and must) be excluded from our condition.

Definition 8: A conditional occurring in a role (tag or reader) is said to be *safe* if it is of the form $\mathsf{let}\ \overline{x} = \overline{t}\ \mathsf{in}\ P\ \mathsf{else}\ Q$ with $\overline{t}$ a sequence of $\mathcal{T}(\Sigma_{\mathsf{pub}}, \{x_1, \ldots, x_n\} \cup \{u_1, \ldots, u_m\})^*$, where the x_i are the variables bound by the inputs preceding the conditional in the role, and the u_i are the messages used in the previous outputs of that role.

In particular, a lookup conditional is never safe.

In order to state our condition, we define the notion of *honest trace*. Intuitively, it corresponds to a trace in which the attacker does not interfere, except to simply forward messages without modifying them.

Definition 9: A trace tr is *honest* for a frame ϕ if $\tau_{\mathsf{else}} \notin \mathsf{tr}$ and $\mathsf{obs}(\mathsf{tr})$ is of the form $\mathsf{out}(c, w_0).\mathsf{in}(c, R_0).\mathsf{out}(c', w_1).\mathsf{in}(c', R_1).\mathsf{out}(c, w_2)\ldots$ where $\{c, c'\} = \{c_T, c_R\}$ and such that

- each input is preceded by an output on the same channel, and followed by an output on the other channel,
- $R_i\phi\Downarrow =_{\mathsf{E}} w_i\phi$ for any action $\mathsf{in}(\_, R_i)$ occurring in tr.

Note that τ_{abort} is allowed in an honest trace.

Definition 10: Two annotations a and a' have an honest interaction in (ta, ϕ) if the subsequence of ta that consists of action of the form $\alpha[a]$ or $\alpha[a']$ is honest for ϕ.

We are now able to state our condition.

Definition 11: The protocol Π is *well-authenticating* if for any

$$\mathcal{M}_\Pi \xrightarrow{\mathsf{ta}.\tau_{\mathsf{then}}[a]} (\mathcal{Q}; \phi; \mathcal{S}')$$

either the last action corresponds to a safe conditional of T or R, or there exists a' such that the annotations a and a' have an honest interaction in (ta, ϕ).

Example 14: Our running example, the Basic Hash protocol, is actually well-authenticating. More practical details on how we prove it will be given in Section VI. The intuition is that the only way for an attacker to build a message that would be accepted by the reader is to replay a tag's output (because the key stored in the state remains secret). Thus, for any replayed tag's output, there exists a (real) tag with which the reader has an honest interaction in the considered trace.

This example also shows that well-authentication only encodes a weak requirement of authentication protocols, as it allows some replay attacks.

D. No desynchronisation

This third condition states that an honest interaction between a tag and a reader cannot fail. The intuition is that, in case a protocol does not correctly handle the situations where the tag and the reader are desynchronised, it could happen that an honest interaction does not pass successfully a conditional because of a mismatch in the states values, thus leaking some information to the attacker.

Definition 12: The protocol Π ensures that *no desynchronisation* occurs if, for any $\mathcal{M}_\Pi \xrightarrow{\mathsf{ta}.\tau_x[a]} (\mathcal{P}; \phi; \mathcal{S}')$ where the last action is performed by an unsafe conditional, we have: if there exists a' such that a and a' have an honest interaction in (ta, ϕ) then $\tau_x = \tau_{\mathsf{then}}$.

Example 15: No desynchronisation holds for the Basic Hash protocol: since the database and references are never updated, an honest interaction between a tag and reader can

only result in a τ_{then}. We discuss further the verification of no desynchronisation in Section VI, notably covering the case of protocols with updates.

E. Main result

Our main result establishes that the previous three conditions are sufficient to ensure unlinkability.

Theorem 1: Let Π be a protocol together with an idealisation operator $\mathrm{ideal}(.)$ for it, such that:

- Π ensures frame opacity w.r.t. to ideal;
- Π is well-authenticating; and
- Π ensures that no desynchronisation occurs.

We have that Π satisfies unlinkability.

To establish this result, we show that any execution of $\mathcal{M}_\Pi$ can be mimicked by an execution of $\mathcal{S}_\Pi$ leading to the same observables. Roughly, we first show that $\mathcal{M}_\Pi$ and $\mathcal{S}_\Pi$ can form a bi-process, putting the states apart, and then prove that the two sides of this bi-process can evolve simultaneously: in particular, the outcome of conditionals is the same on both sides and frames are in static equivalence. Technically, a notion of *ground configuration* [21] is used to put together $\mathcal{M}_\Pi$ and $\mathcal{S}_\Pi$ as a bi-process. See Appendix B for details.

Let us sketch how our three conditions play a role in showing that conditionals have the same outcome on both sides of the bi-process. Suppose a conditional has a positive outcome in an execution of $\mathcal{M}_\Pi$. By well-authentication, it must be the result of an honest interaction. Thanks to frame opacity the relationships between recipes are still satisfied in the execution of $\mathcal{S}_\Pi$, and thus our honest interaction for $\mathcal{M}_\Pi$ maps to an honest interaction for $\mathcal{S}_\Pi$. We can finally use no desynchronisation, which holds on $\mathcal{S}_\Pi$, to conclude that the outcome of the conditional will be positive in the execution of $\mathcal{S}_\Pi$. A similar reasoning allows us to conclude for a negative outcome of a conditional, relying first on the no desynchronisation condition for $\mathcal{M}_\Pi$ and then on well-authentication for $\mathcal{S}_\Pi$. Note that our two conditions are satisfied by $\mathcal{M}_\Pi$ and thus $\mathcal{S}_\Pi$ as well since $\mathcal{S}_\Pi$ has less behaviours than $\mathcal{M}_\Pi$.

The most striking difference between our result and the one of [21], besides the modified definition of unlinkability, is the introduction of the no desynchronisation condition. This new condition is necessary to handle protocols with states: it ensures that, in the end, the specific states of honest agents do not impact the outcome of conditionals. There are more subtle differences: for instance, our well-authentication condition is more permissive than the one of [21], which features a further constraint that becomes both useless and meaningless with our generic readers.

In the stateless case [21], it can be shown that unlinkability would systematically fail if expressed using bisimulation rather than trace equivalence. When trying to mimick a multiple-session execution on the single-session side, one has to look forward in the trace to know which successful interactions will actually happen. The situation is simpler in our case: any new tag session on the multiple-session side must be mimicked by a new identity on the single-session side, and there is no question of which identities should be picked by the reader sessions, as our readers are generic. As a consequence, it should be possible to show that our conditions actually imply a stronger notion of unlinkability where trace equivalence is replaced by some form of bisimulation, or even diff-equivalence. However, these equivalences would have to be lax enough regarding states: as explained at the beginning of this section, existing diff-equivalences cannot be used to prove unlinkability; accordingly, the bi-processes we form in our proof of Theorem 1 do not superpose the single- and multiple-session states, but allow them to evolve independently.

V. Towards automation

Existing tools at our disposal for verifying our sufficient conditions in the unbounded case are ProVerif [12], [1] and Tamarin [10], [2]. In order to analyse our protocols of interest, we need a tool that supports stateful protocols. Handling global states is a known limitation of ProVerif. Our attempts to use private channels to model memory cells have resulted in non-termination issues. We have also experimented the recent extension GSVerif [17], but failed again due to non-termination [3].

On the other hand, Tamarin is naturally well-suited to handle global states. Morever, unlike ProVerif, it supports the XOR operator and provides an interactive mode that can be used to complete the more difficult proofs. We thus chose Tamarin to verify unlinkability of stateful protocols with our method. Despite all these qualities, obtaining satisfying results with Tamarin has required significant effort. We explain here the main choices we made about how to encode the protocols and our conditions.

A. Tamarin *in a nutshell*

We give here a brief introduction to the main features of Tamarin, referring the reader to [2] for more details. In Tamarin, security protocols and attackers are modelled symbolically using multiset rewriting: a state of the system is a multiset of *facts*, and transitions between states are given by rules of the form $l -[a]\rightarrow r$ where l and r are sequences of facts and a is a sequence of *labels*. There are two types of facts: *linear* facts are consumed by a rule while *persistent* facts (written !F) can be consumed arbitrarily often, *i.e.* they are never removed from the state once introduced.

Trace properties are expressed in a fragment of first-order logic over labels, messages and timepoints. These formulas are used as *lemma* statements to verify trace properties of the modelled protocol. Lemmas may be proved automatically or using an interactive mode where the user can manually guide and inspect the proof. Some lemmas can be re-used as helpers in the next proofs. Finally, the tool provides *restrictions*, which are logical formulas constraining the set of traces to be considered in the analysis.

In order to prove observational equivalence between two transition systems, the two systems should be superposed into a single model containing the diff operator, allowing different

messages for the two systems but a shared structure for the states and the transitions.

B. Well-authentication and no desynchronisation

We check well-authentication and no desynchronisation within the same Tamarin file, *i.e.* using the same model of the studied protocol. We comment on this shared model, before discussing the encodings of our two conditions as lemmas.

For illustration purposes, we show next the model for our running example, the Basic Hash protocol. We will give some details about this implementation later on.

```
theory BasicHash_WA_ND
begin

functions: h/1, OK/0

/*************************************
PROTOCOL
*************************************/

rule InitReader:
  [ Fr(~sidR) ]
  --[ NewReaderSession(~sidR) ]->
  [ Reader(~sidR) ]

rule InitTagId:
  [ Fr(~k) ]
  --[ NewId(~k), InsertDB(~k) ]->
  [ !DB(~k), !TagSession(~k) ]

rule InitTagSession:
  [ !TagSession(~kT), Fr(~sidT) ]
  --[ NewTagSession(~sidT,~kT) ]->
  [ Tag(~sidT,~kT) ]

/* READER */

rule R_in:
  [ Reader(~sidR), In(x) ]
  --[ InR(~sidR,x) ]->
  [ Reader1(~sidR,x) ]

rule R_test:
  let x = <xnT,h(<kR,xnT>)> in
  [ !DB(kR), Reader1(~sidR,x) ]
  --[ TestR(~sidR), CompleteR(~sidR) ]->
  [ Out(OK) ]

/* TAG */

rule T_out:
  let m = <~nT,h(<kT,~nT>)> in
  [ Tag(~sidT,kT), Fr(~nT) ]
  --[ PlayT(~sidT,kT), OutT(~sidT,kT,~nT,m),
      CompleteT(~sidT) ]->
  [ Out(m) ]
```

This model contains a few elements that are not strictly necessary for illustrating the encoding of our conditions, but which we systematically use in our models and thus take the opportunity to present here. First, the `sidT` and `sidR` nonces are added to all tag and reader roles as identifiers, even though in this case `nT` could have been used to identify tags. Second, the `CompleteT` and `CompleteR` labels are used in sanity check lemmas (not shown here) to verify that our models can execute properly. Finally, the `PlayT` label would be useful to impose sequentiality on tag sessions using a restriction, for protocols where tags perform more than a single output.

To model sequentiality of a tag's successive sessions, we use restrictions. More specifically, the transition rules modelling our protocols allow many sessions of a given tag to run in parallel but we then use restrictions to specify the set of traces that Tamarin should actually consider when proving our conditions, notably imposing sequentiality of a tag's sessions. This has proved to be helpful for the tool to conclude automatically, instead of encoding directly the sequentiality in the protocol's workflow. Concretely, the sequential restriction is written as follows, building on the systematic use of the `PlayT` label in tag rules:

```
restriction seqSessionTag:
  "not (Ex sidT1 sidT2 kT #i1 #i2 #i3.
    PlayT(sidT1,kT)@i1 &
    PlayT(sidT2,kT)@i2 &
    PlayT(sidT1,kT)@i3 &
    i1<i2 & i2<i3 &
    not(sidT1=sidT2))"
```

We model the presence of a message x in the database by a fact $\mathsf{DB}(x)$. When the reader performs a lookup in the database, we consume the fact $\mathsf{DB}(x)$ that verifies the conditional and we create a new fact $\mathsf{DB}(x')$ where x' is the updated value of the state. When working with monotonic states (*i.e.* never updated), it has proved useful to use persistent facts $!\mathsf{DB}(x)$.

