A Method for Verifying Privacy-Type Properties: The Unbounded Case

Lucca Hirschi, David Baelde and Stéphanie Delaune
LSV, CNRS & ENS Cachan, Université Paris-Saclay, France

***Abstract*—In this paper, we consider the problem of verifying anonymity and unlinkability in the symbolic model, where protocols are represented as processes in a variant of the applied pi calculus notably used in the ProVerif tool. Existing tools and techniques do not allow one to verify directly these properties, expressed as behavioral equivalences. We propose a different approach: we design two conditions on protocols which are sufficient to ensure anonymity and unlinkability, and which can then be effectively checked automatically using ProVerif. Our two conditions correspond to two broad classes of attacks on unlinkability, corresponding to data and control-flow leaks.**

This theoretical result is general enough to apply to a wide class of protocols. In particular, we apply our techniques to provide the first formal security proof of the BAC protocol (e-passport). Our work has also lead to the discovery of new attacks, including one on the LAK protocol (RFID authentication) which was previously claimed to be unlinkable (in a weak sense) and one on the PACE protocol (e-passport).

I. Introduction

Security protocols aim at securing communications over various types of insecure networks (*e.g.* web, wireless devices) where dishonest users may listen to communications and interfere with them. A *secure communication* has a different meaning depending on the underlying application. It ranges from the confidentiality of data (medical files, secret keys, etc.) to, *e.g.* verifiability in electronic voting systems. Another example of a security notion is privacy. In this paper, we focus on two privacy-related properties, namely unlinkability (sometimes called untraceability), and anonymity. These two notions are informally defined in the ISO/IEC standard 15408 [1] as follows:

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

Both are critical for instance for Radio-Frequency Identification Devices (RFID) and are thus extensively studied in that context (see, *e.g.* [2] for a survey of attacks on this type of protocols), but they are obviously not limited to it.

One extremely successful approach when designing and analyzing security protocols is the use of formal verification, *i.e.* the development of rigorous frameworks and techniques to analyze protocols. This approach has notably lead to the discovery of a flaw in the Single-Sign-On protocol used *e.g.* by Google Apps. It has been shown that a malicious application could very easily access to any other application (*e.g.* Gmail or Google Calendar) of their users [3]. This flaw has been found when analyzing the protocol using formal methods, abstracting messages by a term algebra and using the Avantssar validation platform. Another example is a flaw on vote-privacy discovered during the formal and manual analysis of an electronic voting protocol [4]. All these results have been obtained using *formal symbolic models*, where most of the cryptographic details are ignored using abstract structures. The techniques used in symbolic models have become mature and several tools for protocol verification are nowadays available, *e.g.* the Avantssar platform [5], the Tamarin prover [6], and the ProVerif tool [7].

Unfortunately, most of these results and tools focus on trace properties, that is, statements that something bad never occurs on any execution trace of a protocol. Secrecy and authentication are typical examples of trace properties: a data remains confidential if, for any execution, the attacker is not able to produce the data. But privacy properties like unlinkability and anonymity typically cannot be defined as trace properties. Instead, they are usually defined as the fact that an observer cannot distinguish between two situations, which requires a notion of behavioral equivalence. Roughly, two protocols are equivalent if an attacker cannot observe any difference between them. Based on such a notion of equivalence, several definitions of privacy-type properties have been proposed (*e.g.* [8], [9] for unlinkability, and [10], [11] for vote-privacy). In this paper, we consider the well-established definitions of strong unlinkability and anonymity as defined in [8]. They have notably been used to establish privacy for various protocols either by hand or using ad hoc encodings (*e.g.* eHealth protocol [12], mobile telephony [13], [14]). We provide a brief comparison with alternative definitions in Section III-B.

Considering an unbounded number of sessions, the problem of deciding whether a protocol satisfies an equivalence property is undecidable even for a very limited fragment of protocols (see, *e.g.* [15]). Bounding the number of sessions suffices to retrieve decidability for standard primitives (see, *e.g.* [16], [17]). However, analyzing a protocol for a fixed (often low) number of sessions does not allow to prove security. Moreover, in the case of equivalence properties, existing tools scale badly and can only analyze protocols for a very limited number of sessions, typically 2 or 3. Another

This work has been partially supported by the project JCJC VIP ANR-11-JS02-006 and the ANR project Sequoia ANR-14-CE28-0030-01.

approach consists in implementing a procedure that is not guaranteed to terminate. This is in particular the case of ProVerif, a well-established tool for checking security of protocols. ProVerif is able to check a strong notion of equivalence (called diff-equivalence) between processes that share the same structure. Despite recent improvements on diff-equivalence checking [18] intended to prove unlinkability of the BAC protocol (used in e-passport), ProVerif still cannot be used off-the-shelf to establish unlinkability properties, and therefore cannot conclude on the case studies presented in Section VII. Recently, similar approaches have been implemented in two other tools, namely Tamarin [19] and Maude–NPA [20]. They are based on a notion of diff-equivalence, and therefore suffer from the same drawbacks.

In this paper, we follow a different approach. We aim at proposing sufficient conditions that can be automatically checked, and that imply unlinkability and anonymity of the protocol under study. This approach is in the same spirit as the one presented in [9]. However, [9] only considers a very restricted class of protocols (single-step protocols that only use hash functions), while we target more complex protocols. The success of our solution will be measured by confronting it to many case studies.

Our contribution: We identify a large class of 2-party protocols (simple else branches, arbitrary cryptographic primitives) and we devise two conditions that imply unlinkability and anonymity for an unbounded number of sessions. We show how these two conditions can be automatically checked using the ProVerif tool, and we provide tool support for that. We have analyzed several protocols, among them the Basic Access Control (BAC) protocol as well as the Password Authenticated Connection Establishment (PACE) protocol that are both used in e-passports. We notably establish the first proof of unlinkability for the BAC protocol followed by the Passive Authentication (PA) and Active Authentication (AA) protocols. We also report on an attack that we found on the PACE protocol, and another one that we found on the LAK protocol whereas it is claimed untraceable in [2]. It happens that our conditions are rather tight: we provide an attack every time one of them is not satisfied.

We now give an intuitive overview of these two conditions. In order to do this, assume that we want to design a mutual authentication protocol between a tag T and a reader R based on symmetric encryption, and we want this protocol to be unlinkable. We note $\{m\}_k$ the symmetric encryption of a message m with a key k and we assume that k is a symmetric key shared between T and R.

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

$$\begin{array}{lll} 1. & R \to T: & n_R \\ 2. & T \to R: & \{n_R\}_k \end{array}$$

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

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

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

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

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

Our main theorem states that these two conditions, frame opacity and well-authentication, are actually sufficient to ensure both unlinkability and anonymity. This theorem is of interest as our two conditions are fundamentally simpler than the targeted properties: frame opacity can be expressed using diff-equivalence and well-authentication is a trace property. In fact, they are both in the scope of existing automatic verification tools like ProVerif.

Outline: In Section II, we present our model inspired from the applied pi calculus as well as the notion of trace equivalence. We then define in Section III the class of protocols and the formal definitions of unlinkability and anonymity we study in this paper. Our two conditions (frame opacity and well-authentication) and our main theorem are presented in Section IV. Section V is dedicated to the proof of that result. Finally, we discuss how to mechanize the verification

of our conditions in Section VI and present our case studies in Section VII, before concluding in Section VIII.

II. Model

We shall model security protocols using a process algebra inspired from the applied pi calculus [21]. More specifically, we consider the calculus of Blanchet *et al.* [22], which is used in the ProVerif tool. Participants are modeled as processes, and the communication between them is modeled by means of the exchange of messages that are represented by a term algebra.

A. Term algebra

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

Given a signature $\mathcal{F}$, and a set of initial data A, we denote by $\mathcal{T}(\mathcal{F}, \mathsf{A})$ the set of terms built from elements of A by applying function symbols in $\mathcal{F}$. Terms of $\mathcal{T}(\Sigma_c, \mathcal{N} \cup \mathcal{X})$ will be called *constructor terms*. We denote $\mathrm{vars}(u)$ the set of variables that occur in a term u. A *message* is a constructor term u that is *ground*, *i.e.* such that $\mathrm{vars}(u) = \varnothing$. We denote by $\overline{x}$, $\overline{n}$, $\overline{u}$ a (possibly empty) sequence of variables, names, and terms respectively. The application of a substitution σ to a term u is written $u\sigma$, and we denote $\mathrm{dom}(\sigma)$ its *domain*. The *positions* of a term are defined as usual.

Example 1: Consider the signature

$$\Sigma = \{\mathsf{enc},\ \mathsf{dec},\ \langle\,\rangle,\ \pi_1,\ \pi_2,\ \oplus,\ 0,\ \mathsf{eq},\ \mathsf{ok}\}.$$

The symbols enc and dec of arity 2 represent symmetric encryption and decryption. Pairing is modeled using $\langle\ \rangle$ of arity 2, whereas projection functions are denoted π_1 and π_2, both of arity 1. The function symbol $\oplus$ of arity 2 and the constant 0 are used to model the exclusive or operator. Finally, we consider the symbol eq of arity 2 to model equality test, as well as the constant symbol ok. This signature is split into two parts: $\Sigma_c = \{\mathsf{enc}, \langle\,\rangle, \oplus, 0, \mathsf{ok}\}$, and $\Sigma_d = \{\mathsf{dec}, \pi_1, \pi_2, \mathsf{eq}\}$.

As in the process calculus presented in [22], constructor terms are subject to an equational theory; this has proved very useful for modeling algebraic properties of cryptographic primitives (see *e.g.* [23] for a survey). Formally, we consider a congruence $=_{\mathsf{E}}$ on $\mathcal{T}(\Sigma_c, \mathcal{N} \cup \mathcal{X})$, generated from a set of equations E over $\mathcal{T}(\Sigma_c, \mathcal{X})$. Thus, $=_{\mathsf{E}}$ is closed under substitutions and under bijective renaming. We finally assume that there exist u, v such that $u \neq_{\mathsf{E}} v$.

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

$$\begin{array}{rclcrcl} x \oplus 0 & = & x & & (x \oplus y) \oplus z & = & x \oplus (y \oplus z) \\ x \oplus x & = & 0 & & (x \oplus y) & = & (y \oplus x) \end{array}$$

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

We may also want to give a meaning to destructor symbols. For this, we consider the notion of *computation relation*.

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

- $n \Downarrow n$ for any $n \in \mathcal{N}$;
- $\mathsf{f}(t_1, \ldots, t_k) \Downarrow \mathsf{f}(u_1, \ldots, u_k)$ for $\mathsf{f} \in \Sigma_c$ of arity k and $t_i \Downarrow u_i$ for all $1 \leq i \leq k$,
- if $t \Downarrow u$ then $t\rho \Downarrow u\rho$ for any bijective $\rho : \mathcal{N} \to \mathcal{N}$;
- for any term t, messages u and v, and context t' (*i.e.* a term with a hole) built from Σ and $\mathcal{N}$, if $t \Downarrow u$ and $t'[u] \Downarrow v$ then $t'[t] \Downarrow v$;
- if t' is a context built from Σ and $\mathcal{N}$, and t_1, t_2 are terms such that $t_1 =_{\mathsf{E}} t_2$ and $t'[t_1] \Downarrow u_1$ for some u_1, then $t'[t_2] \Downarrow u_2$ for some u_2 such that $u_1 =_{\mathsf{E}} u_2$.