Our Tamarin files also feature intermediate lemmas expressing invariants of the protocol or secrecy of some data, marking them as reusable to guide our proofs.

We pointed out in Section III-C that modelling error messages is important when verifying unlinkability because the outcome of conditionals is often observable in real uses of authentication protocols. However, error messages are not always represented in our Tamarin models. Indeed, encoding else branches with Tamarin is not straightforward when dealing with facts $\mathsf{DB}(x)$ representing the database, and it is actually not necessary when verifying well-authentication and no desynchronisation (else branches do not contribute to the attacker's knowledge, and are not involved in honest executions).

a) Well-authentication: The way we encode this condition is very similar to what is done in [21] and Tamarin is well suited to write such lemmas. Basically, we check that, for each successful conditional of a given agent, the trace contains an honest interaction of this agent with another agent of the opposite role (*i.e.* an alternation of inputs and outputs where each input is equal modulo the equational theory to the previous output).

For our model of the Basic Hash protocol, well-authentication is expressed through the following lemma.

```
lemma WA_Reader [use_induction]:
  "All sidR #i3.
    TestR(sidR)@i3 ==>
      ( Ex sidT kT nT m #i1 #i2.
        ( InR(sidR,m)@i2 &
          OutT(sidT,kT,nT,m)@i1 &
          i1<i2 & i2<i3 ))"
```

b) No desynchronisation: Encoding this condition is more subtle than well-authentication but the flexibility allowed by Tamarin to write lemmas is helpful. We verify that, whenever an agent has reached a point where a conditional is going to be evaluated, if this agent is having an honest interaction with another agent of the opposite role, then the conditional will be positively evaluated. The way we encode that *the conditional will be positively evaluated* changes depending on the type of conditional. For let conditionals, we only have to check that the test holds: it generally consists in checking that a message is of the expected form. The same holds for lookup conditionals in the case of a protocol where the database is monotonic: if the test holds at some point in time, it will hold at any future point since the database is never updated; the only difference is that the test will involve database facts. When the database is updated, verifying no desynchronisation for lookup conditionals is more involved: we must check that the test will hold at any point in the future. It generally consists in checking that a message is of the expected form, that some appropriate database entry has been inserted in the database, and that it would still be there in the future when the test might be evaluated.

The encoding of the no desynchronisation condition for our Basic Hash model (for which the database is monotonic) is as follows:

```
lemma ND_Reader [use_induction]:
  "All sidT kT nT sidR m #i1 #i2.
    (InR(sidR,m)@i2 &
     OutT(sidT,kT,nT,m)@i1 &
     i1<i2)
     ==>
    (Ex kR xnT #i0.
     m = <xnT,h(<kR,xnT>)> &
     InsertDB(kR)@i0 &
     i0<i2)"
```

C. Frame opacity

Similarly to [21], we check frame opacity using a notion of diff-equivalence. We adapt the encoding of [21] which uses the diff-equivalence feature of ProVerif, in order to use the observational equivalence mode of Tamarin. In this mode, we write bi-systems, *i.e.* systems that only differ in terms using the diff operator. In the case of frame opacity, the left-hand side system encodes the real execution and, in the right-hand side system, we replace each outputted message by its idealisation. If Tamarin proves that these two systems are equivalent, then we have that the resulting frames are in static equivalence so frame opacity holds.

Frame opacity is defined w.r.t. an idealisation operator. In our case studies, we use an idealisation that maps hashes and encryptions to fresh nonces, public constants to themselves, and pairs of terms to pairs of the terms' idealisations.

The main problem is that conditionals lose their meaning on the right-hand side of the bi-system: the conditions are meant for the real messages, not their idealisations. To avoid this mismatch, which would break diff-equivalence, we over-approximate the protocol by removing the conditionals. We thus verify that frame opacity holds for a larger set of possible traces, which is sound.

For some case studies, conditionals are used to define new variables that are used later on. When removing the conditionals, we thus lose the definition of these variables. In order to overcome this difficulty, we enrich the equational theory. Consider for example an input x equal to $\mathsf{senc}(n,k)$ in the real frame and equal to a fresh nonce n' in the idealised frame. In order to recover n without using a conditional in the real execution (left-hand side system) we introduce a private constructor extract with the equation $\mathsf{extract}(\mathsf{senc}(y,z),z) = y$. Then $\mathsf{extract}(x)$ will reduce to n in the left-hand side system, and will not fail in the right-hand side system since extract is not a destructor symbol.

Over-approximating the protocol as explained above has sometimes not been sufficient to make Tamarin conclude. For some case studies, we have used open variables, *i.e.* inputs that the attacker can fill with any value, including the real values. But here again, it is sound to check frame opacity on a set of traces that is larger than the actual traces of the protocol.

In order to verify frame opacity for the Basic Hash protocol, we use the observational equivalence mode on a modified model, where the reader's test is removed and the output of the tag is a diff-term. The model is thus identical to the previous one except for two rules:

```
rule R_test:
  // let x = <xnT,h(<kR,xnT>)> in
  [ !DB(kR), Reader1(~sidR,x) ]
  --[ TestR(~sidR), CompleteR(~sidR) ]->
  [ Out(OK) ]

rule T_out:
  let m = diff(<~nT,h(<kT,~nT>)>,<~nId1,~nId2>) in
  [ Tag(~sidT,kT), Fr(~nT), Fr(~nId1), Fr(~nId2) ]
  --[ PlayT(~sidT,kT), OutT(~sidT,kT,~nT,m),
      CompleteT(~sidT) ]->
  [ Out(m) ]
```

VI. Case studies

We have applied our method to several case studies. Our results are summarised in Figure 6, descriptions of the protocols are available in Appendix C, and the Tamarin files to reproduce these results can be found at [3]. We have found two attacks on unlinkability for the OSK and LAK protocols that had not been previously reported in the literature, and propose fixes that we prove unlinkable using our method.

For a few cases, we have used the interactive mode of Tamarin to conclude. In most cases, Tamarin is able to conclude automatically in a few seconds (the worst cases take about one minute). But it has required a lot of time to implement these case studies and to write appropriate intermediate lemmas in such a way that Tamarin can prove automatically our conditions.

A. Basic Hash, Hash-Lock and Feldhofer

We study here three examples falling into our class of protocols and being similar in the sense that they involve a monotonic state (*i.e.* never updated). Our sufficient conditions

	Unlink.	WA	FO	ND
Basic Hash	ok	✓	✓	✓
Hash-Lock	ok	✓	✓	✓
Feldhofer	ok	✓	✓	✓
OSK (v1)	**attack**	✓*		✗
OSK (v2)	ok	✓	✓	✓
LAK (pairs)	**attack**	✓*		✗
LAK (pairs, fix v1)	ok	✓	✓	✓
LAK (pairs, fix v2)	ok	✓*	✓	✓
5G-AKA (simplified)	ok	✓	✓	✓

Fig. 6. Summary of case studies: ✓ means the condition hold, ✗ means the conditions does not hold. These conditions are verified automatically with Tamarin, except for those with a * meaning the interactive mode has been used.

can be automatically verified with Tamarin, thus we conclude that these protocols ensure unlinkability.

Our conclusions regarding these three protocols are in line with the literature. Note that if the Basic Hash protocol had been studied in [21], it would have been declared linkable for the reason we detailed in Section III-C. In [10], the Feldhofer protocol is analysed using the observational equivalence mode of Tamarin but for a notion of privacy that is different from our definition of unlinkability.

B. OSK

The OSK protocol is proved unlinkable in [15] but in a simplified model, as explained in Section III-C. A privacy vulnerability in the OSK protocol is highlighted in [23], but this attack exploits an upper bound on the number of operations in the tag's lifetime, which is a practical consideration that we do not consider here. We present here two versions inspired from the original description of the OSK protocol that can be found in [29].

The first version (v1) is the one described in Example 9 for which we highlight an attack scenario. On this version, Tamarin concludes automatically that our no desynchronisation condition does not hold.

The second version (v2) differs only in the reader's conditional. Instead of expecting a message of the form $\mathsf{g}(\mathsf{hash}^n(x))$ for some database entry x (to allow resynchronisations) and updating that entry with $\mathsf{hash}^{n+1}(x)$ (to avoid replay attacks), the reader never updates the database and accepts any message of the form $\mathsf{g}(\mathsf{hash}^n(k))$ for some initial shared key k. In this second version, an attacker can replay a message but this is not an issue for unlinkability. Indeed, Tamarin is able to prove that our three conditions hold.

In order to check frame opacity on OSK v2, we do not simply remove the reader's conditional as explained in Section V-B but we also over-approximate the protocol workflow. Since the reader's conditional is removed, this role becomes useless in the sense that it only outputs a public constant that stays the same in the idealised frame. We can thus remove all reader rules without affecting diff-equivalence. We also over-approximate the possible values for the tags' states, by allowing the attacker to choose them. Over-approximating the protocol in such a way helps Tamarin to terminate. Since diff-equivalence holds for a larger set of traces (containing the real ones) we conclude that frame opacity holds.

In our Tamarin models, we represent the successive applications of the same hash functions by using a multiset of the constant 1. For example, we represent $\mathsf{hash}^3(x)$ with $\mathsf{hash}(x, 1+1+1)$. This allows us to express conditions such as $\exists m \leq n.\ x = \mathsf{h}^m(k)$, but has an impact on the attacker model. This is why we have added an extra rule to represent the fact that the attacker can obtain $\mathsf{hash}^{n+1}(x)$ from $\mathsf{hash}^n(x)$.

C. LAK

The LAK protocol as described in [33] is a stateful protocol using the XOR operator. As highlighted in [21], this protocol suffers from an authentication attack (based on algebraic properties of the XOR operator) which can be turned into an attack on unlinkability. An analysis of unlinkability for the LAK protocol using Tamarin is done in [19] but only for a bounded number of sessions, and this same attack is found.

In [21], a stateless version of this protocol where the XOR operator is replaced by the pairing operator was proved unlinkable. The three different versions of the LAK protocol we study here also replace the XOR operator by pairs.

We first analyse a version of the protocol that is very close to the original one: the only difference is the replacement of the XOR operator by the pairing operator, *i.e.* the way tags and readers update their states is faithful to the description in [33]. We show that this version suffers from an attack on unlinkability, based on the fact that there exists a scenario in which a tag can be desynchronised w.r.t. the states stored in the database, thus leaking information to the attacker. This scenario is depicted in Figure 15 of Appendix C. As expected, our no desynchronisation condition fails; Tamarin is able to automatically find an attack trace.

Then, we propose two possible fixes to correct this flaw (fix v1 and fix v2 in Figure 6). Our first fix is the stateless version as proposed in [21] but with generic readers having access to a common database, which is closer to the original specification. Our second fix consists in modifying the reader's conditional: for a given value $\langle k_0, n\rangle$ in the database, the reader will accept any message containing $\mathsf{hkey}^p(k_0)$ with $p \leq n$, instead of accepting only $\mathsf{hkey}^n(k_0)$ or $\mathsf{hkey}^{n-1}(k_0)$. Thus, our fix handles unbounded desynchronisations, whereas the original specification only handled desynchronisation by one step. Note that, even if this second fix is more complex than the first one, a protocol where states are updated at each session is interesting when considering properties such as forward privacy. Using Tamarin to check our three conditions, we conclude that these two fixes ensure unlinkability.

Our models in Tamarin use two different hash functions: a function h for the messages and another one, hkey, for the key updates. Similarly to the OSK protocol, we also represent the successive applications of the same hash functions by using a multiset of the constant 1 and we use the same extra rule to complement the attacker's capacities.

The fixed version v2 required more work to automatically verify our conditions. Lemmas checking no desynchronisation are more complex to write since we are in the case where states in the database are updated, and proving them required more intermediate lemmas than in other cases.

D. Simplified 5G-AKA

The AKA protocol, designed by the *3rd Generation Partnership Project* (3GPP) which is responsible for the standardisation of 3G, 4G and 5G technologies, aims at authenticating a subscriber to its service provider and establishing shared keys for future communications.

A number of linkability attacks (resp. fixes) have been shown (resp. proposed) in the literature [7], [20], [24], [13]. We study here a simplified version of the AKA protocol, inspired from the different fixes proposed in the literature. More specifically, we add a challenge-response mechanism at the beginning of the protocol to avoid the encrypted IMSI replay attack [20]. With this modification, we add the generation of fresh nonces for the network and the mobile station, thus the resynchronisation mechanism using the sequence number becomes useless and we remove this sequence number, which takes away the main difficulty of the original protocol. In the end, we have a stateful protocol with monotonic states (*i.e.* never updated) for which we are able to prove, using Tamarin, that our three conditions hold. We conclude that this simplified version of the protocol ensures unlinkability.

VII. Limitations

We discuss in this section a number of limitations of our work. Obvious ones include the restriction to two-party protocols and the assumption that tags and readers are not corrupted. We believe that the first restriction could be lifted relatively easily, but do not know of case studies that would be enabled in this way. Concerning corruption, it would be interesting to incorporate it in our notion of unlinkability: we leave it for future work. We discuss below the other limitations concerning the method and its mechanisation.

A. Method

Our approach based on sufficient conditions is not complete: there are protocols that are unlinkable but do not satisfy our conditions. In particular, our conditions impose that the outcome of conditionals is the same in the real world (multiple sessions) and in the ideal one (single session), but one can easily design a toy protocol that is unlinkable but does not verify such a property. However, we are not aware of practical cases where this source of incompleteness is an issue.

Our Definition 9 of an *honest trace* is arbitrary and might be too restrictive for some protocols. This can be seen by taking any protocol proved unlinkable with our method, and adding a simple exchange at the beginning of the protocol, that does not break unlinkability: the tag's session starts by receiving a public constant get_challenge sent by the reader. The expected interaction would thus be as follows:

$$\begin{array}{rcl} R & : & \mathsf{out}(c_R, \mathsf{get\_challenge}) \\ T & : & \mathsf{in}(c_R, \mathsf{get\_challenge}) \end{array}$$

But the following one is also possible, since get_challenge is public and thus derivable by the attacker:

$$\begin{array}{rcl} T & : & \mathsf{in}(c_R, \mathsf{get\_challenge}) \\ R & : & \mathsf{out}(c_R, \mathsf{get\_challenge}) \end{array}$$

Our well-authentication condition would not hold anymore due to this possible (benign) interaction, because honest traces require that each input is preceded by its corresponding output.

We believe that such examples could be avoided by adopting a richer notion of honest trace. We believe that our theoretical results can be adapted, as long as both well-authentication and no desynchronisation use the same notion of honest trace (this is indeed a key element for the proof of our main result).

B. Mechanisation

Regarding the mechanisation of our method, we hope to see further improvements of verification tools (Tamarin or new ones) that would help users to verify our conditions in more cases and with less effort. We discuss below two points that are particularly important in that respect.

Many protocols falling in the scope of this paper use the XOR operator, but we were not able to apply our method to these protocols because the support for the XOR operator in Tamarin is not yet sufficient. As an example, Tamarin does not terminate on the simple static equivalence illustrated in Example 6 (see [3] for the code). It seems to us that improving Tamarin's XOR support is necessary to be able to apply our method to protocols using XOR.

Another aspect which limits the mechanisation of our method is the lack of more inductive reasoning in Tamarin. Using inductive proof methods is helpful when dealing with protocols with mutable states, but it is currently only available for proving trace properties and not in observational equivalence mode. We encountered non-termination issues when checking frame opacity for protocols where the states are updated. We overcame this difficulty by significantly simplifying the protocol's workflow, as explained in Section V: this is not an issue for soundness, but limiting this non-trivial manual step would be a key to further automate our method.

VIII. Conclusion

We have adapted the method of [21] to verify unlinkability for stateful protocols. We have proposed a definition of unlinkability that is well-suited to such protocols, improving on earlier propositions. We have adapted the two conditions originally proposed in [21], identified a third, no desynchronisation condition, and proved that these three conditions imply unlinkability. This provides a method to verify unlinkability for an unbounded number of sessions, which we have applied on several case studies using Tamarin. We have thus found new attacks and obtained new proofs of unlinkability.

A way to go further would be to bridge the gap between the formal model of Tamarin, based on multiset rewriting, and our formal model based on the applied-pi calculus, which is better suited to our theoretical developments relying *e.g.* on the notions of conditionals and honest execution. For this purpose,

we could use a tool such as SAPIC [25]. It would allow us to describe our protocols in a language close to the applied-pi calculus while still using, after an automatic translation step, the Tamarin language to express and verify conditions. A potential drawback of this approach is that the generated code might not be satisfying: currently, several details of our manual encodings are crucial to obtain automatic proofs; these design choices would have to be systematised.

REFERENCES

[1] [Online]. Available: https://prosecco.gforge.inria.fr/personal/bblanche/proverif/

[2] [Online]. Available: https://tamarin-prover.github.io/

[3] See supplementary files attached to this report.

[4] "5G PPP phase 1 security landscape," 5G PPP Security Working Group, 2017. [Online]. Available: https://5g-ppp.eu/wp-content/uploads/2014/02/5G-PPP_White-Paper_Phase-1-Security-Landscape_June-2017.pdf

[5] M. Abadi and C. Fournet, "Mobile values, new names, and secure communication," in *Conference Record of POPL 2001: The 28th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, London, UK, January 17-19, 2001*, C. Hankin and D. Schmidt, Eds. ACM, 2001, pp. 104–115.

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

[7] M. Arapinis, L. I. Mancini, E. Ritter, M. Ryan, N. Golde, K. Redon, and R. Borgaonkar, "New privacy issues in mobile telephony: fix and verification," in *the ACM Conference on Computer and Communications Security, CCS'12, Raleigh, NC, USA, October 16-18, 2012*, T. Yu, G. Danezis, and V. D. Gligor, Eds. ACM, 2012, pp. 205–216. [Online]. Available: https://doi.org/10.1145/2382196.2382221

[8] M. Arapinis, J. Phillips, E. Ritter, and M. D. Ryan, "StatVerif: Verification of stateful processes," *Journal of Computer Security*, vol. 22, no. 5, pp. 743–821, 2014.

[9] A. Armando *et al.*, "The AVANTSSAR platform for the automated validation of trust and security of service-oriented architectures," in *Tools and Algorithms for the Construction and Analysis of Systems - 18th International Conference, TACAS 2012, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2012, Tallinn, Estonia, March 24 - April 1, 2012. Proceedings*, ser. Lecture Notes in Computer Science, C. Flanagan and B. König, Eds., vol. 7214. Springer, 2012, pp. 267–282.

[10] D. A. 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, Denver, CO, USA, October 12-16, 2015*, I. Ray, N. Li, and C. Kruegel, Eds. ACM, 2015, pp. 1144–1155.

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

[12] B. Blanchet, M. Abadi, and C. Fournet, "Automated verification of selected equivalences for security protocols," *J. Log. Algebr. Program.*, vol. 75, no. 1, pp. 3–51, 2008.

[13] R. Borgaonkar, L. Hirschi, S. Park, and A. Shaik, "New privacy threat on 3G, 4G, and upcoming 5G AKA protocols," *PoPETs*, vol. 2019, no. 3, pp. 108–127, 2019.

[14] A. Bruni, S. Mödersheim, F. Nielson, and H. R. Nielson, "Set-Pi: Set membership p-calculus," in *IEEE 28th Computer Security Foundations Symposium, CSF 2015, Verona, Italy, 13-17 July, 2015*, C. Fournet, M. W. Hicks, and L. Viganò, Eds. IEEE Computer Society, 2015, pp. 185–198.

[15] M. Brusò, K. Chatzikokolakis, and J. den Hartog, "Formal verification of privacy for RFID systems," in *Proceedings of the 23rd IEEE Computer Security Foundations Symposium, CSF 2010, Edinburgh, United Kingdom, July 17-19, 2010*. IEEE Computer Society, 2010, pp. 75–88.

[16] M. Brusò, K. Chatzikokolakis, S. Etalle, and J. den Hartog, "Linking unlinkability," in *Trustworthy Global Computing - 7th International Symposium, TGC 2012, Newcastle upon Tyne, UK, September 7-8, 2012, Revised Selected Papers*, ser. Lecture Notes in Computer Science, C. Palamidessi and M. D. Ryan, Eds., vol. 8191. Springer, 2012, pp. 129–144.

[17] V. Cheval, V. Cortier, and M. Turuani, "A little more conversation, a little less action, a lot more satisfaction: Global states in ProVerif," in *31st IEEE Computer Security Foundations Symposium, CSF 2018, Oxford, United Kingdom, July 9-12, 2018*. IEEE Computer Society, 2018, pp. 344–358.

[18] V. Cheval, S. Kremer, and I. Rakotonirina, "The DEEPSEC prover," in *Computer Aided Verification - 30th International Conference, CAV 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 14-17, 2018, Proceedings, Part II*, ser. Lecture Notes in Computer Science, H. Chockler and G. Weissenbacher, Eds., vol. 10982. Springer, 2018, pp. 28–36.

[19] 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, Oxford, United Kingdom, July 9-12, 2018*. IEEE Computer Society, 2018, pp. 359–373.

[20] P. Fouque, C. Onete, and B. Richard, "Achieving better privacy for the 3GPP AKA protocol," *PoPETs*, vol. 2016, no. 4, pp. 255–275, 2016.

[21] L. Hirschi, D. Baelde, and S. Delaune, "A method for unbounded verification of privacy-type properties," *Journal of Computer Security*, vol. 27, no. 3, pp. 277–342, 2019.

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

[23] A. Juels and S. A. Weis, "Defining strong privacy for RFID," *ACM Trans. Inf. Syst. Secur.*, vol. 13, no. 1, pp. 7:1–7:23, 2009.

[24] A. Koutsos, "The 5G-AKA authentication protocol privacy," in *IEEE European Symposium on Security and Privacy, EuroS&P 2019, Stockholm, Sweden, June 17-19, 2019*. IEEE, 2019, pp. 464–479.

[25] S. Kremer and R. Künnemann, "Automated analysis of security protocols with global state," *Journal of Computer Security*, vol. 24, no. 5, pp. 583–616, 2016.

[26] S. Meier, B. Schmidt, C. Cremers, and D. A. Basin, "The TAMARIN prover for the symbolic analysis of security protocols," in *Computer Aided Verification - 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings*, 2013, pp. 696–701.

[27] S. Mödersheim and A. Bruni, "AIF-ω: Set-based protocol abstraction with countable families," in *Principles of Security and Trust - 5th International Conference, POST 2016, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2016, Eindhoven, The Netherlands, April 2-8, 2016, Proceedings*, ser. Lecture Notes in Computer Science, F. Piessens and L. Viganò, Eds., vol. 9635. Springer, 2016, pp. 233–253.

[28] S. Mödersheim and L. Viganò, "Alpha-beta privacy," *ACM Trans. Priv. Secur.*, vol. 22, no. 1, pp. 7:1–7:35, 2019.

[29] M. Ohkubo, K. Suzuki, S. Kinoshita *et al.*, "Cryptographic approach to "privacy-friendly" tags," in *RFID privacy workshop*, vol. 82. Cambridge, USA, 2003.

[30] J. Schwartz, "Researchers see privacy pitfalls in no-swipe credit cards," *New York Times*. [Online]. Available: https://www.nytimes.com/2006/10/23/business/23card.html

[31] D. Strobel, "IMSI catcher," *Chair for Communication Security, Ruhr-Universität Bochum*, vol. 14, 2007.

[32] T. van Deursen, S. Mauw, and S. Radomirovic, "Untraceability of RFID protocols," in *Information Security Theory and Practices. Smart Devices, Convergence and Next Generation Networks, Second IFIP WG 11.2 International Workshop, WISTP 2008, Seville, Spain, May 13-16, 2008. Proceedings*, ser. Lecture Notes in Computer Science, J. A. Onieva, D. Sauveron, S. Chaumette, D. Gollmann, and C. Markantonakis, Eds., vol. 5019. Springer, 2008, pp. 1–15.