The relation $\Downarrow$ associates, to any ground term t, at most one message up to the equational theory E. When no such message exists, we say that the *computation fails*; this is noted $t \Downarrow$. As a slight abuse of notation, we may sometimes use directly $t \Downarrow$ as a message, when we know that the computation succeeds and the choice of representative is irrelevant.

A computation relation is often obtained from a *rewriting system*, *i.e.* a set of rewriting rules $\mathsf{g}(u_1, \ldots, u_n) \to u$ where g is a destructor, and $u, u_1, \ldots, u_n \in \mathcal{T}(\Sigma_c, \mathcal{X})$. A ground term t can be rewritten into t' if there is a position p in t and a rewriting rule $\mathsf{g}(u_1, \ldots, u_n) \to u$ such that $t|_p = \mathsf{g}(v_1, \ldots, v_n)$ and $v_1 =_{\mathsf{E}} u_1\theta, \ldots, v_n =_{\mathsf{E}} u_n\theta$ for some substitution θ, and $t' = t[u\theta]_p$ (*i.e.* t in which the sub-term at position p has been replaced by $u\theta$). Moreover, we assume that $u_1\theta, \ldots, u_n\theta$ as well as $u\theta$ are messages.

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

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

This rewriting system is convergent modulo the equational theory E given in Example 2, and therefore induces a computation relation as defined in Definition 1. For instance, we have that $\mathsf{dec}(\mathsf{enc}(c, a \oplus b), b \oplus a) \Downarrow c$, whereas $\mathsf{dec}(\mathsf{enc}(c, a \oplus b), b) \Downarrow$, and $\mathsf{dec}(a, b) \oplus \mathsf{dec}(a, b) \Downarrow$.

Our generic notion of computation relation gives us enough flexibility to define a destructor symbol neq, and consider that $\mathsf{neq}(u, v) \Downarrow \mathsf{ok}$ if, and only if, u and v can be reduced to messages that are not equal modulo E.

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

B. Process algebra

We consider a set $\mathcal{C}$ of channel names that are assumed to be public. Protocols are modeled through processes using the grammar in Figure 1.

$$\begin{array}{rcll} P, Q & := & 0 & \text{null} \\ & | & \mathtt{in}(c, x).P & \text{input} \\ & | & \mathtt{out}(c, u).P & \text{output} \\ & | & \mathsf{let}\ \overline{x} = \overline{v}\ \mathtt{in}\ P\ \mathsf{else}\ Q & \text{evaluation} \\ & | & P \mid Q & \text{parallel} \\ & | & !P & \text{replication} \\ & | & \nu n.P & \text{restriction} \end{array}$$

where $c \in \mathcal{C}$, $x \in \mathcal{X}$, $n \in \mathcal{N}$, $u \in \mathcal{T}(\Sigma_c, \mathcal{N} \cup \mathcal{X})$ is a constructor term, $\overline{x}$ (resp. $\overline{v}$) is a sequence of variables in $\mathcal{X}$ (resp. terms in $\mathcal{T}(\Sigma, \mathcal{N} \cup \mathcal{X})$) both of the same length.

Fig. 1. Syntax of processes

Most of the constructions are rather standard. We may note the special construct $\mathsf{let}\ \overline{x} = \overline{v}\ \mathtt{in}\ P\ \mathsf{else}\ Q$ that combines several standard constructions, allowing to write computations and conditionals compactly. Such a process tries to evaluate the sequence of terms $\overline{v}$ and in case of success, *i.e.* when $\overline{v} \Downarrow \overline{u}$ for some messages $\overline{u}$, the process P in which $\overline{x}$ are replaced by $\overline{u}$ is executed; otherwise the process Q is executed. The goal of this construct is to avoid nested let instructions to be able to define our class of protocols in a simple way later on. Note also that the let instruction together with the eq theory as defined in Example 3 can encode the usual conditional construction. Indeed, "$\mathsf{let}\ x = \mathsf{eq}(u, v)\ \mathtt{in}\ P\ \mathsf{else}\ Q$" will execute P only if the computation succeeds on $\mathsf{eq}(u, v)$, that is only if $u \Downarrow u'$, $v \Downarrow v'$, and $u' =_{\mathsf{E}} v'$ for some messages u' and v'.

For brevity, we sometimes omit "else 0" and null processes after outputs. We write $fv(P)$ for the set of *free variables* of P, *i.e.* the set of variables that are not in the scope of an input or a let construct. A process P is ground if $fv(P) = \varnothing$.

Example 4: We consider the RFID protocol due to Feldhofer *et al.* as described in [24] and which can be presented using Alice & Bob notation as follows:

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

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

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

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

$$\begin{array}{rcl} P_I & := & \mathtt{out}(c_I, n_I).\mathtt{in}(c_I, x_1). \\ & & \mathsf{let}\ x_2, x_3 = \mathsf{eq}(n_I, \pi_1(u)), \pi_2(u)\ \mathtt{in} \\ & & \mathtt{out}(c_I, \mathsf{enc}(\langle x_3, n_I \rangle, k)) \\ P_R & := & \mathtt{in}(c_R, y_1).\mathtt{out}(c_R, \mathsf{enc}(\langle y_1, n_R \rangle, k)).\mathtt{in}(c_R, y_2). \\ & & \mathsf{let}\ y_3 = \mathsf{eq}(y_2, \mathsf{enc}(\langle n_R, y_1 \rangle, k))\ \mathtt{in}\ 0 \end{array}$$

C. Semantics

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

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

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

The operational semantics of a process is given by the relation $\xrightarrow{\alpha}$ defined in Figure 2. The rules are quite standard and correspond to the intuitive meaning of the syntax given in the previous section. The first rule allows the attacker to send on channel c a message as soon as it is the result of a computation done by applying public function symbols on messages that are in his current knowledge. The second rule corresponds to the output of a term: the corresponding term is added to the frame of the current configuration, which means that the attacker gains access to it. The third and fourth rules correspond to the evaluation of a sequence of terms $\overline{v} = v_1, \ldots, v_n$; if this succeeds, *i.e.* if there exist messages $u_1, \ldots u_n$ such that $v_1 \Downarrow u_1, \ldots v_n \Downarrow u_n$ then variables $\overline{x}$ are bound to those messages, and P is executed; otherwise the process will continue with Q. The three remaining rules allow one to execute a restriction, unfold a replication, and split a parallel composition. The two first rules are the only observable actions. However, for reasons that will become clear later on, we make a distinction when a process evolves using LET or LET-FAIL. The relation $\xrightarrow{\alpha_1 \ldots \alpha_n}$ between configurations (where $\alpha_1 \ldots \alpha_n$ is a sequence of actions) is defined as the transitive closure of $\xrightarrow{\alpha}$.

Example 5: Continuing Example 4. We have that:

$$P_{\mathsf{Fh}} \xrightarrow{\mathsf{tr}} (\varnothing; \phi_0)$$

where tr and ϕ_0 are as follows, for fresh names k', n'_I and n'_R:

$$\mathsf{tr} = \begin{cases} \tau.\tau.\tau.\tau.\mathtt{out}(c_I, w_1).\mathtt{in}(c_R, w_1).\mathtt{out}(c_R, w_2) \\ \quad \mathtt{in}(c_I, w_2).\tau_{\mathsf{then}}.\mathtt{out}(c_I, w_3).\mathtt{in}(c_R, w_3).\tau_{\mathsf{then}} \end{cases}$$

$$\phi_0 = \{w_1 \mapsto n'_I,\ w_2 \mapsto \mathsf{enc}(\langle n'_I, n'_R \rangle, k'), \\ w_3 \mapsto \mathsf{enc}(\langle n'_R, n'_I \rangle, k')\}.$$

This execution corresponds to a normal execution of one session of the protocol.

D. Trace equivalence

We are concerned with trace equivalence, which is commonly used [9], [25] to express many privacy-type properties such as anonymity, unlinkability, strong secrecy, etc. Intuitively, two configurations are trace equivalent if an attacker

IN	$(\mathtt{in}(c,x).P \cup \mathcal{P};\phi) \xrightarrow{\mathtt{in}(c,R)} (P\{x \mapsto u\} \cup \mathcal{P};\phi)$ where R is a recipe such that $R\phi \Downarrow u$ for some message u	
OUT	$(\mathtt{out}(c,u).P \cup \mathcal{P};\phi) \xrightarrow{\mathtt{out}(c,w)} (P \cup \mathcal{P};\phi \cup \{w \mapsto u\})$	with w a fresh variable in $\mathcal{W}$
LET	$(\mathsf{let}\ \overline{x} = \overline{v}\ \mathtt{in}\ P\ \mathsf{else}\ Q \cup \mathcal{P};\phi) \xrightarrow{\tau_{\mathsf{then}}} (P\{\overline{x} \mapsto \overline{u}\} \cup \mathcal{P};\phi)$	when $\overline{v} \Downarrow \overline{u}$ for some $\overline{u}$
LET-FAIL	$(\mathsf{let}\ \overline{x} = \overline{v}\ \mathtt{in}\ P\ \mathsf{else}\ Q \cup \mathcal{P};\phi) \xrightarrow{\tau_{\mathsf{else}}} (Q \cup \mathcal{P};\phi)$	when $v_i \not\Downarrow$ for some $v_i \in \overline{v}$
NEW	$(\nu n.P \cup \mathcal{P};\phi) \xrightarrow{\tau} (P \cup \mathcal{P};\phi)$	where n is a fresh name from $\mathcal{N}$
REPL	$(!P \cup \mathcal{P};\phi) \xrightarrow{\tau} (P \cup !P \cup \mathcal{P};\phi)$	
PAR	$(\{P_1 \mid P_2\} \cup \mathcal{P};\phi) \xrightarrow{\tau} (\{P_1, P_2\} \cup \mathcal{P};\phi)$	

Fig. 2. Semantics for processes

cannot tell whether he is interacting with one or the other. Before defining formally this notion, we first introduce a notion of equivalence between frames, called *static equivalence*.

Definition 2: A frame ϕ is *statically included* in ϕ' when $\mathrm{dom}(\phi) = \mathrm{dom}(\phi')$, and

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

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

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

Example 6: Consider the frame ϕ_0 as given in Example 5, we have that $\phi_0 \sqcup \{w_4 \mapsto k'\} \not\sim \phi_0 \sqcup \{w_4 \mapsto k''\}$. Indeed, the attacker may observe that the computation $R = \mathsf{dec}(w_2, w_4)$ succeeds in $\phi \sqcup \{w_4 \mapsto k'\}$ but fails in $\phi \sqcup \{w_4 \mapsto k''\}$.