[33] T. van Deursen and S. Radomirovic, "Attacks on RFID protocols," *IACR Cryptology ePrint Archive*, vol. 2008, p. 310, 2008.

[34] S. Vaudenay, "On privacy models for RFID," in *Advances in Cryptology - ASIACRYPT 2007, 13th International Conference on the Theory and Application of Cryptology and Information Security, Kuching, Malaysia, December 2-6, 2007, Proceedings*, ser. Lecture Notes in Computer Science, K. Kurosawa, Ed., vol. 4833. Springer, 2007, pp. 68–87.

APPENDIX A
DISCUSSION ON UNLINKABILITY

A. Comparing with weak and strong unlinkability

Intuitively, weak unlinkability [6] requires that, for any *two* observable tag actions in an annotated trace, there exists an indistinguishable trace where the corresponding actions either belong to the same session or to different identities. In other words, just looking at these two actions, it could be that they originate from a single session scenario.

We reformulate the formal definition of weak unlinkability [6, Definition 11] in our setting, adapting it to our framework and specialising it for tag unlinkability but preserving its essence. In this definition, ta_i denotes the i$^{\text{th}}$ *observable* action in an annotated trace ta.

Definition 13 (Weak unlinkability): A protocol Π ensures *weak unlinkability* for its tags when:

- for any annotated trace ta and any execution $\mathcal{M}_\Pi \xrightarrow{\mathsf{ta}} (\mathcal{P};\phi;\mathcal{S})$,
- for any i, j such that ta_i and ta_j carry tag annotations,
- there exists an annotated trace ta' and an execution $\mathcal{M}_\Pi \xrightarrow{\mathsf{ta}'} (\mathcal{P}';\phi';\mathcal{S}')$ such that $\mathsf{obs}(\mathsf{ta}) = \mathsf{obs}(\mathsf{ta}')$ and $\phi \sim \phi'$ and
- ta'_i and ta'_j carry tag annotations corresponding either to a same session or to different identities, *i.e.* ta'_i and ta'_j are either the same tag annotation or two annotations of the form $T(r, \overline{n}_T)$ and $T(r', \overline{n}'_T)$.

Assuming that the attacker can tell, for each observable action, the role from which it originates, we can easily show that our notion of unlinkability implies weak unlinkability.

Proposition 1: Let Π be a protocol (according to Definition 4). If $\mathcal{M}_\Pi \approx \mathcal{S}_\Pi$, then Π ensures weak unlinkability for its tags.

Proof. Consider a trace ta and positions i and j as in Definition 13. By $\mathcal{M}_\Pi \approx \mathcal{S}_\Pi$ we have an annotated trace ta' that can be executed by $\mathcal{S}_\Pi$, leading to an indistinguishable frame. By our assumption on Π, and because $\mathsf{obs}(\mathsf{ta}') = \mathsf{obs}(\mathsf{ta})$, ta'_i and ta'_j still carry tag annotations. Finally, by definition of $\mathcal{S}_\Pi$, these tag annotations either belong to the same session or to two distinct identities. □

The converse implication does not hold, as shown by the next example, where we use some obvious syntactic sugar for computing disjunctions in a test.

Example 16: Consider a protocol $\Pi = (i, \mathsf{ok}, T, R)$ with the following role processes, where k is a private constant, and i and n are names respectively encoding a tag identity and a tag session nonce:

$$
\begin{aligned}
T \;&:=\; \mathsf{get}(r,y).\mathsf{out}(c_T, \{y,n\}_{\mathsf{k}}) \\
R \;&:=\; \mathsf{in}(c_T,x).\mathsf{in}(c_T,y).\mathsf{in}(c_T,z). \\
&\quad\; \mathsf{if}\ \mathsf{fst}(\mathsf{dec}(x,\mathsf{k})) = \mathsf{fst}(\mathsf{dec}(y,\mathsf{k})) \\
&\quad\; \mathsf{or}\ \mathsf{fst}(\mathsf{dec}(z,\mathsf{k})) = \mathsf{fst}(\mathsf{dec}(x,\mathsf{k})) \\
&\quad\; \mathsf{or}\ \mathsf{fst}(\mathsf{dec}(y,\mathsf{k})) = \mathsf{fst}(\mathsf{dec}(z,\mathsf{k}))\ \mathsf{then} \\
&\quad\; \mathsf{out}(c_R, \mathsf{ok})
\end{aligned}
$$

With this protocol, the attacker can use the reader to test whether two tag outputs out of three correspond to the same identity. However, he cannot tell which two outputs correspond to the same identity. Because the output ok on c_R can be observed with $\mathcal{M}_\Pi$ but not with $\mathcal{S}_\Pi$, we have $\mathcal{M}_\Pi \not\approx \mathcal{S}_\Pi$. But Π ensures weak unlinkability, as shown next.

Consider a trace ta where ta_i and ta_j carry tag annotations corresponding to distinct sessions of the same identity i_0. Note that, since all tag outputs are randomised with a nonce, and since k remains secret, the frame is equivalent to one where each handle maps to a fresh name. In particular, renaming identities in the frame keeps it statically equivalent. To conclude that weak unlinkability holds, we choose to map all identities to i_0 except the one associated to ta_i. Doing so we preserve the executability of reader tests, and thus outputs, because there are only two distinct identities involved.

This example is of course extremely artificial; we kept it simple to focus on the key idea. It is also obviously outside the class of protocols that our method can meaningfully analyse: specifically, the notions of honest interaction and well-authentication are not adequate here. We believe it should be possible to adapt our counter-example into one that involves only two-party interactions, but that is not the point of this discussion.

Importantly, we claim that Example 16 shows not only a gap between two definitions, but also an actual attack on unlinkability that is not captured by weak unlinkability. More precisely, it violates the requirement that the attacker should not learn anything about the relationship between sessions (here, the reader learns that at least two sessions among three correspond to the same identity) although it does not contradict the simpler informal notion of unlinkability that only requires that the attacker cannot tell if two sessions are related. In our view, the generalised form of unlinkability is important, since any information leakage could be used to track people. With our example protocol, if the attacker controlled one honest tag, it would be able to tell when two sessions correspond to the same identity.

B. Comparison with unlinkability of [21]

Consider a protocol Π in the sense of [21] where the tag and reader roles are given by processes T and R, with identity parameters $\overline{k}$ and session parameters $\overline{n}_T \sqcup \overline{n}_R$ such that $fn(T) \subseteq \overline{n}_T \sqcup \overline{k}$ and $fn(R) \subseteq \overline{n}_R \sqcup \overline{k}$. It does not matter which of T and R is the initiator role. Given the class of protocols that we consider in this paper, we restrict our attention to the case where tag sessions are played sequentially and reader sessions are played concurrently. We also assume that identity parameters are shared by the tag and reader roles, *i.e.* we are in the so-called *shared case* of [21] where $\overline{k} \cap fn(T) \neq \varnothing$ and $\overline{k} \cap fn(R) \neq \varnothing$.

From Π we construct a protocol according to our definition. Relying on some obvious syntactic sugar for translating a tuple into a sequence and conversely, we define $\Pi' = (\overline{k}, \overline{k}, T', R')$ with the role processes defined as follows:

- $T' = \mathsf{get}(r, \overline{y}).T\{\overline{k} \mapsto \overline{y}\}$;

- R' = lookup $\overline{y}$ such that $x = \mathsf{ok}, \overline{y'} = \overline{y}$ in $R\{\overline{k} \mapsto \overline{y}\}$.

In other words, each identity $\overline{k}$ is stored both in the corresponding tag state and in the global reader database. These values will only be read but never modified. The tag process reads the names $\overline{k}$ from r. After that, it will behave like T. The reader process R' chooses non-deterministically an identity $\overline{k}$ and becomes the process R where the identity parameters are instantiated accordingly. Note that $fn(T') \subseteq \overline{n}_T$ and $fn(R') \subseteq \overline{n}_R$.

In [21], the unlinkability of Π is defined as the trace equivalence $\mathcal{M} \approx \mathcal{S}$ where:

$$\begin{aligned} \mathcal{M} &= ! \ \mathsf{new}\, \overline{k}.\big((\mathsf{i}\,\mathsf{new}\, \overline{n}_T.T) \mid (!\mathsf{new}\, \overline{n}_R.R)\big) \\ \mathcal{S} &= ! \ \mathsf{new}\, \overline{k}.\big((\mathsf{new}\, \overline{n}_T.T) \mid (\mathsf{new}\, \overline{n}_R.R)\big) \end{aligned}$$

Proposition 2: In the shared case, unlinkability of [21] implies our notion of unlinkability: $\mathcal{M} \approx \mathcal{S}$ implies $\mathcal{M}_{\Pi'} \approx \mathcal{S}_{\Pi'}$.

Proof. We first observe that $\mathcal{M} \approx \mathcal{M}_{\Pi'}$. Indeed, the only difference between the two processes is in the way one obtains instances of tag and reader processes: in $\mathcal{M}$ they are directly spawned by possibly replicating the outermost ! and creating a new identity $\overline{k}$, then creating an instance of the tag or reader process with the desired session parameters; in $\mathcal{M}_{\Pi'}$ the processes T' or R' are first instanciated with session parameters, before reading their identity parameters $\overline{k}$ from the memory cell or database. Thus, for any trace (ta, ϕ) of $\mathcal{M}$ we have a trace (ta', ϕ) of $\mathcal{M}_{\Pi'}$ with $\mathsf{obs}(\mathsf{ta}) = \mathsf{obs}(\mathsf{ta}')$: it suffices to add τ actions corresponding to memory accesses, and modify τ actions corresponding to replications and sequences. Similarly, for any trace (ta', ϕ) of $\mathcal{M}_{\Pi'}$ there exists a trace (ta, ϕ) of $\mathcal{M}$ with $\mathsf{obs}(\mathsf{ta}) = \mathsf{obs}(\mathsf{ta}')$.

We also have $\mathcal{S} \sqsubseteq \mathcal{S}_{\Pi'}$, for the same reasons as above. The converse does not hold, because traces of $\mathcal{S}_{\Pi'}$ may feature several reader sessions with the same identity parameters, which cannot be mimicked by $\mathcal{S}$.

Hence, assuming $\mathcal{M} \approx \mathcal{S}$ we have $\mathcal{M}_{\Pi'} \approx \mathcal{M} \approx \mathcal{S} \sqsubseteq \mathcal{S}_{\Pi'}$. The converse inclusion $\mathcal{S}_{\Pi'} \sqsubseteq \mathcal{M}_{\Pi'}$ is immediate, thus our notion of unlinkability holds. □

APPENDIX B
PROOFS OF OUR MAIN RESULT

We provide in this appendix the proof of our main result (Theorem 1) that we recall here.

Theorem 1: Let Π be a protocol together with an idealisation operator $\mathrm{ideal}(.)$ for it, such that:

- Π ensures frame opacity w.r.t. to ideal;
- Π is well-authenticating; and
- Π ensures that no desynchronisation occurs.

We have that Π satisfies unlinkability.

The goal is to prove that $\mathcal{M}_\Pi \sqsubseteq \mathcal{S}_\Pi$, since the other direction is trivial. The argument follows the same general structure as for the theorem of [21]; we recall it and highlight key differences.

We fix a protocol $\Pi = (init_T, init_R, T, R)$, and we denote $\overline{k} = fn(init_T) \cup fn(init_R)$. For any execution of $\mathcal{M}_\Pi$, we will show that there exists an indistinguishable execution of $\mathcal{S}_\Pi$. Essentially, we will apply a renaming of agents involved in the given execution of $\mathcal{M}_\Pi$ to map this multiple-session execution to a single-session execution. We will then show that this renaming preserves the executability and that the frames before and after the renaming are statically indistinguishable.

A. Abstraction of configurations

In order to facilitate the development of the proof, we will work with an abstraction of configurations that we call *ground configurations*. We will associate to any execution of $\mathcal{M}_\Pi$ and $\mathcal{S}_\Pi$ a ground configuration containing all involved agents already correctly instantiated. These ground configurations are obtained from *well-formed* sequences of annotations, which we define below.

Definition 14: A sequence S_a of annotations is *well-formed* if the following conditions hold.

- In all annotations $T(r, \overline{n}_T)$ and $R(\overline{n}_R)$, the session parameters $\overline{n}_T$ and $\overline{n}_R$ are names from $\mathcal{N}$ such that $\overline{n}_T \cap \overline{n}_R = \varnothing$ and r is a reference from $\mathcal{R}$.
- Two different annotations never share a session parameter.

Definition 15: We say that a well-formed sequence S_a of annotations is *single-session* if for all tag annotations $T(r, \overline{n}_{T1})$ and $T(r, \overline{n}_{T2})$ in S_a sharing the same reference r, we have that $\overline{n}_{T1} = \overline{n}_{T2}$.

We lift those definitions to annotated traces, by only keeping the annotations appearing in the trace. We then have that an annotated trace obtained from an execution of $\mathcal{M}_\Pi$ (resp. $\mathcal{S}_\Pi$) is well-formed (resp. well-formed and single-session).

To define ground configurations, we make use of the following notations, where S_a is a well-formed sequence of annotations:

- $\mathrm{param}_R(S_a) = \{\overline{n}_R \mid R(\overline{n}_R) \in S_a\}$ is the set of session parameters for the reader role in S_a;
- $\mathrm{ref}(S_a) = \{r \mid T(r, \overline{n}_T) \in S_a\}$ is the set of references appearing in the annotations of the tag role in S_a;
- for each $r \in \mathrm{ref}(S_a)$, $\mathrm{param}_T^{\mathrm{seq}}(S_a, r)$ is the sequence of session parameters for the tag role associated to a reference r in S_a, without repetition and in order of first occurrence in S_a;
- for each $r \in \mathrm{ref}(S_a)$, we assume a sequence of names $\overline{k}_r$ having the same length as $\overline{k}$ and such that $\overline{k}_r$ does not contain any name occurring in $\mathrm{param}_R(S_a)$, $\mathrm{param}_T^{\mathrm{seq}}(S_a, r')$ or $\overline{k}_{r'}$, for any $r' \neq r$.

Given a sequence $s = (e_i)_{i \in [1;n]}$, we write $\coprod_{e \in s} P(e)$ for $P(e_1); (P(e_2); (\ldots; P(e_n)\ldots))$.

Definition 16: Let S_a be a well-formed sequence of annotations. The *ground configuration* associated to S_a, denoted

by $\mathcal{K}(S_a)$, is the configuration $(\mathcal{P}_T \sqcup \mathcal{P}_R; \varnothing; (\mathcal{S}_T, \mathcal{S}_R))$ where $\mathcal{P}_T$, $\mathcal{P}_R$, $\mathcal{S}_T$ and $\mathcal{S}_R$ are defined as follows:

$$\mathcal{P}_T = \Big\{ \coprod_{\overline{n}'_T \in \mathrm{param}_T^{\mathrm{seq}}(S_a, r')} T\{r \mapsto r', \overline{n}_T \mapsto \overline{n}'_T\}[T(r', \overline{n}'_T)] \;\Big|\; r' \in \mathrm{ref}(S_a) \Big\}$$

$$\mathcal{P}_R = \Big\{ R\{\overline{n}_R \mapsto \overline{n}'_R\}[R(\overline{n}'_R)] \;\Big|\; \overline{n}'_R \in \mathrm{param}_R(S_a) \Big\}$$

$$\mathcal{S}_T = \Big\{ r \mapsto init_T\{\overline{k} \mapsto \overline{k}_r\} \;\Big|\; r \in \mathrm{ref}(S_a) \Big\}$$

$$\mathcal{S}_R = \Big\{ init_R\{\overline{k} \mapsto \overline{k}_r\} \;\Big|\; r \in \mathrm{ref}(S_a) \Big\}$$

We lift these definitions to annotated traces as before, allowing us to define a ground configuration $\mathcal{K}(\mathsf{ta})$ from a well-formed annotated trace ta. We can then establish precise connections between the executions of $\mathcal{M}_\Pi$ or $\mathcal{S}_\Pi$ and those of the associated ground configurations.

Proposition 3: Let ta be a well-formed annotated trace.

(1) If $\mathcal{M}_\Pi \xRightarrow{\mathsf{ta}} K$ (resp. $\mathcal{S}_\Pi \xRightarrow{\mathsf{ta}} K$), then $\mathcal{K}(\mathsf{ta}) \xRightarrow{\mathsf{ta}} K'$ for some K' such that $\phi(K) \sim \phi(K')$.
(2) If $\mathcal{K}(\mathsf{ta}) \xRightarrow{\mathsf{ta}} K$, then $\mathcal{M}_\Pi \xRightarrow{\mathsf{ta}} K'$ for some K' such that $\phi(K) = \phi(K')$.
(3) If $\mathcal{K}(\mathsf{ta}) \xRightarrow{\mathsf{ta}} K$ and ta is single-session, then $\mathcal{S}_\Pi \xRightarrow{\mathsf{ta}} K'$ 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.

(1) The operator $\mathcal{K}(\cdot)$ mimicks how $\mathcal{M}_\Pi$ and $\mathcal{S}_\Pi$ create agents: if an agent is available in the execution of $\mathcal{M}_\Pi$ or $\mathcal{S}_\Pi$ then the same agent is also available in the execution of $\mathcal{K}(\mathsf{ta})$. In particular, the SET and INSERT rules used to initialise the store in $\mathcal{M}_\Pi$ or $\mathcal{S}_\Pi$ before the creation of agents are replaced in $\mathcal{K}(\mathsf{ta})$ by the initialisation of the store before starting the execution, up to a renaming: if $init_T\{k \mapsto k'\}$ and $init_R\{k \mapsto k'\}$ are used when initializing a tag $T(r, \overline{n})$, then $\mathcal{K}(\mathsf{ta})$ will have $\mathcal{S}_T(r) = init_T\{k \mapsto \overline{k}_r\}$ and $init_R\{k \mapsto \overline{k}_r\} \in \mathcal{S}_R$. This alpha-equivalence will be maintained not only between the stores but also the annotated processes of our two executions. In particular the resulting frames will be alpha-equivalent, hence statically equivalent.
(2) This item relies on the fact that the memory cells and the database are initialised with fresh alpha-renamings of the initialisation parameters of the protocol: the same parameters can be chosen when creating agents in $\mathcal{M}_\Pi$.
(3) This item is similar to item (2), adding the single-session hypothesis to justify that we never need to create two tag sessions with the same identity.
(4) This item relies on the fact that extra processes and references from ta_2 are unused when executing ta_1. □

B. Renaming

As mentionned before, we will use a renaming of annotations to show that, for any execution of $\mathcal{M}_\Pi$, there exists an indistinguishable execution of $\mathcal{S}_\Pi$. We give below a generic definition of such renamings of annotations.

Definition 17: 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_a, the sequence $S_a\rho$ is well-formed and single session;
- ρ is role-preserving, *i.e.* tag (resp. reader) annotations are mapped to tag (resp. reader) annotations.

We define $\mathsf{ta}\rho$ as the annotated trace obtained from ta by applying ρ to annotations only. We can also define the effect of renaming on ground configurations. To that end, we first define the renaming on names induced by the renaming on annotations.

a) Renaming on names induced by the renaming on annotations: Given a renaming ρ on annotations, we can define a renaming σ (induced by ρ on a particular annotation A) that applies on names and references for this annotation A.

- If $A = T(r, \overline{n}_T)$ and $\rho(A) = T(r', \overline{n}'_T)$ then σ is such that $r\sigma = r'$ and $\overline{n}_T\sigma = \overline{n}'_T$;
- If $A = R(\overline{n}_R)$ and $\rho(A) = R(\overline{n}'_R)$ then σ is such that $\overline{n}_R\sigma = \overline{n}'_R$.

Renaming on names induced by a renaming of annotations may conflict. For example, when $\rho(T(r, \overline{n}_T)) = T(r_1, \overline{n}_T)$ and $\rho(T(r, \overline{n}'_T)) = T(r_2, \overline{n}'_T)$, the two induced renamings map r to r_1 and r_2 respectively.

b) Renaming on ground configurations: Consider a ground configuration $K = (\mathcal{P}; \varnothing; (\mathcal{S}_T, \mathcal{S}_R))$ where $\mathcal{P} = \{\coprod_j P_j^i[a_j^i]\}_i$, $\mathcal{S}_T = \{r_k \mapsto {v_T}^k\}_k$ and $\mathcal{S}_R = \{{v_R}^l\}_l$. We note $\mathrm{ref}(K, \rho)$ the set of all references appearing in the renamed annotations $\rho(a_j^i)$. We also assume vectors $\overline{k}_r$ as in Section B-A. We define $K\rho = (\mathcal{P}'; \varnothing; (\mathcal{S}'_T, \mathcal{S}'_R))$ where:

- $\mathcal{P}' = \{\coprod_j P_j^i \sigma_j^i[\rho(a_j^i)]\}_i$ with σ_j^i the renaming induced by ρ on a_j^i;
- $\mathcal{S}'_T = \{r \mapsto init_T\{\overline{k} \mapsto \overline{k}_r\} \mid r \in \mathrm{ref}(K, \rho)\}$;
- $\mathcal{S}'_R = \{init_R\{\overline{k} \mapsto \overline{k}_r\} \mid r \in \mathrm{ref}(K, \rho)\}$.

Proposition 4: 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 3: extra processes and references that are added to the ground configuration $\mathcal{K}(\mathsf{ta}_1)\rho$ when considering $\mathcal{K}(\mathsf{ta}_1.\mathsf{ta}_2)\rho$ are unused in the execution of $\mathsf{ta}_1\rho$. □

Proposition 5: Let ta be a well-formed annotated trace and ρ be a renaming of annotations. If $\mathcal{K}(\mathsf{ta})\rho \xrightarrow{\mathsf{ta}\rho} (\mathcal{P}; \phi; \mathcal{S})$ then $\mathcal{K}(\mathsf{ta}\rho) \xRightarrow{\mathsf{ta}\rho} (\mathcal{P}'; \phi; \mathcal{S})$.

Proof. The sets of annotations involved in $\mathcal{K}(\mathsf{ta}\rho)$ and $\mathcal{K}(\mathsf{ta})\rho$ are the same, thus we have:

- when considering $\mathcal{K}(\mathsf{ta})\rho$ and $\mathcal{K}(\mathsf{ta}\rho)$ as multisets of processes without sequence, we obtain the same multisets;
- $\mathsf{store}(\mathcal{K}(\mathsf{ta})\rho) = \mathsf{store}(\mathcal{K}(\mathsf{ta}\rho)) = \mathcal{S}$, by definition of the renaming on ground configurations.

Moreover, as we consider a renaming ρ that maps a sequence of annotations to a *single session* sequence of annotations,

there is no sequence in $\mathcal{K}(\mathsf{ta}\rho)$ (because we cannot have two tag processes sharing the same references).

As a result, when a process $P[\rho(a)]$ is available in the execution starting from $\mathcal{K}(\mathsf{ta})\rho$ then this process is also available in the execution starting from $\mathcal{K}(\mathsf{ta}\rho)$ because no process is blocked by a sequence in this execution. □

C. Frame idealisations

Before establishing here the main result regarding frame opacity, we first generalise the notion of frame idealisation to consider more complex idealisation operators.

We will obtain idealised frames by replacing each output message by a message built from session parameters *and* from the idealisations of previously inputted messages. The precise construction will depend on the specific output, identified by its label ℓ.