Then, *trace equivalence* is the active counterpart of static equivalence taking into account the fact that the attacker may interfere during the execution of the process in order to distinguish between the two situations.

Given a configuration $K = (\mathcal{P};\phi)$, we define $\mathsf{trace}(K)$:

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

We define $\mathsf{obs}(\mathsf{tr})$ to be the subsequence of tr obtained by erasing all the τ actions (*i.e.* $\tau, \tau_{\mathsf{then}}, \tau_{\mathsf{else}}$).

Definition 3: Let K and K' be two configurations. We say that K is *trace included* in K', written $K \sqsubseteq K'$, when, for any $(\mathsf{tr},\phi) \in \mathsf{trace}(K)$ there exists $(\mathsf{tr}',\phi') \in \mathsf{trace}(K')$ such that $\mathsf{obs}(\mathsf{tr}) = \mathsf{obs}(\mathsf{tr}')$ and $\phi \sim \phi'$. They are in *trace equivalence*, written $K \approx K'$, when $K \sqsubseteq K'$ and $K' \sqsubseteq K$.

Example 7: We may be interested in checking whether $K = (!P_{\mathsf{Fh}};\varnothing)$ and $K' = (!\nu k.(!\nu n_I.P_I \mid !\nu n_R.P_R);\varnothing)$ are in trace equivalence. Intuitively, this equivalence models the fact that P_{Fh} is unlinkable: each session of the protocol appears to an attacker as if it has been initiated by a different tag, since a given tag can perform at most one session in the idealized scenario K. This equivalence actually holds. It is non-trivial, and cannot be established using existing verification tools such as ProVerif or Tamarin. The technique developed in this paper will notably allow one to establish it automatically.

III. Our class of protocols and properties

We aim to propose sufficient conditions to ensure unlinkability and anonymity for a generic class of 2-party protocols. In this section, we define formally the class of protocols and the security properties we are interested in.

A. A generic class of 2- party protocols

As already mentioned, we consider 2-party protocols that are therefore made of two roles called the initiator and responder role respectively. We assume a set $\mathcal{L}$ of labels that will be used to name output actions in these roles, allowing us to identify outputs that are performed by a same syntactic output action. These labels have no effect on the semantics.

Definition 4: An *initiator role* is a ground process obtained using the following grammar:

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

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

$$\begin{array}{rcl} P_R & ::= & 0 \\ & \mid & \mathtt{in}(c,y).\mathsf{let}\ \overline{x} = \overline{v}\ \mathtt{in}\ P_I\ \mathsf{else}\ 0 \\ & \mid & \mathtt{in}(c,y).\mathsf{let}\ \overline{x} = \overline{v}\ \mathtt{in}\ P_I\ \mathsf{else}\ \ell : \mathtt{out}(c',u') \end{array}$$

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

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

Example 8: Continuing our running example, P_I (resp. P_R) as defined in Example 4 is an initiator (resp. responder) role, up to the addition of trivial conditionals and distinct labels ℓ_1, ℓ_2, and ℓ_3 to decorate output actions.

Then, a protocol consists of an initiator role and a responder role that can interact together. This is formally stated through the notion of *honest trace*.

Definition 5: A trace tr (*i.e.* a sequence of actions) is *honest* for a frame ϕ if $\tau_{\mathsf{else}} \notin \mathsf{tr}$ and $\mathsf{obs}(\mathsf{tr})$ is of the form

$$\mathtt{out}(\_, w_0).\mathtt{in}(\_, R_0).\mathtt{out}(\_, w_1).\mathtt{in}(\_, R_1).\ldots$$

for arbitrary channel names, and such that $R_i\phi \Downarrow_{=\mathsf{E}} w_i\phi \Downarrow$ for any action $\mathtt{in}(\_, R_i)$ occurring in tr.

An honest trace is a trace in which the attacker does not really interfere, and that allows the execution to progress without going into an else branch that intuitively correspond to a way to abort the protocol.

Now, among the names that occur in the roles, we need to distinguish those that correspond to long-term data *e.g.* keys (called *identity names*) from others that are freshly generated at each session (called *session names*). We also need to introduce the notion of *public messages*. A message u is *public* if $u =_{\mathsf{E}} v$ for some $v \in \mathcal{T}(\Sigma_c \cap \Sigma_{\mathsf{pub}}, \varnothing)$. Intuitively, a message is public if it is equal modulo E to a term that is built using public symbols only.

Definition 6: A protocol Π is a tuple $(\overline{k}, \overline{n}_I, \overline{n}_R, \mathcal{I}, \mathcal{R})$ where $\overline{k}$, $\overline{n}_I$, $\overline{n}_R$ are three disjoint sets of names, $\mathcal{I}$ (resp. $\mathcal{R}$) is an initiator (resp. responder) role such that $fn(\mathcal{I}) \subseteq \overline{k} \sqcup \overline{n}_I$ (resp. $fn(\mathcal{R}) \subseteq \overline{k} \sqcup \overline{n}_R$). Labels of $\mathcal{I}$ and $\mathcal{R}$ must be pairwise distinct. Names $\overline{k}$ (resp. $\overline{n}_I \sqcup \overline{n}_R$) are called *identity names* (resp. *session names*).

Let $P_\Pi = \nu\overline{k}.(\nu\overline{n}_I.\mathcal{I} \mid \nu\overline{n}_R.\mathcal{R})$. We assume that $P_\Pi \xrightarrow{\mathsf{tr}_h} (\varnothing; \phi_h)$ for some frame ϕ_h that does not contain any public message, and some trace tr_h that is honest for ϕ_h.

Example 9: Let $\Pi = (k, n_I, n_R, P_I, P_R)$ with P_I and P_R as defined in Example 4. We have already seen that P_I is an initiator role whereas P_R is a responder role. Let $P_\Pi = \nu k.(\nu n_I.P_I \mid \nu n_R.P_R)$. Let $\mathsf{tr}_h = \mathsf{tr}$, and $\phi_h = \phi_0$ as defined in Example 5. They satisfy the requirements stated in Definition 6, and therefore Π is a protocol according to our definition.

B. Security properties under study

We consider both anonymity and unlinkability as defined in [8]. Before recalling the formal definition of these two notions, we first introduce some useful notation.

Given a protocol Π, as defined above, we denote $\mathcal{M}_\Pi$ the process that represents an arbitrary number of agents that may possibly execute an arbitrary number of sessions, whereas $\mathcal{S}_\Pi$ represents an arbitrary number of agents that can at most execute one session each. Formally, we define:

$$\begin{aligned} \mathcal{M}_\Pi &:= !\nu\overline{k}.(!\nu\overline{n}_I.\mathcal{I} \mid !\nu\overline{n}_R.\mathcal{R}); \text{ and} \\ \mathcal{S}_\Pi &:= !\nu\overline{k}.(\nu\overline{n}_I.\mathcal{I} \mid \nu\overline{n}_R.\mathcal{R}). \end{aligned}$$

a) Unlinkability: Informally, a protocol preserves unlinkability w.r.t. the roles $\mathcal{I}$ and $\mathcal{R}$ if each session of these roles looks to an outside observer as if it has been executed with different identity names. In other words, an ideal version of the protocol with respect to unlinkability, allows the roles $\mathcal{I}$ and $\mathcal{R}$ to be executed at most once for each identity names. An outside observer should then not be able to tell the difference between the original protocol and the ideal version of this protocol as formally stated below.

Definition 7: Let $\Pi = (\overline{k}, \overline{n}_I, \overline{n}_R, \mathcal{I}, \mathcal{R})$ be a protocol. We say that Π preserves *unlinkability* if $\mathcal{M}_\Pi \approx \mathcal{S}_\Pi$.

Although unlinkability of only one role (*e.g.* the tag for RFID protocols) is often considered in the literature, we consider a stronger notion where both roles are treated symmetrically. We believe this is needed to not miss practical attacks (see Sections VII-C,VII-E for a discussion).

b) Anonymity: In order to express anonymity w.r.t. some identities $\overline{id} \subseteq \overline{k}$, we introduce the following process:

$$\mathcal{M}_\Pi^{\mathsf{id}} := \mathcal{M}_\Pi \mid \nu\overline{k}.(!\nu\overline{n}_I.\mathcal{I}_0 \mid !\nu\overline{n}_R.\mathcal{R}_0)$$

where $\mathcal{I}_0 = \mathcal{I}\{\overline{id} \mapsto \overline{\mathsf{id}_0}\}$, $\mathcal{R}_0 = \mathcal{R}\{\overline{id} \mapsto \overline{\mathsf{id}_0}\}$, and $\overline{\mathsf{id}_0}$ are fresh constants from $\Sigma_c \cap \Sigma_{\mathsf{pub}}$ (*i.e.* not used in Π). In this process, in addition to the arbitrary number of agents that may execute an arbitrary number of sessions, there are two agents $\mathcal{I}_0$ and $\mathcal{R}_0$ that have disclosed (part of) their identity $\overline{\mathsf{id}_0}$ to the attacker, and that may also execute an arbitrary number of sessions.

Definition 8: Let $\Pi = (\overline{k}, \overline{n}_I, \overline{n}_R, \mathcal{I}, \mathcal{R})$ be a protocol, $\overline{id} \subseteq \overline{k}$. We say that Π preserves *anonymity w.r.t.* $\overline{id}$ if $\mathcal{M}_\Pi \approx \mathcal{M}_\Pi^{\mathsf{id}}$.

Defined in this way, anonymity ensures that an attacker does not see the difference between the system $\mathcal{M}_\Pi^{\mathsf{id}}$ (in which $\overline{\mathsf{id}_0}$ is present) and the original system $\mathcal{M}_\Pi$ (in which $\overline{\mathsf{id}_0}$ is not present). Since $\overline{\mathsf{id}_0}$ is not present in the system $\mathcal{M}_\Pi$, his anonymity is trivially preserved.

c) Discussion: A flurry of alternative definitions of unlinkability have been proposed in the literature (see, *e.g.* [26], [27] for a comparison). Among the strongest ones, various game-based formulations have been considered, both in the computational and symbolic models. Some of these definitions, unlike strong unlinkability, can be verified directly in ProVerif using diff-equivalence [28]. However, such game-based definitions do not imply strong unlinkability (see Appendix C for a counter-example) which leaves open the problem of automatically verifying it.

IV. Our Approach

We now define our two conditions, namely frame opacity and well-authentication, and our result which states that these conditions are sufficient to ensure unlinkability and anonymity. Before doing that, we shall introduce annotations in the semantics of our processes, in order to ease their analysis. After having stated our conditions and result, we will illustrate that our conditions are realistic on various case studies.