We consider two disjoint and countable subsets of variables: *input variables* $\mathcal{X}^{\mathrm{i}} = \{x_1^{\mathrm{i}}, x_2^{\mathrm{i}}, \ldots\} \subseteq \mathcal{X}$ and *name variables* $\mathcal{X}^{\mathrm{n}} = \{x_1^{\mathrm{n}}, x_2^{\mathrm{n}}, \ldots\} \subseteq \mathcal{X}$. Variable x_j^{i} will refer to the j-th input received by a given agent. Variables in $\mathcal{X}^{\mathrm{n}}$ will refer to fresh names used in the idealised frame. We also assume a fixed but arbitrary *idealisation operator* $\mathrm{ideal}(\cdot) : \mathcal{L} \mapsto \mathcal{T}(\Sigma, \mathcal{X}^{\mathrm{i}} \cup \mathcal{X}^{\mathrm{n}})$. We assume that *input variables* appearing in the term $\mathrm{ideal}(\ell)$ are in $\{x_1^{\mathrm{i}}, \ldots, x_k^{\mathrm{i}}\}$ where k is the number of inputs preceeding the output labelled ℓ.

Thus, compared to the simplified presentation of Section IV-B, idealisation operators may now refer to input variables, and rely on arbitrary function symbols (not only constructor symbols). The notion of idealised frame is thus a bit more complex and is now defined inductively.

Definition 18: Let $\mathrm{fr} : \mathcal{A} \times \mathcal{X}^{\mathrm{n}} \mapsto \mathcal{N}$ be an injective function assigning names to annotation 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:

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

This definition is in line with Definition 18 where the application of σ^{i} and $\Downarrow$ were useless. Note also that this notion is not necessarily well-defined since, for some cases, $\mathrm{ideal}(\ell)\sigma^{\mathrm{i}}\sigma^{\mathrm{n}}$ might not evaluate to a message. Due to that, the definition of frame opacity has to be slightly adapted.

Definition 19: The protocol Π ensures *frame opacity* w.r.t. the idealisation operator ideal if for any execution $(\mathcal{M}_\Pi; \varnothing; \mathcal{S}) \xrightarrow{\mathsf{ta}} (Q; \phi; \mathcal{S}')$ we have that $\Phi_{\mathrm{ideal}}(\mathsf{ta})$ is well-defined and $\Phi_{\mathrm{ideal}}(\mathsf{ta}) \sim \phi$.

The strength of frame opacity lies in the fact that idealised frames do not depend on the names and states of the agents involved in an execution. This allows to show that, when a renaming preserves executability, the resulting frame is statically equivalent to the original one.

Proposition 6: Assume that Π satisfies frame opacity. Let ρ be a renaming of annotations and ta be a well-formed annotated trace. If we have both $\mathcal{K}(\mathsf{ta}) \overset{\mathsf{ta}}{\Longrightarrow} (\mathcal{P}_1; \phi_1; \mathcal{S}_1)$ and $\mathcal{K}(\mathsf{ta}\rho) \overset{\mathsf{ta}\rho}{\Longrightarrow} (\mathcal{P}_2; \phi_2; \mathcal{S}_2)$, then $\phi_1 \sim \phi_2$.

Proof. As in [21], the idealised frames $\Phi_{\mathrm{ideal}}(\mathsf{ta})$ and $\Phi_{\mathrm{ideal}}(\mathsf{ta}\rho)$ are equal up to an α-renaming and thus they are statically equivalent. □

D. Renaming does not break executability

We now show a key lemma, stating that for any renaming, if $\mathcal{K}(\mathsf{ta})$ can execute an annotated trace ta then $\mathcal{K}(\mathsf{ta})\rho$ can execute the renamed annotated trace $\mathsf{ta}\rho$. We rely for this on the notion of bi-process, i.e. processes in which terms are replaced by bi-terms, denoted $\mathsf{choice}[t_1, t_2]$. Such a bi-process can evolve if both sides of the bi-process agree on the outcome of a rule. For instance, in case of evaluation of a let instruction, both sides evaluate positively (resp. negatively). Otherwise, the bi-process is blocked. Given a bi-process B, we denote $\mathsf{fst}(B)$ the process obtained from B by replacing any occurrence of $\mathsf{choice}[t_1, t_2]$ by t_1. The process $\mathsf{snd}(B)$ is defined in a similar way.

Lemma 1: We assume that Π ensures frame opacity, well-authentication and no desynchronisation. Let ta be a well-formed annotated trace such that $\mathcal{K}(\mathsf{ta}) \xrightarrow{\mathsf{ta}} (\mathcal{P}; \phi; \mathcal{S})$. Let ρ be a renaming according to Definition 17. We have that $\mathcal{K}(\mathsf{ta})\rho \xrightarrow{\mathsf{ta}\rho} (\mathcal{Q}; \psi; \mathcal{Z})$ for some ψ such that $\phi \sim \psi$.

Proof. Processes in the ground configurations $\mathcal{K}(\mathsf{ta})$ and $\mathcal{K}(\mathsf{ta})\rho$ having the same shape (they differ only by their terms), we introduce bi-configurations whith the following syntax and semantics.

We note a bi-configuration $B = (\mathcal{P}_B; \phi_B; \mathsf{choice}[\mathcal{S}_1, \mathcal{S}_2])$ where $\mathcal{P}_B$ is a set of bi-processes and ϕ_B is a bi-frame, *i.e.* processes and frames where terms are bi-terms of the form $\mathsf{choice}[t_1, t_2]$. The notation $\mathsf{choice}[\mathcal{S}_1, \mathcal{S}_2]$ means that stores are not bi-stores and evolve independently: the bi-matching concerns only processes and frames.

Outputs of the bi-process B obtained from $\mathcal{K}(\mathsf{ta})$ and $\mathcal{K}(\mathsf{ta})\rho$ are decorated with handles from $\mathcal{W}$ used to store the corresponding output bi-messages (one handle for each output of the bi-process, all handles distinct).

We also associate a vector $\overline{R}$ of terms in $\mathcal{T}(\Sigma_{\mathsf{pub}}, \mathcal{W} \cup \mathcal{X})$ to each safe conditional of the protocol. We note that $\overline{R}$ may contain variables from $\mathcal{X}$ corresponding to inputs performed before the conditional. These variables cannot refer to values from the store by definition of a safe conditional. These variables will be instantiated by ground terms during the execution.

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

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

(b) any bi-conditional let $\overline{z} = \mathsf{choice}[\overline{t}_l, \overline{t}_r]$ in B_P 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 $w \notin \mathsf{dom}(\phi_0)$ or a term in $\mathcal{T}(\Sigma_{\mathsf{pub}}, \mathsf{dom}(\phi_0))$. Moreover, we have that $\overline{C}[R_1\phi_0^+\Downarrow, \ldots, R_k\phi_0^+\Downarrow] = \bar{t}_l$ and $\overline{C}[R_1\psi_0^+\Downarrow, \ldots, R_k\psi_0^+\Downarrow] = \bar{t}_r$ 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. $\mathsf{snd}(B_0)$), *i.e.* extended with outputs that have not yet occured;

(c) $\phi_0 \sim \psi_0$.

We prove this by induction on the length of the prefix ta_0 of ta. In case ta_0 is empty, the three conditions trivially holds. In particular, (b) is a consequence of our definition of being a safe conditional. We now consider the case where ta_0 is of the form $\mathsf{ta}_0'.\alpha$.

We note a the annotation associated to the action α produced by a process also annotated a in $\mathcal{P}_0$. By construction of a ground configuration, the action α can be an input, an output, a conditional (let or lookup), a τ action produced by the rules SET or GET, or a τ_{abort} action.

We have that $\mathcal{K}(\mathsf{ta}) \xrightarrow{\mathsf{ta}_0'} (\mathcal{P}_0'; \phi_0'; \mathcal{S}_0') \xrightarrow{\alpha} (\mathcal{P}_0; \phi_0; \mathcal{S}_0)$ is a prefix of the execution $\mathcal{K}(\mathsf{ta}) \xrightarrow{\mathsf{ta}} (\mathcal{P}; \phi; \mathcal{S})$ and thus by induction hypothesis, we know that there exists an execution $\mathcal{K}(\mathsf{ta})\rho \xrightarrow{\mathsf{ta}_0'\rho} (\mathcal{Q}_0'; \psi_0'; \mathcal{Z}_0')$ satisfying the three invariants mentionned above. In particular, we have that

$$B \xrightarrow{\mathsf{choice}[\mathsf{ta}_0', \mathsf{ta}_0'\rho]} B_0'$$

for some bi-process B_0' such that $\mathsf{fst}(B_0') = (\mathcal{P}_0'; \phi_0'; \mathcal{S}_0')$, and $\mathsf{snd}(B_0') = (\mathcal{Q}_0'; \psi_0'; \mathcal{Z}_0')$. We have $(\mathcal{P}_0'; \phi_0'; \mathcal{S}_0') \xrightarrow{\alpha[a]} (\mathcal{P}_0; \phi_0; \mathcal{S}_0)$. We have to prove that there exists an execution $(\mathcal{Q}_0'; \psi_0'; \mathcal{Z}_0') \xrightarrow{\alpha[\rho(a)]} (\mathcal{Q}_0; \psi_0; \mathcal{Z}_0)$ that preserves our invariants.

- Case where α is an output. As $(\mathcal{P}_0'; \phi_0'; \mathcal{S}_0')$ and $(\mathcal{Q}_0'; \psi_0'; \mathcal{Z}_0')$ form a bi-process, namely B_0', $(\mathcal{Q}_0'; \psi_0'; \mathcal{Z}_0')$ can obviously perform $\alpha[\rho(a)]$ on the same channel and with the same handle, so $(\mathcal{Q}_0'; \psi_0'; \mathcal{Z}_0') \xrightarrow{\alpha[\rho(a)]} (\mathcal{Q}_0; \psi_0; \mathcal{Z}_0)$, and the bi-process B_0 obtained from B_0' by executing this output satisfies (a). Condition (b) can be established as in [21]. Note that even if the frame ϕ_0' (resp. ψ_0') has evolved in ϕ_0 (resp, ψ_0), we have that $\phi_0^+ = {\phi_0'}^+$ (resp. $\psi_0^+ = {\psi_0'}^+$). It remains to establish our invariant (c). By applying Proposition 3 (item 4) on $\mathcal{K}(\mathsf{ta}) \xrightarrow{\mathsf{ta}_0'.\alpha[a]} (\mathcal{P}_0; \phi_0; \mathcal{S}_0)$, we obtain:

$$\mathcal{K}(\mathsf{ta}_0'.\alpha[a]) \xrightarrow{\mathsf{ta}_0'.\alpha[a]} (\mathcal{P}_0''; \phi_0; \mathcal{S}_0'')$$

Proposition 4 on $\mathcal{K}(\mathsf{ta})\rho \xrightarrow{\mathsf{ta}_0'\rho.\alpha[\rho(a)]} (\mathcal{Q}_0; \psi_0; \mathcal{Z}_0)$ gives:

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

We conclude $\phi_0 \sim \psi_0$ using Proposition 5 and Proposition 6 and relying on the fact that frame opacity holds

- Case where α is a conditional (let or lookup). We note τ_x (resp. τ_y) the action produced by evaluating the conditional of the annotation a (resp. $\rho(a)$). We have $(\mathcal{P}_0'; \phi_0'; \mathcal{S}_0') \xrightarrow{\tau_x[a]} (\mathcal{P}_0; \phi_0'; \mathcal{S}_0)$ (frames are the same because evaluating a conditional does not change the frames). $\mathcal{Q}_0'$ can perform $\tau_y[\rho(a)]$, so there exists an execution $(\mathcal{Q}_0'; \psi_0'; \mathcal{Z}_0') \xrightarrow{\alpha[\rho(a)]} (\mathcal{Q}_0; \psi_0'; \mathcal{Z}_0)$.
We first prove that the outcome of the test is the same, *i.e.* $\tau_x = \tau_{\mathsf{then}}$ if, and only if, $\tau_y = \tau_{\mathsf{then}}$. In case the conditional is a safe one, as done in [21], the term $\overline{R}$ labelling the conditional on both side is a recipe and it allows us to ensure that the conditional will be evaluated in the same way on both side of the bi-process. Thus, we consider the case where the conditional is unsafe. As the case of an output, applying Proposition 3.(4) and Proposition 4, we have that:

$$\mathcal{K}(\mathsf{ta}_0'.\tau_x[a]) \xrightarrow{\mathsf{ta}_0'.\tau_x[a]} (\mathcal{P}_0''; \phi_0'; \mathcal{S}_0'')$$

$$\mathcal{K}(\mathsf{ta}_0'.\tau_y[a])\rho \xrightarrow{\mathsf{ta}_0'\rho.\tau_y[\rho(a)]} (\mathcal{Q}_0''; \psi_0'; \mathcal{Z}_0'')$$

Let assume that $\tau_x = \tau_{\mathsf{then}}$. Using Proposition 3 and the well-authentication condition, we have $\mathcal{M}_\Pi \xrightarrow{\mathsf{ta}_0'.\tau_{\mathsf{then}}[a]} (\mathcal{P}_0''; \phi_0'; \mathcal{S}_0'')$ and there exists a' such that the annotations a and a' have an honest interaction in $(\mathsf{ta}_0', \phi_0')$. Thus, $\rho(a)$ and $\rho(a')$ also have an honest interaction in $(\mathsf{ta}_0'\rho, \psi_0')$ because:
  - $\mathsf{obs}(\mathsf{ta}_0')$ and $\mathsf{obs}(\mathsf{ta}_0'\rho)$ are equal up to annotations;
  - for any action $\mathsf{in}(\_, R_i)$ occurring in ta_0', we have $R_i\phi_0'\Downarrow w_i\phi_0'$, we deduce $R_i\psi_0'\Downarrow w_i\psi_0'$ using invariant (c).

As a result, we have $\mathcal{M}_\Pi \xrightarrow{\mathsf{ta}_0'\rho.\tau_y[\rho(a)]} (\mathcal{Q}_0''; \psi_0'; \mathcal{Z}_0'')$ and there exists $\rho(a')$ such that the annotations $\rho(a)$ and $\rho(a')$ have an honest interaction in $(\mathsf{ta}_0'\rho, \psi_0')$.
We conclude $\tau_y = \tau_{\mathsf{then}}$ using the no desynchronisation condition[2]. The other direction, *i.e.*

$$\tau_y = \tau_{\mathsf{then}} \Rightarrow \tau_x = \tau_{\mathsf{then}}$$

can be established in a similar way. Note that in case the conditional is made by a lookup construct (reader role), updating the database does not break the invariants.
- Case where $\alpha = \mathsf{in}(c, R_{in})$ is an input. As in the case of an output, this action can be mimicked on the right hand side of the bi-process. The terms used to decorate safe conditionals have to be updated by replacing the variable x occurring in input by R_{in}, thus the invariant will be

[2]In [21] the argument is very different here. The corresponding lemma imposes a further condition on ρ ensuring that, at this point in the proof, $\rho(a)$ and $\rho(a')$ share the same identity. Hence, their interaction is (an alpha-renaming of) the (assumed) honest execution of the protocol, where all conditions evaluate positively. The assumption on ρ does not make sense due to our generic readers, and the fact that tags and readers sharing the same identity must have *the* honest interaction becomes incorrect in presence of state: our no desynchronisation condition avoids these problems while providing a meaningful condition on protocols.

preserved. Frames are still in static equivalence as they remain unchanged.

- Case where α is a τ action produced by the rule GET or SET. This case is easy since stores of the bi-process may evolve separately. No relation has to be preserved between the elements stored on the left and those stored on the right. In case of the rule GET, by construction of a ground configuration, any memory-cell lookup is well-defined because the reference is already initialised in the store, so this action can be mimicked on the left.
- Case where α is a τ_{abort} action produced by the rule ABORT. This case is easy since the bi-process ensures that $\mathsf{fst}(B_0')$ and $\mathsf{snd}(B_0')$ have the same shape. In case ABORT can be applied on $\mathsf{fst}(B_0')$, the rule can thus be also applied on $\mathsf{snd}(B_0')$. Our invariants are satisfied.

This concludes the proof. □

E. Proof of Theorem 1

Theorem 1: Let Π be a protocol together with an idealisation operator $\mathrm{ideal}(.)$ for it, such that:

- Π ensures frame opacity w.r.t. to ideal;
- Π is well-authenticating; and
- Π ensures that no desynchronisation occurs.

We have that Π satisfies unlinkability.

Proof. The inclusion $\mathcal{S}_\Pi \sqsubseteq \mathcal{M}_\Pi$ is trivial. It remains to establish that $\mathcal{M}_\Pi \sqsubseteq \mathcal{S}_\Pi$.

Consider an execution $\mathcal{M}_\Pi \xrightarrow{\mathsf{ta}} (\mathcal{P};\phi;\mathcal{S})$. Proposition 3 gives us that there exists a well-formed annotated trace $\mathsf{ta}' \overset{\tau}{=} \mathsf{ta}$ such that $\mathcal{K}(\mathsf{ta}) \xrightarrow{\mathsf{ta}'} (\mathcal{P}';\phi;\mathcal{S}')$.

Let ρ be an arbitrary renaming of annotations. Existence is guaranteed, as the following construction shows. Let $ref_\rho : \mathcal{N}^* \mapsto \mathcal{R}$ be an injective function that associates to any sequence of names $\overline{n}$ a reference in $\mathcal{R}$. We then define ρ as follows:

$$\begin{aligned} T(r,\overline{n}_T) &\mapsto T(ref_\rho(\overline{n}_T),\overline{n}_T) \\ R(\overline{n}_R) &\mapsto R(\overline{n}_R) \end{aligned}$$

The mapping ρ is a renaming of annotations because:

- ρ is role-preserving;
- ρ is single-session because all references are disjoint since ref_ρ is injective ;
- ρ preserves the well-formedness of any sequence of annotations.

By Lemma 1, we have that $\mathcal{K}(\mathsf{ta})\rho \xrightarrow{\mathsf{ta}'\rho} (\mathcal{Q};\psi;\mathcal{Z})$ for some frame ψ such that $\psi \sim \phi$. By Proposition 5 we deduce that $\mathcal{K}(\mathsf{ta}\rho) \xRightarrow{\mathsf{ta}'\rho} (\mathcal{Q};\psi;\mathcal{Z})$, and thus $\mathcal{K}(\mathsf{ta}\rho) \xRightarrow{\mathsf{ta}\rho} (\mathcal{Q};\psi;\mathcal{Z})$. Since $\mathsf{ta}\rho$ is single-session, Proposition 3 implies that $(\mathcal{S}_\Pi;\varnothing;\varnothing) \xRightarrow{\mathsf{ta}\rho} (\mathcal{Q}';\psi;\mathcal{Z}')$. We thus conclude that $\mathcal{M}_\Pi \sqsubseteq \mathcal{S}_\Pi$. □

APPENDIX C
CASE STUDIES

We present in Figures 7 to 16 the descriptions of the protocols studied in Section VI.