A. Annotations

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

Formally, an *annotation* is of the form $A(\overline{k},\overline{n})$ where $A \in \{I,R\}$. An *annotated action* is either τ or $\alpha[a]$ where α is an action other than τ (possibly τ_{then} or τ_{else}) and a is an annotation. Finally, an *annotated process* is of the form $P[a]$ where P is a role process and a is an annotation.

Given a protocol $\Pi = (\overline{k},\overline{n}_I,\overline{n}_R,\mathcal{I},\mathcal{R})$, consider any execution of $\mathcal{M}^{\mathsf{id}}_\Pi$, $\mathcal{M}_\Pi$ or $\mathcal{S}_\Pi$. In such an execution, τ actions are solely used to instantiate new agents, by unfolding a replication, breaking a parallel and choosing fresh names. Performing these actions results in the creation of agents, that is, instances of $\mathcal{I}$ and $\mathcal{R}$ with fresh names. Actions other than τ (that is, input, output and conditionals) are then only performed by those agents.

This allows us to define an *annotated semantics* for our processes of interest. In that semantics, agents in the multiset of processes are annotated by their identity (*i.e.* identity and session names that have been created for them), and actions other than τ are annotated with the identity of the agent responsible for that action. Traces of the annotated semantics will be denoted by ta. We also assume that labels used to decorate output actions are added into the frame together with the outputted term so that we can refer to them when needed.

Example 10: Considering the protocol of Example 9, process $\mathcal{S}_\Pi$ can essentially perform the execution seen in Example 5. The annotated execution has the trace ta given below, where k', n'_I and n'_R are fresh names, $a_I = I(k',n'_I)$ and $a_R = R(k',n'_R)$:

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

After the initial τ actions, the annotated configuration is

$$(\{\, \mathcal{I}\sigma_I[a_I],\, \mathcal{R}\sigma_R[a_R],\, \mathcal{S}_\Pi \,\};\phi_0).$$

where $\sigma_I = \{k \mapsto k', n_I \mapsto n'_I\}$, and $\sigma_R = \{k \mapsto k', n_R \mapsto n'_R\}$. The structure is preserved for the rest of the execution of ta, with three processes in the multiset (until they become null), two of which remaining annotated with a_I and a_R. The three terms in ϕ_0 are decorated with ℓ_1, ℓ_2 and ℓ_3 respectively.

Note that annotations of the specific agents whose identity contains constants $\overline{\mathsf{id}_0}$ will contain those constants (*i.e.* they are of the form $A(\overline{k},\overline{n})$ with $\overline{\mathsf{id}_0} \subseteq \overline{k}$).

B. Frame opacity

In light of attacks based on leakage from messages where relations between outputted messages are exploited by the attacker to trace an agent, our first condition will basically require that, in any execution, outputs are indistinguishable from pure randomness and therefore do not reveal anything to the attacker. Formally, we define this notion by comparing a frame with an ideal version of it, which is essentially obtained by replacing each message of a frame by a fresh name. However, in order to obtain a reasonable condition, we must make an exception there for constructors which can be inverted (*e.g.* pairs, lists, XML data) which we call *transparent* and are often used in protocols without any inherent risk.

Definition 9: A set of constructors $\Sigma_t \subseteq \Sigma_c \cap \Sigma_{\mathsf{pub}}$ is said to be *transparent* if it satisfies the following conditions:

- for all $\mathsf{f} \in \Sigma_t$ of arity n, and for all $1 \le i \le n$, there exists a recipe $R_i \in \mathcal{T}(\Sigma_{\mathsf{pub}},\{w\})$ such that for any message $u = \mathsf{f}(u_1,\ldots,u_n) \in \mathcal{T}(\Sigma_c,\mathcal{N})$, one has $R_i\{w \mapsto u\} \Downarrow v_i$ for some v_i such that $v_i =_{\mathsf{E}} u_i$;
- symbols of Σ_t do not occur in the equations of E.

In the rest of our theoretical development, we assume an arbitrary transparent set Σ_t. Note that our results hold even if some constructors satisfying the previous two conditions are not part of the chosen Σ_t.

Example 11: In the signature of Example 1, the largest set of transparent constructors is $\{\langle\,\rangle,\ 0,\ \mathsf{ok}\}$.

We now define the idealization of (the observable parts of) messages and frames: we first replace non-transparent subterms by holes (denoted by $\Box$), and then fill-in these holes using distinct fresh names. The technical details of the first step may be found in Appendix A.

Proposition 1: There exists a function

$$[\cdot]^{\mathsf{ideal}} :\ \mathcal{T}(\Sigma_c,\mathcal{N}) \to \mathcal{T}(\Sigma_t,\{\Box\})$$

such that $[u]^{\mathsf{ideal}} = \mathsf{f}([u_1]^{\mathsf{ideal}},\ldots,[u_n]^{\mathsf{ideal}})$ whenever $u =_{\mathsf{E}} \mathsf{f}(u_1,\ldots,u_n)$ for $\mathsf{f} \in \Sigma_t$, and $[u]^{\mathsf{ideal}} = \Box$ otherwise. Furthermore, we have that $[u]^{\mathsf{ideal}} = [v]^{\mathsf{ideal}}$ whenever $u =_{\mathsf{E}} v$.

Definition 10: A *concretization* of $u_t \in \mathcal{T}(\Sigma_t,\{\Box\})$ is any term obtained by replacing each hole of u_t by a fresh nonce. We denote by $\mathsf{inst}(u_t)$ the set of all concretizations of u_t, and by $[u]^{\mathsf{nonce}}$ the set $\mathsf{inst}([u]^{\mathsf{ideal}})$.

Example 12: Let u be $\langle n_P, \mathsf{enc}(\langle \mathsf{ok}, n_P\rangle, k)\rangle$. We have that $[u]^{\mathsf{ideal}} = \langle \Box, \Box\rangle$ and $[u]^{\mathsf{nonce}} = \{\langle n_1,n_2\rangle \mid n_1 \neq n_2 \in \mathcal{N}\}$.

Those definitions are extended to frames in a natural way, with the freshness condition on nonces being understood at the level of frame and not of individual messages. As a result, we immediately have that, for any $u' \in [u]^{\mathsf{nonce}}$ (resp. $\phi' \in [\phi]^{\mathsf{nonce}}$), no nonce appears twice in u' (resp. ϕ'), and therefore for all frames ψ and $\phi_1,\phi_2 \in [\psi]^{\mathsf{nonce}}$, one has $\phi_1 \sim \phi_2$.

We are now ready to state our first condition:

Definition 11: The protocol Π ensures *frame-opacity* if, for any execution $(\mathcal{M}^{\mathsf{id}}_\Pi;\varnothing) \xrightarrow{\mathsf{ta}} (Q;\phi)$, we have that:

1) $\phi \sim \psi$ for some $\psi \in [\phi]^{\mathsf{nonce}}$, and
2) for any w_i, w_j in $\mathrm{dom}(\phi)$ that carry the same label $\ell \in \mathcal{L}$, we have that $[w_i\phi]^{\mathsf{ideal}} = [w_j\phi]^{\mathsf{ideal}}$.

Example 13: Consider the frame ϕ_0 as defined in Example 5. We have that

$$[\phi_0]^{\mathsf{ideal}} = \{w_1 \mapsto \Box, w_2 \mapsto \Box, w_3 \mapsto \Box\}.$$

We have that $\phi_0 \sim \phi$ for any $\phi \in [\phi_0]^{\mathsf{nonce}}$.

However, in case, $n'_R = n'_I$, static equivalence between ϕ_0 and its idealized version $[\phi_0]^{\mathsf{nonce}}$ does not hold, and therefore any protocol that generates such a frame is not frame opaque.

C. Well-authentication

Our second condition will prevent the attacker to obtain some information about agents through the outcome of conditionals. To do so, we will essentially require that conditionals of $\mathcal{I}$ and $\mathcal{R}$ can only be executed successfully in honest, intended interactions. It is unnecessary to impose such a condition on conditionals that never leak any information, which are found in several security protocols. We characterize below a simple class of such conditionals, for which the attacker will always know the outcome of the conditional based on the past interaction.

Definition 12: A conditional $\mathsf{let}\ \overline{x} = \overline{v}\ \mathtt{in}\ P\ \mathsf{else}\ Q$ occurring in $\mathcal{A} \in \{\mathcal{I}, \mathcal{R}\}$ is *safe* if $\overline{v} \in \mathcal{T}(\Sigma_{\mathsf{pub}}, \{x_1, \ldots, x_n\} \cup \{u_1, \ldots, u_n\})$, where the x_i are the variables bound by the previous inputs of that role, and u_i are the messages used in the previous outputs of that role.

Example 14: Consider the process given below:

$$\mathtt{out}(c, u).\mathtt{in}(c, x).\mathsf{let}\ z = \mathsf{neq}(x, u)\ \mathtt{in}\ P\ \mathsf{else}\ Q$$

The conditional is used to ensure that the agent will not accept as input the message he sent at the previous step. Such a conditional is safe according to our definition.

Note that trivial conditionals required by the grammar are safe and will thus not get in the way of our analysis.

We can now formalize the notion of association, which expresses that two agents are having an honest, intended interaction (*i.e.* the attacker essentially did not interfere in their communications). For an annotated trace ta and annotations a and a', we denote by $\mathsf{ta}|_{\mathrm{a},\mathrm{a}'}$ the subsequence of ta that consists of actions of the form $\alpha[\mathrm{a}]$ or $\alpha[\mathrm{a}']$.

Definition 13: Two agents $A_1(\overline{k}_1, \overline{n}_1)$ and $A_2(\overline{k}_2, \overline{n}_2)$ are *associated* in (ta, ϕ) if:

- the agents are *dual*, *i.e.* $A_1 \neq A_2$ and $\overline{k_1} = \overline{k_2}$;
- the interaction $\mathsf{ta}|_{A_1(\overline{k}_1,\overline{n}_1),A_2(\overline{k}_2,\overline{n}_2)}$ is honest for ϕ.

Example 15: Continuing Example 10, the agents $I(k', n'_I)$ and $R(k', n'_R)$ are associated in (ta, ϕ_0).

We can finally state our second condition:

Definition 14: The protocol Π is *well-authenticating* if, for any execution

$$(\mathcal{M}_\Pi^{\mathsf{id}}; \varnothing) \xrightarrow{\mathsf{ta}.\tau_{\mathsf{then}}[A(\overline{k},\overline{n}_1)]} (\mathcal{P}; \phi)$$

either the last action corresponds to a safe conditional, or there exists A' and $\overline{n}_2$ such that *(i)* $A(\overline{k}, \overline{n}_1)$ and $A'(\overline{k}, \overline{n}_2)$ are associated in (ta, ϕ), and *(ii)* $A'(\overline{k}, \overline{n}_2)$ is only associated with $A(\overline{k}, \overline{n}_1)$ in (ta, ϕ).

Intuitively, this condition does not require anything for safe conditional as we already know that they cannot leak new information to the attacker (he already knows their outcome). For unsafe conditionals, condition *(i)* requires that whenever an agent a evaluates them positively (*i.e.* he does not abort the protocol), it must be the case that this agent a is so far having an honest interaction with a dual agent a'. Indeed, as discussed in introduction, it is crucial to avoid such unsafe conditionals to be evaluated positively when the attacker is interfering because this could leak crucial information. Condition *(ii)* is needed to prevent from having executions where an agent is associated to several agents, which would break unlinkability.

D. Soundness w.r.t. unlinkability and anonymity

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

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

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

Before establishing this theorem in the next section, let us comment on its practical impact. We summarize the result of the confrontation of our method to our case studies in Figure 3, focusing on unlinkability. Detailed descriptions of those protocols and discussions are in Section VII. We remark that our conditions have proven to be tight enough for all our case studies: when a condition fails to hold, we could always discover a real attack on unlinkability. Most of the positive results (when unlinkability holds) and all attacks are new. Note that all positive results were established automatically using our tool UKano (which is based on ProVerif).

Protocol	Frame opacity	Well-auth.	Unlinkability
Feldhofer	✓	✓	**safe**
Hash-Lock	✓	✓	**safe**
LAK (stateless)	–	×	attack
Fixed LAK	✓	✓	**safe**
BAC	✓	✓	**safe**
BAC/PA/AA	✓	✓	**safe**
PACE (faillible dec)	–	×	attack
PACE (as in [29])	–	×	attack
PACE	–	×	attack
PACE with tags	✓	✓	**safe**

Fig. 3. Summary of our case studies. We note ✓ for a condition automatically checked using UKano and × when the condition does not hold.

V. Proofs

We provide in this section the proof of Theorem 1. Our main argument consists in showing that, for any execution of $\mathcal{M}_\Pi^{\mathsf{id}}$, there is an indistinguishable execution of $\mathcal{S}_\Pi$.

Instead of working with $\mathcal{M}_\Pi^{\mathsf{id}}$, $\mathcal{M}_\Pi$ and $\mathcal{S}_\Pi$, it will be more convenient to work with *ground configurations* of the protocol under consideration, which are annotated multisets of instances of $\mathcal{I}$ and $\mathcal{R}$. Intuitively, ground configurations correspond to the annotated multisets obtained from $\mathcal{M}_\Pi^{\mathsf{id}}$, $\mathcal{M}_\Pi$ or $\mathcal{S}_\Pi$ by launching a few sessions (performing τ actions corresponding to replication and names creations) and then removing the initial replicated process to keep only the instantiated agents.

We first define *ground configuration annotations* as sets of annotations satisfying the following conditions:

- in all annotations $A(\overline{k},\overline{n})$, *the session parameters* $\overline{n}$ are names and *the identity parameters* $\overline{k}$ are made of names or constants $\overline{\mathtt{id}_0}$;
- no name appears both as identity and session parameter in any two annotations;
- no two annotations share a session parameter;
- two annotations either have the same identity parameters, or do not share any identity parameter at all.

Then, a *ground configuration* is any annotated multiset of the form $\mathcal{P}_\mathcal{I} \sqcup \mathcal{P}_\mathcal{R}$ where

$$\mathcal{P}_\mathcal{I} = \{\, \mathcal{I}\{\overline{k} \mapsto \overline{l}, \overline{n_I} \mapsto \overline{m}\}[I(\overline{l},\overline{m})] \mid I(\overline{l},\overline{m}) \in S \,\}$$

and similarly for $\mathcal{P}_\mathcal{R}$, where S is a ground configuration annotation.

We shall say that a ground configuration $\mathcal{P}$ is *single-session* if there is at most one agent per identity and role (*i.e.* if $A(\overline{k},\overline{n})$ and $A(\overline{k},\overline{m})$ occur in $\mathcal{P}$ then $\overline{n} = \overline{m}$) and $\overline{\mathtt{id}_0}$ does not occur in it. Any ground configuration can be reached from $\mathcal{M}_\Pi^{\mathsf{id}}$; single-session ground configurations are those which can also be obtained from $\mathcal{S}_\Pi$.

We now introduce formally the notion of renaming of agents that we shall use in the proof, before presenting a few key results that will finally allow us to prove our theorem.

Definition 15: A *renaming of agents* (denoted by ρ) is an injective mapping from annotations to annotations which preserves roles (*i.e.* initiator (resp. responder) annotations are mapped to initiator (resp. responder) annotations) such that the image of a ground configuration annotation is still a ground configuration annotation.

If $\mathtt{ta}$ is an annotated trace whose annotations are all in $\mathrm{dom}(\rho)$, we define $\mathtt{ta}\rho$ as the annotated trace obtained from $\mathtt{ta}$ by replacing any annotation a by $\rho(a)$, without changing the actions of the trace.

If $\rho(A(\overline{k},\overline{n})) = A(\overline{k'},\overline{n'})$, the renaming σ induced by ρ on $A(\overline{k},\overline{n})$ is the (injective) mapping such that $\sigma(\overline{k}) = \overline{k'}$ and $\sigma(\overline{n}) = \overline{n'}$. Given a ground configuration $\mathcal{P} = \{\mathcal{A}_i[a_i]\}_i$ whose annotations are in $\mathrm{dom}(\rho)$, we define $\mathcal{P}\rho = \{\mathcal{A}_i\sigma_i[\rho(a_i)]\}_i$ where σ_i is the renaming induced by ρ on a_i.

Note that the renaming on parameters induced by a renaming of agents may conflict: this happens, for example, when $\rho(A(\overline{k},\overline{n})) = A(\overline{k_1},\overline{n})$ and $\rho(A(\overline{k},\overline{m})) = A(\overline{k_2},\overline{m})$. This means, in particular, that we cannot meaningfully define $\phi\rho$ for a frame ϕ. However, given an execution $\mathtt{ta}$ that yields ϕ, each handle $w \in \mathrm{dom}(\phi)$ is uniquely associated in $\mathtt{ta}$ to an output, and thus an agent a_w. We can then define $\phi\rho$ (omitting the mention of $\mathtt{ta}$ as a slight abuse of notation) as

$$\{\, w \mapsto u\sigma \mid w \in \mathrm{dom}(\phi),\ \sigma \text{ induced by } \rho \text{ on } a_w \,\}.$$

A. Control is determined by associations

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

Proposition 2: Let Π be a well-authenticating protocol, and $\mathcal{P}$ a ground configuration of Π such that

$$(\mathcal{P};\varnothing) \xrightarrow{\mathtt{ta}.\tau_x[A(\overline{k},\overline{n_1})]} (\mathcal{P}';\phi)$$

and the last action is performed by an unsafe conditional. We have $\tau_x = \tau_{\mathsf{then}}$ iff there exists $\overline{n_2}$ such that $\overline{A}(\overline{k},\overline{n_2})$ is associated to $A(\overline{k},\overline{n_1})$ in $(\mathtt{ta},\phi)$.

Proof. The $\Rightarrow$ direction is a direct consequence of well-authentication. For the other direction, we observe that (up to changes of recipes that do not affect the resulting messages) if two agents are associated then they are executing *the* honest trace of Π modulo a renaming of parameters, thus the considered test must be successful. Assuming that $a_1 = A(\overline{k},\overline{n_1})$ and $a_2 = A(\overline{k},\overline{n_2})$ are associated in $(\mathtt{ta},\phi)$, we shall prove that $\tau_x = \tau_{\mathsf{then}}$. By hypothesis, $\mathtt{ta}|_{a_1,a_2}$ is honest: its observable actions are of the form $\mathtt{out}(c_1,w_1).\mathtt{in}(c'_1,M_1)\ldots\mathtt{out}(c_n,w_n).\mathtt{in}(c'_n,M_n)$ with possibly an extra output at the end, and are such that $M_i\phi \Downarrow_{=_\mathsf{E}} w_i\phi$ for all $1 \le i \le n$. Consider $\mathtt{ta}'$ obtained from $\mathtt{ta}$ by replacing each recipe M_i by w_i. Since this change of recipes does not affect the resulting messages, the modified trace can still be executed by $(\mathcal{P};\varnothing)$ and yields the same configuration $(\mathcal{P}';\phi)$. But now $\mathtt{ta}'|_{a_1,a_2}$ is a self-contained execution, *i.e.* if P and Q are the processes respectively annotated a_1 and a_2 in $\mathcal{P}$, we have

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

In that execution, everything is deterministic (up to the equational theory) and thus the execution is actually a prefix of *the* honest execution of Π, up to a bijective renaming of parameters (note that P and Q do not share session parameters). Thus the next action, *i.e.* the conditional performed by a_1, is a τ_{then}. □

B. Invariance of frame idealizations

In general, a renaming of agents can break executability; typically, mapping two dual agents to agents of different identities breaks the ability of these two agents to communicate successfully. Even when executability is preserved, parameters change (so do names) and thus frames are modified. However, frame opacity immediately implies that a renaming of agents has no effect on the resulting *idealized* frames, because the renaming has no effect on the labels associated to the agent outputs. Therefore, we have the following result (proof in Appendix B).

Proposition 3: Let Π be a protocol ensuring frame opacity. Let $\mathcal{P}$ be a ground configuration of Π, $\mathtt{ta}$ an annotated trace, and ρ an arbitrary renaming of agents. If $(\mathcal{P};\varnothing) \xrightarrow{\mathtt{ta}} (\mathcal{P}_1;\phi_1)$ and $(\mathcal{P}\rho;\varnothing) \xrightarrow{\mathtt{ta}\rho} (\mathcal{P}_2;\phi_2)$, then $[\phi_1]^{\mathsf{ideal}} = [\phi_2]^{\mathsf{ideal}}$.

C. A sufficient condition for preserving executability

We can now state a key lemma, identifying a class of renamings which yields indistinguishable executions.

Definition 16: Agents a and a' are *connected* in $(\mathtt{ta},\phi)$ if they are associated in $(\mathtt{ta}_0,\phi)$ for some prefix $\mathtt{ta}_0$ of $\mathtt{ta}$

that contains at least one τ_{then} action of an unsafe conditional annotated with either a or a'.

Lemma 1: Let Π be a well-authenticating protocol ensuring frame opacity, and ta be an annotated trace executed by some ground configuration $\mathcal{P}$:

$$(\mathcal{P};\varnothing) \xrightarrow{\mathsf{ta}} K$$

Let ρ be a renaming of agents whose domain contains the annotations of $\mathcal{P}$, and such that $\rho(a)$ and $\rho(a')$ are dual iff a and a' are connected in $(\mathsf{ta}, \phi(K))$. Then we have:

$$(\mathcal{P}\rho;\varnothing) \xrightarrow{\mathsf{ta}\rho} K\rho \quad \text{and} \quad \phi(K) \sim \phi(K\rho).$$

Proof. We shall focus on establishing that $\mathsf{ta}\rho$ is executable; once this is known, static equivalence is a direct consequence of Proposition 3. For any prefix ta_0 of ta, we prove that:

$$(\mathcal{P}\rho;\varnothing) \xrightarrow{\mathsf{ta}_0\rho} K_0\rho$$

with an additional invariant: $\rho(a)$ and $\rho(a')$ are associated in $(\mathsf{ta}_0\rho, \phi(K_0\rho))$ iff a and a' are associated in $(\mathsf{ta}_0, \phi(K_0))$ and connected in $(\mathsf{ta}, \phi(K))$. We proceed by induction on ta_0. If it is empty, then $\mathsf{ta}_0\rho$ can also obviously be executed. For empty traces, association coincides with duality, thus the hypothesis on ρ implies our invariant.