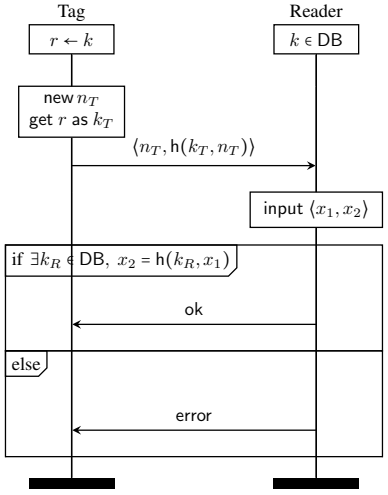

Fig. 7. Basic Hash

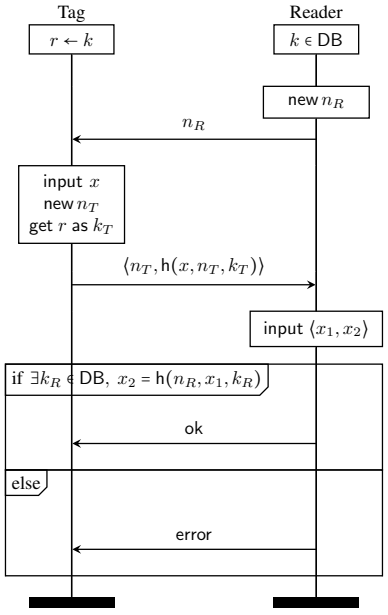

Fig. 8. Hash-Lock

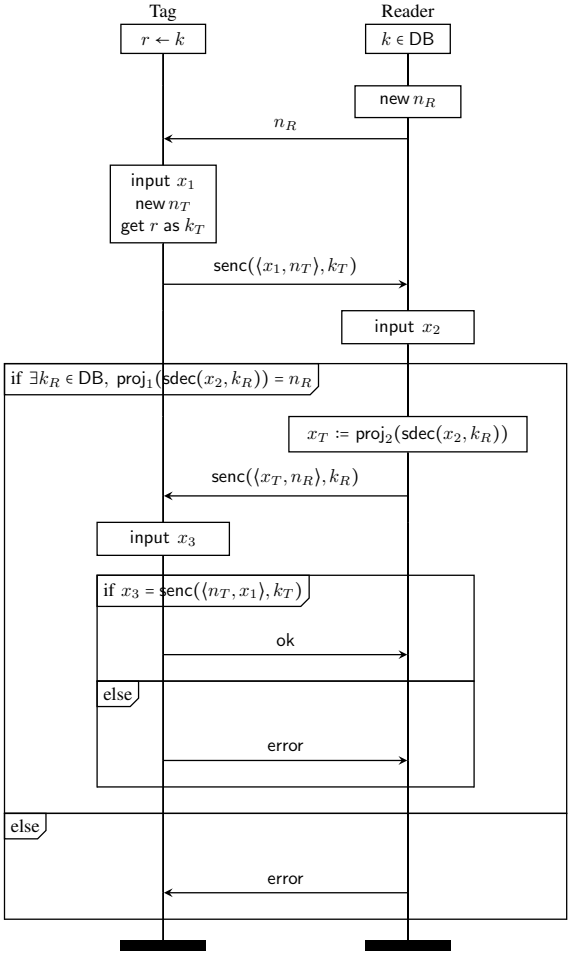

Fig. 9. Feldhofer

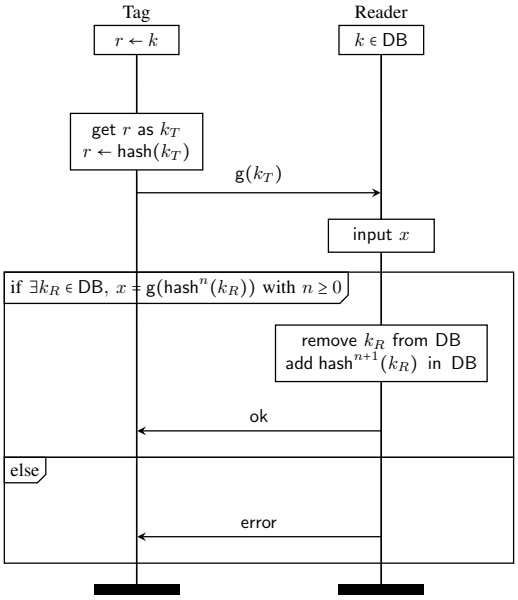

Fig. 10. OSK v1

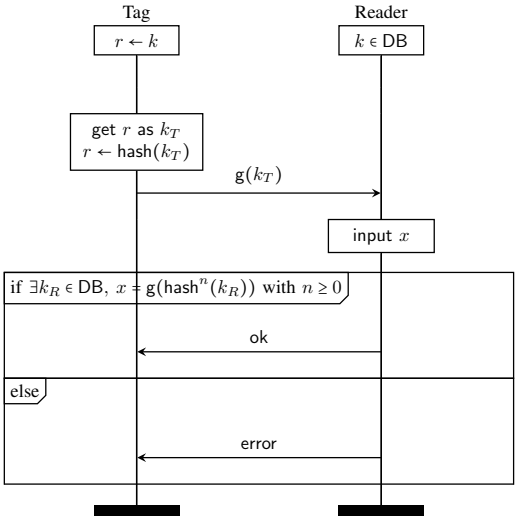

Fig. 11. OSK v2

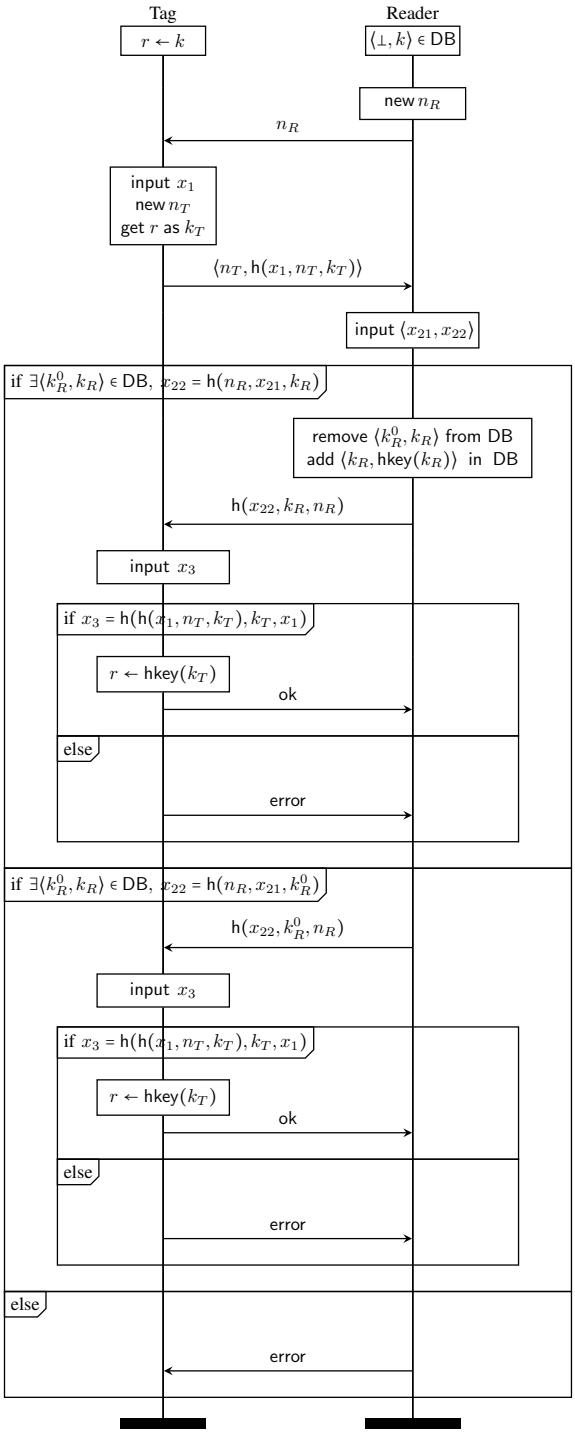

Fig. 12. LAK (pairs)

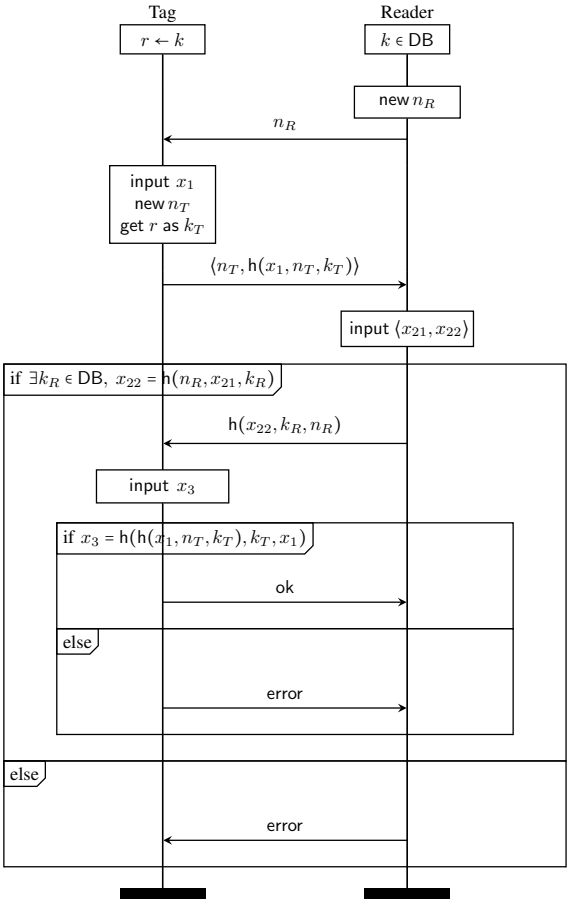

Fig. 13. LAK (pairs, fix v1)

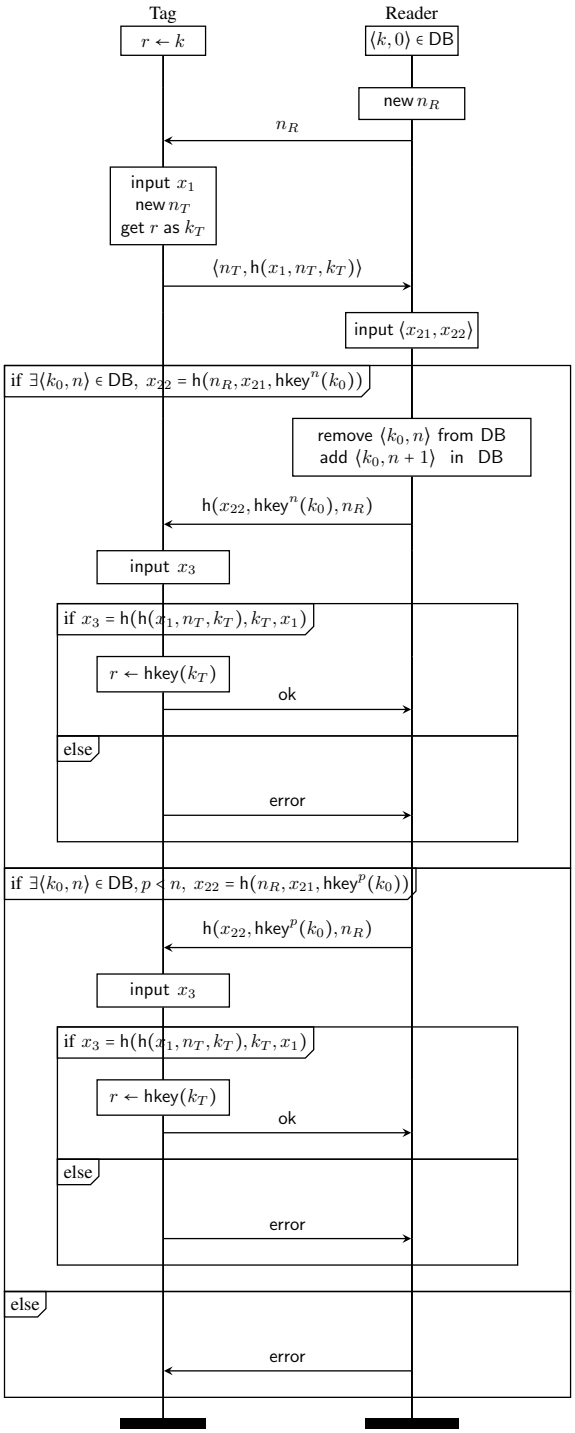

Fig. 14. LAK (pairs, fix v2)

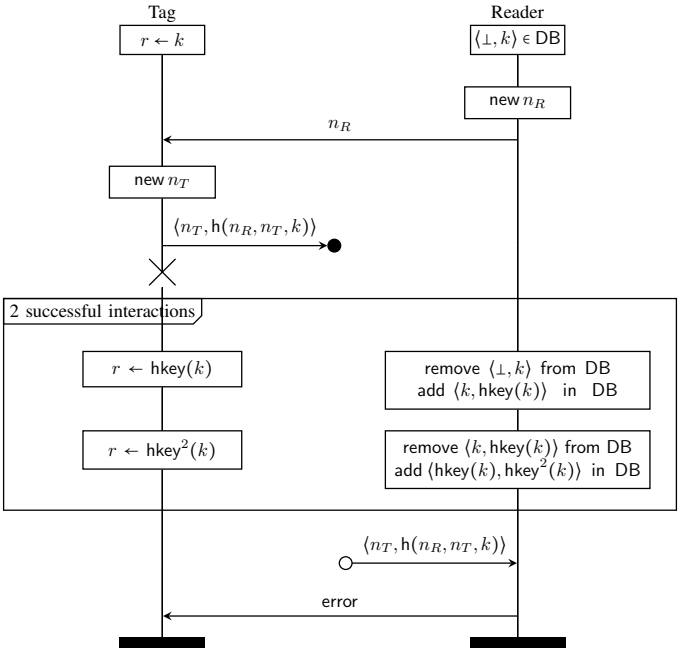

Fig. 15. Linkability attack for LAK (pairs)

This scenario is an attack for our condition no desynchronisation and also for unlinkability: by replaying a tag's ouput, an attacker can know if this tag has successfully interacted at most twice with the reader. This scenario cannot be mimicked in a system where each tag can play only one session.

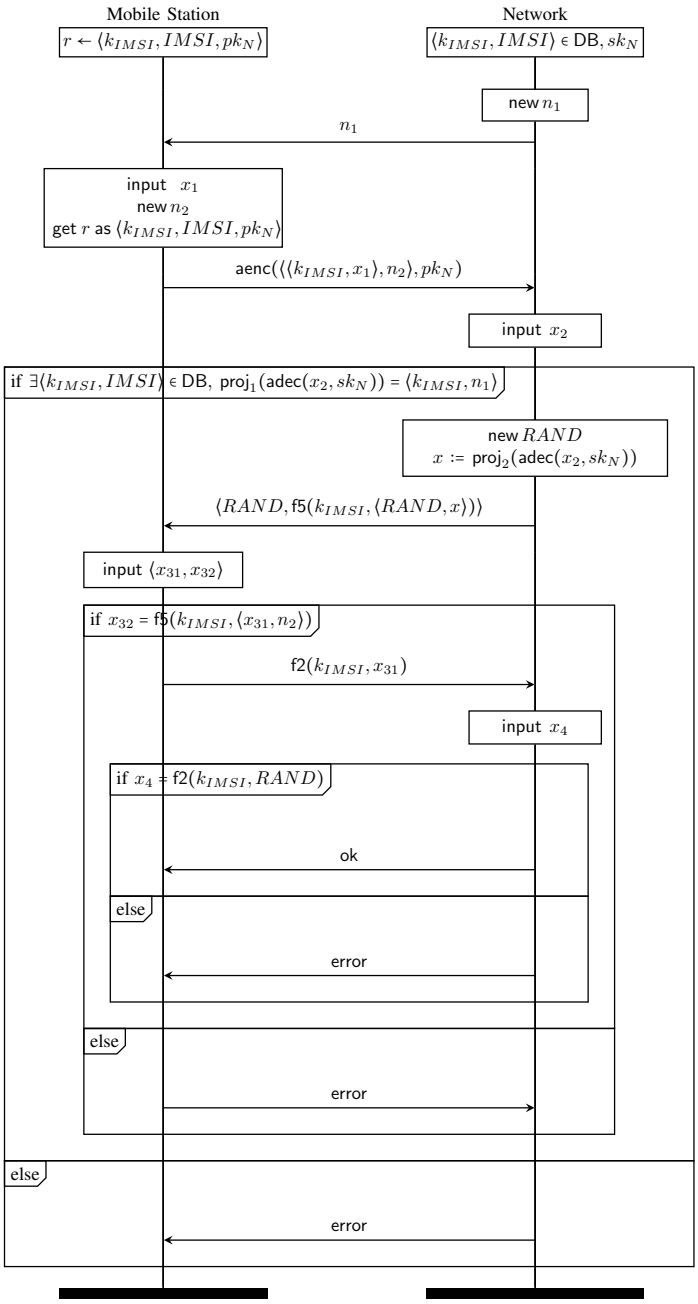

Fig. 16. AKA (simplified)

Here, sk_N represents the network unique private key and pk_N is the corresponding public key, *i.e.* $pk_N = \mathsf{pk}(sk_N)$. The symbols aenc and adec model asymmetric encryption and decryption, with the following rewriting rule: $\mathsf{adec}(\mathsf{aenc}(m, \mathsf{pk}(sk)), sk) = m$.