Consider now a prefix of ta of the form $\mathsf{ta}_0.\alpha[a]$. By induction hypothesis, we have K_0 (resp. $K_0\rho$) resulting from the execution of ta_0 by $\mathcal{P}$ (resp. $\mathsf{ta}_0\rho$ by $\mathcal{P}\rho$) and our invariant satisfied for ta_0. Moreover, by Proposition 3, we know that $\phi(K_0) \sim \phi(K_0\rho)$. The action α performed by the process annotated a in $\mathcal{P}$ may be an input, an output, or a test. In any case, the corresponding process in $\mathcal{P}\rho$ can perform an action of the same nature. To conclude, we distinguish the three kinds of actions:

Case 1: α is an output. We only have to check our invariant for $\mathsf{ta}_0.\alpha[a]$. It essentially follows from the fact that association is not affected by the execution of an output: $\rho(a)$ and $\rho(a')$ are associated in $(\mathsf{ta}_0.\alpha[a])\rho$ iff they are associated in $\mathsf{ta}_0\rho$, and similarly without ρ.

Case 2: α is a conditional. We first need to make sure that the outcome of the test is the same for a and $a\rho$. We distinguish two cases, whether the conditional is safe or not.

- If the conditional is safe, then its outcome only depends on the inputs and outputs of a that are statically equivalent to those of $\rho(a)$. Hence, outcome of that test is the same for a and $a\rho$.
- If the conditional is unsafe, we make use of Proposition 2 to show that the outcome of the conditional is the same on both sides. We can do it because our invariant, in this case, implies that a and a' are associated in ta_0 iff $\rho(a)$ and $\rho(a')$ are associated in $\mathsf{ta}_0\rho$. This is simply because, if a and a' are associated in ta_0, then they are having an honest interaction, thus the outcome of the test will be positive, and a and a' are connected in ta.

In both cases (safe or unsafe) we need to make sure that our invariant is preserved. This is because the association between a and a' is preserved iff the outcome of the test is positive, which is the same before and after the renaming.

Case 3: α is an input. We immediately have that $a\rho$ can perform α, on the same channel and with the same recipe. Let us now check that our invariant is preserved. We only check one direction, the other being very similar. Assume that $\rho(a)$ and $\rho(a')$ are associated in $\mathsf{ta}_0\rho.\alpha[\rho(a)]$. The renamed agents are also associated in ta_0, thus a and a' are connected in ta and associated in ta_0. Now, because α did not break the association of $\rho(a)$ and $\rho(a')$ in $\mathsf{ta}_0\rho$, it must be that the input message in $\alpha = \mathtt{in}(c, M)$ corresponds to the last output of $\rho(a')$ in $\mathsf{ta}_0\rho$. Formally, if that last output corresponds to the handle w in $\phi(K_0\rho)$, we have $M\phi(K_0\rho) \Downarrow_{=\mathsf{E}} w\phi(K_0\rho)$. But, because $\phi(K_0) \sim \phi(K_0\rho)$, we then also have $M\phi(K_0) \Downarrow_{=\mathsf{E}} w\phi(K_0)$. The association of a and a' in ta_0 carries over to $\mathsf{ta}_0.\alpha[a]$. □

D. Proof of Theorem 1

Thanks to our lemma, we can change any execution of $\mathcal{M}_\Pi^{\mathsf{id}}$ into an indistinguishable execution of $\mathcal{S}_\Pi$, provided that an appropriate renaming of agents exists. This is our last step before the final proof:

Proposition 4: For any protocol and any ground configuration $\mathcal{P}$ of the protocol such that

$$(\mathcal{P};\varnothing) \xrightarrow{\mathsf{ta}} K,$$

there exists an agent renaming ρ satisfying the hypothesis of Lemma 1 and such that $\mathcal{P}\rho$ is single-session.

Proof sketch. The renaming maps all session (resp. identity) parameters to new distinct, fresh session (resp. identity) parameters, with the only constraint that connected agents are sent to dual agents (and thus share identity parameters). The precise definition of the renaming as well as the complete proof of this proposition can be found in Section B. □

We now have all the ingredients to prove our main result.

Proof of Theorem 1. It is easy to see that $\mathcal{S}_\Pi \sqsubseteq \mathcal{M}_\Pi \sqsubseteq \mathcal{M}_\Pi^{\mathsf{id}}$, so it only remains to establish that $\mathcal{M}_\Pi^{\mathsf{id}} \sqsubseteq \mathcal{S}_\Pi$. Consider an execution $\mathcal{M}_\Pi^{\mathsf{id}} \xrightarrow{\mathsf{ta}} K$. We can assume w.l.o.g. that session creations are all performed at the beginning of ta, *i.e.* it is of the form $\tau^*.\mathsf{ta}'$ with no occurrence of τ in ta': otherwise we can modify ta to satisfy this condition, without changing its observable actions and the resulting frame. Thus we have a ground configuration $\mathcal{P}$ of Π, such that:

$$(\mathcal{P};\varnothing) \xrightarrow{\mathsf{ta}'} K.$$

Let ρ be the renaming obtained in Proposition 4 for ta'. By Lemma 1, $\mathsf{ta}'\rho$ remains executable and is indistinguishable from ta'. Moreover, since $\mathcal{P}\rho$ is single-session, we have that:

$$(\mathcal{S}_\Pi;\varnothing) \xrightarrow{\tau^*} (\mathcal{P}\rho \cup \mathcal{S}_\Pi;\varnothing) \xrightarrow{\mathsf{ta}'\rho} K\rho.$$

This execution allows us to conclude: it has the same observables as ta, and yields a statically equivalent frame. □

VI. Mechanisation

We now discuss how to delegate the verification of frame-opacity and well-authentication to a fully automatic tool. In this section, we show that appropriate encodings can be used to leverage ProVerif [30] to do so. These encodings have been implemented in our tool UKano [31]. This tool basically takes

as inputs a specification of a protocol in our class and, by applying translations described in this section and by calling ProVerif, it automatically checks our two conditions and thus unlinkability and anonymity.

A. Frame Opacity

We first explain how to check frame opacity using the diff-equivalence feature of ProVerif [32]. Diff-equivalence is a property of *bi-processes*. A bi-process is a process in which some terms are replaced by *bi-terms*, denoted $\texttt{choice}[u_1, u_2]$. Intuitively, a bi-process represents two processes. The first (resp. second) process is obtained by considering terms occurring on the left-hand side (resp. right-hand side) of the choice operators. Checking the diff-equivalence of a bi-process boils down to checking that when the two processes are executed simultaneously, the resulting frames are in static equivalence.

Our notion of frame opacity requires that for any execution $\mathcal{M}_\Pi^{\mathsf{id}} \xrightarrow{\mathsf{ta}} (P;\phi)$, one has *(1)* $\phi \sim \psi$ for some $\psi \in [\phi]^{\mathsf{nonce}}$; and *(2)* outputs carrying the same label produce messages with the same idealization. It is possible to verify both points by checking the diff-equivalence between $\mathcal{M}_\Pi^{\mathsf{id}}$ and a modified version of this process where each syntactical output u (identified by a label) has been replaced by a *static idealization*, *i.e.* the idealization of some message that this output may produce.

We assume that all syntactical outputs of the protocol can be executed. This is obviously not a restriction in practice as dead code should be removed and can be detected very easily using, *e.g.* ProVerif. Under this assumption we shall compute, for each syntactical output $\ell : \texttt{out}(c, u)$, its static idealization u_ℓ^{ideal}: knowing that there is at least one execution where this output is triggered, producing some message m, we simply set $u_\ell^{\mathsf{ideal}} = [m]^{\mathsf{ideal}}$. Choosing an arbitrary execution in this way is justified since all possible messages that this output may produce must have the same idealization anyway.

Let us now describe the bi-process $\mathrm{biproc}(\mathcal{M}_\Pi^{\mathsf{id}})$ whose diff-equivalence implies frame opacity. As a first approximation, the bi-process is defined from $\mathcal{M}_\Pi^{\mathsf{id}}$ by replacing each $\ell : \texttt{out}(c, u_\ell)$ by $\ell : \texttt{out}(c, \texttt{choice}[u_\ell, u_\ell^{\mathsf{nonce}}])$ where u_ℓ^{nonce} is obtained from u_ℓ^{ideal} by filling its holes with fresh names. A crucial point is to consider fresh names from messages u_ℓ^{nonce} as new session names of the bi-process so that they will be different for each session. The only remaining issue at this stage is that diff-equivalence in ProVerif forces the left-hand and right-hand processes to execute exactly the same kind of actions at the same time. This might be a problem for conditionals that have no real meaning for the right-hand part. We overcome this difficulty in the actual definition of $\mathrm{biproc}(\mathcal{M}_\Pi^{\mathsf{id}})$ by pushing conditionals into messages and putting else branches in parallel. We do not formally explain how to do so as it heavily depends on specificities of ProVerif, but just give an example to illustrate the pushing of conditionals: we show in Figure 4 (part of) the bi-process resulting from the application of our transformation to our running example. One can see that the computation of `merge` never fails (neither on the left, nor on the right) because `catchFail` always returns a message. More examples can be found in the ProVerif files produced by UKano for our case studies, available online [31].

```
! new k;
! new nI; new nR; new n1; new n2; new n3;
 (   (* Initiator role: *)
  out(cI,choice[nI, n1]);
  in(cI,x);
  let merge = choice[
    let catchFail =
      let yt,ynR = eq(pi1(dec(x,k)),nI),
                   pi2(dec(x,k)) in
        enc((ynR,nI),k)
    else n2,
    n2] in
  out(cI,merge))
|(   (*  Responder role: *)
  in(cR,z);
  out(cR,choice[enc((nI,nR),k), n3]); ...)
```

Fig. 4. Example of ProVerif file checking frame opacity (Feldhofer)

Assuming that diff-equivalence holds for $\mathrm{biproc}(\mathcal{M}_\Pi^{\mathsf{id}})$, we have that frame opacity holds too. Indeed, for any execution $\mathcal{M}_\Pi^{\mathsf{id}} \xrightarrow{\mathsf{tr}} (P;\phi)$, there exists some execution $\mathrm{biproc}(\mathcal{M}_\Pi^{\mathsf{id}}) \xrightarrow{\mathsf{tr}} (Q, [\phi, \phi_r])$ with $\phi \sim \phi_r$. By construction of the bi-process we have that $\phi_r \in [\phi]^{\mathsf{nonce}}$, which implies item *(1)* of frame opacity. Moreover, idealizations only depends on output labels, which implies item *(2)*.

B. Well-authentication

We first explain how to check condition *(i)* of well-authentication. It is basically a conjunction of reachability properties, which can be checked in ProVerif using correspondence properties [33]. For each role $\mathtt{A} \in \{\mathcal{I}, \mathcal{R}\}$, we associate to each syntactical output (resp. input) of the role an event which uniquely identifies the action. More formally, we use events of the form $\mathtt{OutA_i}(\overline{k}, \overline{n}, m)$ and $\mathtt{InA_j}(\overline{k}, \overline{n}, m)$, whose arguments contain:

- identity parameters $\overline{k}$ and session parameters $\overline{n}$;
- the message m that is inputted or outputted.

In the same fashion, we also add events of the form $\mathtt{TestA_k}(\overline{k}, \overline{n})$ at the beginning of each then branch.

For each conditional of the protocol, we first check if the simple syntactical criterion of *safe* conditionals holds. If it is the case we do nothing for this conditional. Otherwise, we need to check condition *(i)*. It can be expressed as a correspondence property using events as explained next. Given a role $\mathtt{A} \in \{\mathcal{I}, \mathcal{R}\}$ and a conditional of this role whose event is $\mathtt{TestA_i}(\overline{k}, \overline{n})$, the fact that if the conditional is positively evaluated, then the involved agent must be associated to a dual agent, can be expressed by the following correspondence property:

1) when the event $\mathtt{TestA_i}(\overline{k}, \overline{n})$ is fired,
2) there must be a previous event $\mathtt{InA_j}(\overline{k}, \overline{n}, m)$ ($\mathtt{InA_j}$ corresponding to the input just before the conditional),

3) and a previous event $\mathtt{OutB_k}(\overline{k}, \overline{n}', m)$ ($\mathtt{OutB_k}$ corresponding to the output that fed the input $\mathtt{InA_j}$ in the honest execution),
4) and a previous event $\mathtt{InB_1}(\overline{k}, \overline{n}', m')$ ($\mathtt{InB_1}$ corresponding to the first input before $\mathtt{OutB_k}$),
5) and a previous event $\mathtt{OutA_m}(\overline{k}, \overline{n}, m')$ ($\mathtt{OutA_m}$ corresponding to the output that fed the input $\mathtt{InB_1}$ in the honest execution), etc.

Note that by using the same messages m and m' for inputs and outputs, we express that the messages that are outputted and inputted are equal modulo the equational theory E.

Example 16: Those kinds of correspondence properties are better explained by showing the ProVerif code we produce. We depict in Figure 5 the query we produce for checking condition *(i)* on the first conditional of P_I from our running example.

```
query k:key, n1:bitstring, n2:bitstring,
      nt:bitstring, nr:bitstring,
      mP:bitstring, mR:bitstring;
      event(TestI1(k,n1)) ==>
        (event(InI1(k,n1,mR)) ==>
          (event(OutR1(k,n2,mR)) ==>
            (event(InR1(k,n2,mP)) ==>
              (event(OutI1(k,n1,mP)))
        ) ) ).
```

Fig. 5. Example of ProVerif query for checking condition *(i)*

Finally, checking that condition *(ii)* of well-authentication is satisfied is rather trivial once we know that the the other two conditions hold. Indeed, as shown in the following lemma, condition *(ii)* holds as soon as the first conditional occurring in the responder role has been identified as safe. Remark that if the latter does not hold then unlinkability most likely fails to hold.

Lemma 2: Let $\Pi = (\overline{k}, \overline{n}_I, \overline{n}_R, \mathcal{I}, \mathcal{R})$ be a protocol that satisfies condition *(i)* of well-authentication, and frame opacity. If the first conditional that occurs in $\mathcal{R}$ is safe, then condition *(ii)* of well-authentication holds.

Proof. Consider an execution where two agents a and a' are associated and a' has performed a τ_{then}. If this test corresponds to a safe conditional, there is nothing to prove. Otherwise, we shall prove that a is only associated to a'. Thanks to our hypothesis, a has performed at least one input even if it is an initiator. Let m be that input message. We know that it is equal (modulo E) to the previous output of a', and want to show that it cannot be equal to any output of another agent. Let ℓ be the label of the previous output of a'. By definition of a protocol, that output label cannot correspond to a public message in the honest trace. Thus the idealization of the output message associated to ℓ in the honest trace contains at least one hole. By condition *(2)* of frame opacity, the same holds for the idealization of m. Therefore, if m had been outputted twice, it would have lead to two different messages in the idealized frames, violating frame opacity. □

VII. Case studies

In this section we apply our proof technique to several case studies. We rely on our tool UKano to check automatically whether the protocol under study satisfies frame opacity and well-authentication as defined in Section IV. We also discuss some variations of the protocols to examine how privacy is affected. As explained in Section VI, UKano relies on ProVerif; the source code of our tool, and all produced ProVerif files can be found in [31].

A. Feldhofer's protocol

As already mentioned, this protocol falls into our generic class of 2-party protocols. We succeeded in establishing automatically frame opacity and well-authentication. For both requirements, UKano concludes in less than 1 second.

B. Hash-Lock protocol

We consider the Hash-Lock protocol as described in [34]. This is an RFID protocol that has been designed to achieve privacy even if no formal proof is given. The protocol relies on a hash function, and can be informally described as follows.

$$\begin{array}{rcll} \mathsf{Reader} & \rightarrow & \mathsf{Tag}: & n_R \\ \mathsf{Tag} & \rightarrow & \mathsf{Reader}: & n_T,\ \mathsf{h}(n_R, n_T, k) \end{array}$$

This protocol falls into our generic class of 2-party protocols, and frame opacity and well-authentication can be automatically established in less than 1 second. We can therefore conclude that the protocol preserves unlinkability.

C. LAK protocol

We present an RFID protocol first introduced in [35], and we refer to the description given in [2]. To avoid traceability attacks, the main idea is to ask the tag to generate a nonce and to use it to send a different message at each session. We suppose that initially, each tag has his own key k and the reader maintains a database containing those keys.

The protocol is informally described below (h models an hash function). In the original version (see *e.g.* [2]), in case of a successful execution, both parties update the key k with $\mathsf{h}(k)$ (they always store the two last keys). Our framework does not allow one to model protocol that rely on a mutable state. Therefore, we consider here a version where the key is not updated at the end of a successful execution allowing the key k to be reuse from one session to another.

$$\begin{array}{rcll} \mathsf{Reader} & \rightarrow & \mathsf{Tag}: & r_1 \\ \mathsf{Tag} & \rightarrow & \mathsf{Reader}: & r_2,\ \mathsf{h}(r_1 \oplus r_2 \oplus k) \\ \mathsf{Reader} & \rightarrow & \mathsf{Tag}: & \mathsf{h}(\mathsf{h}(r_1 \oplus r_2 \oplus k) \oplus k \oplus r_1) \end{array}$$

Actually, this protocol suffers from an authentication attack. The protocol does not allow the reader to authenticate the tag. This attack can be informally described as follows (and already exists on the original version of this protocol). By using algebraic properties of $\oplus$, an attacker can impersonate

a tag by injecting previously eavesdropped messages. Below, $I(\mathsf{A})$ means that the attacker plays the role A.

$$\begin{array}{rcll} I(\mathsf{Reader}) & \to & \mathsf{Tag}: & r_1 \\ \mathsf{Tag} & \to & \mathsf{Reader}: & r_2,\ \mathsf{h}(r_1 \oplus r_2 \oplus k) \\ \mathsf{Reader} & \to & \mathsf{Tag}: & r_1' \\ I(\mathsf{Tag}) & \to & \mathsf{Reader}: & r_2^I,\ \mathsf{h}(r_0 \oplus r_1 \oplus k) \\ \mathsf{Reader} & \to & \mathsf{Tag}: & \mathsf{h}(\mathsf{h}(r_0 \oplus r_1 \oplus k) \oplus k \oplus r_1') \end{array}$$

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

Due to this, the protocol does not satisfy our well-authentication requirement. Indeed, the reader can end a session with a tag whereas the tag has not really participated to this session. In other words, the reader passes a test (which does not correspond to a safe conditional) with success, and therefore performs a τ_{then} action whereas it has not interact honestly with a tag.

Actually, this trace can be turned into an attack against the unlinkability property. Indeed, by continuing the previous trace, the reader can send a new request to the tag generating a fresh nonce r_1''. The attacker $I(\mathsf{Tag})$ can again answer to this new request choosing his nonce r_2'' accordingly, *i.e.* $r_2'' = r_1'' \oplus r_1 \oplus r_2$. This execution, involving two sessions of the reader talking to the same tag, cannot be mimicked in the single session scenario, and corresponds to an attack trace.

More importantly, this scenario can be seen as a traceability attack on the original version of the protocol (the stateful version) leading to a practical attack. The attacker will first start a session with the targeted tag by sending it a nonce r_0 and storing its answer. Then, later on, he will interact with the reader as described in the second part of the attack scenario. Two situations may occur: either the interaction is successful meaning that the targeted tag has not been used since its last interaction with the attacker; or the interaction fails meaning that the key has been updated on the reader's side, and thus the targeted tag has performed a session with the reader since its last interaction with the attacker. This attack shows that the reader may be the source of leaks exploited by the attacker to trace a tag. This is why we advocate for the strong notion of unlinkability we used, taking into account the reader and considering it as important as the tag.

We may note that the same protocol was declared untraceable in [2] due to the fact that they have in mind a weaker notion of unlinkability.

To avoid the algebraic attack due to the properties of the xor operator, we may replace this operator using the pairing operator. The resulting protocol is a 2-party protocol that falls into our class, and for which frame opacity and well-authentication can be established using UKano, again in less than 1 second. Therefore, Theorem 1 allows us to conclude that it preserves unlinkability.

D. BAC protocol and some others

An e-passport is a paper passport with an RFID chip that stores the critical information printed on the passport. The International Civil Aviation Organization (ICAO) standard [36] specifies several protocols through which this information can be accessed. Before executing the Basic Access Control (BAC) protocol, the reader optically scans a weak secret from which it derives two keys k_E and k_M that are then shared between the passport and the reader. Then, the BAC protocol establishes a key seed from which two sessions keys are derived. The session keys are then used to prevent skimming and eavesdropping on the subsequent communication with the e-passport.

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

The BAC protocol using Alice & Bob notation between Tag *(*i.e. *passport) and* Reader *is depicted above. A process modeling* Tag *more precisely is defined below, where* $m = \mathsf{enc}(\langle n_T, \langle \pi_1(\mathsf{dec}(x_E, k_E)), k_T\rangle\rangle, k_E)$.

$$\begin{array}{l} T(k_E, k_M) = \nu n_T.\nu k_T.\mathtt{out}(c_T, n_T).\mathtt{in}(c_T, x). \\ \mathsf{let}\ x_E = \pi_1(x), x_M = \pi_2(x), z_{\mathsf{test}} = \mathsf{eq}(x_M, \mathsf{mac}(x_E, k_M))\ \mathtt{in} \\ \quad \mathsf{let}\ z'_{\mathsf{test}} = \mathsf{eq}(n_T, \pi_1(\pi_2(\mathsf{dec}(x_E, k_E))))\ \mathtt{in} \\ \qquad\qquad \mathtt{out}(c_T, \langle m, \mathsf{mac}(m, k_M)\rangle) \\ \qquad\qquad \mathsf{else}\ \mathtt{out}(\mathsf{error}_{\mathsf{Nonce}}) \\ \quad \mathsf{else}\ \mathtt{out}(\mathsf{error}_{\mathsf{Mac}}) \end{array}$$

We consider the signature given in Example 1 augmented with a function symbol mac *of arity* 2*. This is a public constructor whose purpose is to model message authentication code, taking as arguments the message to authenticate and the mac key. There is no rewriting rule and no equation regarding this symbol. We also assume public constants to model error messages. The UK version of the protocol does not distinguish the two cases of failure,* i.e. $\mathsf{error}_{\mathsf{Mac}}$ *and* $\mathsf{error}_{\mathsf{Nonce}}$ *are the same constant, whereas the French version does.*

Fig. 6. Description of the BAC protocol

In [8], two variants of the BAC protocol are described and analyzed w.r.t. the unlinkability property as formally stated in this paper. We refer below to these two variants as the French version and the UK version. The UK version is claimed unlinkable (with no formal proof) whereas an attack is reported on the French version. To explain the difference between the two versions, we give a description of the passport's role in Figure 6. The relevant point is the fact that, in case of failure, the French version sends a different error message indicating whether the failure occurs due to a problem when checking the mac, or when checking the nonce. This allows the attacker to exploit this conditional to learn if the mac key of a Tag is the one used in a given message $\langle m, \mathsf{mac}(m, k)\rangle$. Using this, he can very easily trace a tag T by first eavesdropping an honest interaction between the tag T and a reader.

The UK version of the BAC protocol is a 2-party protocol

according to our definition[1]. Note that since the two error messages are actually identical, we can merge the two let instructions, and therefore satisfy our definition of being a responder role. Then, we established frame opacity and well-authentication using UKano. It took less than 1 minute. Therefore, Theorem 1 allows us to conclude that unlinkability is indeed satisfied.

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

Once the BAC protocol has been successfully executed, the reader gains access to the information stored in the RFID tag through the Passive and Active Authentication protocols (PA and AA). They are respectively used to prove authenticity of the stored information and prevent cloning attacks, and may be executed in any order. A formal description of these protocols is available in [37]. These two protocols also fall into our class and our conditions can be checked automatically both for unlinkability and anonymity properties. We can also use our technique to analyze directly the three protocols together (*i.e.* the UK version of the BAC together with the PA and AA protocols in any order). We thus prove unlinkability and anonymity w.r.t. all private data stored in the RFID chip (name, picture, etc.). UKano concludes within 7 minutes to establish both well-authentication and frame opacity.

E. PACE protocol

The Password Authenticated Connection Establishment protocol [38] (PACE) has been proposed by the German Federal Office for Information Security (BSI) to replace the BAC protocol. It has been studied in the literature [29], [39], [40] but to the best of our knowledge, no formal proofs about privacy have been given. Similarly to BAC, the purpose of PACE is to establish a secure channel based on an optically-scanned key k. The protocol comprises four main steps (see Figure 7):

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

[1]We do not model the getChallenge constant message that is used to initiate the protocol but it is clear this message does not play any role regarding the security of the protocol.

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

Fig. 7. PACE in Alice & Bob notation

A description in Alice & Bob notation is given in Figure 7. Moreover, at step 6, the reader will not accept as input a message which is equal to the previous message that it has just sent.

To formalize such a protocol, we consider the following signature:

$$\Sigma_c = \{\mathsf{enc},\ \mathsf{dec},\ \mathsf{dh},\ \mathsf{mac},\ \mathsf{gen},\ \mathsf{g},\ \mathsf{ok}\} \text{ and } \Sigma_d = \{\mathsf{neq}\}.$$

Except g and ok which are public constants, all these function symbols are public constructor symbols of arity 2. The destructor neq has already be defined in Section II. The symbol dh is used to model modular exponentiation whereas mac will be used to model message authentication code. We consider the equational theory E defined by the following equations:

$$\begin{aligned} \mathsf{dec}(\mathsf{enc}(x,y),y) &= x \\ \mathsf{dh}(\mathsf{dh}(x,y),z) &= \mathsf{dh}(\mathsf{dh}(x,z),y) \end{aligned}$$

This protocol falls into our generic class of 2-party protocols. We take

$$\Pi_{\mathsf{PACE}} = (k, \{s_T, n_T, n'_T\}, \{n_R, n'_R\}, \mathcal{I}_{\mathsf{PACE}}, \mathcal{R}_{\mathsf{PACE}})$$

where the $\mathcal{R}_{\mathsf{PACE}}$ process (reader), described in Figure 8, is a responder role (we do not detail the continuation R' and we omit trivial conditionals). The process modeling the role $\mathcal{I}_{\mathsf{PACE}}$ can be obtained in a similar way.

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

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

Fig. 8. Process $\mathcal{R}_{\mathsf{PACE}}$

Unfortunately, ProVerif cannot handle the equation above on the dh operator (due to some termination issues). Instead of that single equation, we consider the following equational theory that is more suitable for ProVerif:

$$\begin{aligned} \mathsf{dh}(\mathsf{dh}(\mathsf{g}, y), z) &= \mathsf{dh}(\mathsf{dh}(\mathsf{g}, z), y) \\ \mathsf{dh}(\mathsf{dh}(\mathsf{gen}(x_1, x_2), y), z) &= \mathsf{dh}(\mathsf{dh}(\mathsf{gen}(x_1, x_2), z), y) \end{aligned}$$

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

First, we would like to highlight an imprecision in the official specification [38] that may lead to practical attacks on unlinkability. As the specification seems to not forbid it, we could have assumed that the decryption operation in $G = \mathsf{gen}(\mathsf{dec}(y_1, k), \mathsf{dh}(y_2, n_R))$ is implemented in such a way that it may fail when the key k does not match with the key of the ciphertext y_1. In that case, an attacker could eavesdrop a first message $c^0 = \mathsf{enc}(s_T^0, k^0)$ of a certain tag T^0 and then, in a future session, it would let the reader optically scan a tag T but replace its challenge $\mathsf{enc}(s_T, k)$ by c^0 and wait for an answer of the reader. If it answers, he learns that the decryption did not fail and thus $k = k^0$: the tag T is actually T^0. We discovered this attack using our method since in our first attempt to modelize the protocol, we modelized $\mathsf{dec}(\cdot,\cdot)$ as a destructor (that may fail) and the computation of G as an evaluation:

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

This test has to satisfy our requirement in order to declare the protocol well-authenticating. But this conditional computing G is not safe and does not satisfy the requirements of Definition 14 (the attack scenario described is a counter example). The same attack scenario shows that the protocol does not ensure unlinkability (this scenario cannot be observed when interacting with $\mathcal{S}_\Pi$). Similarly to the attack on LAK, we highlight here the importance to take the reader into account and give it as much importance as the tag in the definition of unlinkability. Indeed, it is actually a leakage from the reader that allows an attacker to trace a specific tag.

Second, we report on an attack[2] that that we discovered using our method on some modelizations of PACE found in the literature [29], [39], [40]. Indeed, in all those papers, the first conditional of the reader

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

is omitted. Then the resulting protocol is not well-authenticating. To see this, we simply have to consider a scenario where the attacker will send to the reader the message it has outputted at the previous step. Such an execution will allow the reader to execute its role until the end, and therefore execute τ_{then}, but the resulting trace is not an honest one. Again, this scenario can be turned into an attack against unlinkability as explained next. As before, an attacker could eavesdrop a first message $c^0 = \mathsf{enc}(s_T^0, k^0)$ of a certain tag T^0. Then, in a future session, it would let the reader optically scans a tag T but replace its challenge $\mathsf{enc}(s_T, k)$ by c^0. Whatever k is equal or k^0, the reader answers g^{n_R}. The attacker then plays the two rounds of Diffie-Hellman by reusing messages from the reader (he actually performs a reflection attack). More precisely, he replies with $g^{n_T} = g^{n_R}$, $G^{n'_T} = G^{n'_R}$ and $\mathsf{mac}(G^{n'_R}, k') = \mathsf{mac}(G^{n'_T}, k')$. The crucial point is that the attacker did not prove he knows k (he supposed to do so to generate G at step 4) thanks to the reflection attack that is not detected. Now, the attacker waits for the reader's answer. If it is positive (the process R' is executed), he learns that $k = k^0$: the tag T is actually the same as T^0.

Third, we turn to PACE as properly understood from the official specification: when the latter test is present and the decryption may not fail. In that case, we report on a new attack. UKano found that the last test of the reader violates well-authentication. This is the case for the following scenario: the message $\mathsf{enc}(s_T, k)$ from a tag $T(k, n_T)$ is fed to two readers $R(k, n_R^1), R(k, n_R^2)$ of same identity name. Then, the attacker just forwards messages from one reader to the other. They can thus complete the two rounds of Diffie-Hellman (note that the test avoiding reflection attacks holds). More importantly, the mac-key verification phase (messages 8 and 9 from Figure 7) goes well and the attacker observes that the last conditional of the two readers holds. This violates well-authentication but also unlinkability because the latter scenario cannot be observed at all in $\mathcal{S}_\Pi$: if the attacker makes two readers talk to each other in $\mathcal{S}_\Pi$ they cannot complete a session because they must have different identity names. In practice, this flaw seems hard to exploit but it could be a real privacy concern: if a tag initiates multiple readers, an attacker may learn which ones it had initiated by forwarding messages from one to another. It does not seem to be realistic in the e-passport scenario, but could be harmful in other contexts.

Finally, we propose a simple fix to the above attack by adding tags avoiding confusions between reader's messages and tag's messages. It suffices to replace messages 8 and 9 from Figure 7 by respectively $\mathsf{mac}(\langle \mathsf{c}_r, G^{n'_T}\rangle, k')$ and $\mathsf{mac}(\langle \mathsf{c}_t, G^{n'_R}\rangle, k')$ where $\mathsf{c}_r, \mathsf{c}_t$ are public constants, and adding the corresponding checks. Frame opacity and well-authentication can be automatically established using UKano, in around 20 minutes. Therefore, PACE with tags preserves unlinkability in the model considered here.

VIII. Conclusion

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

In the future, we plan to develop a mature implementation of our tool in order to make it widely accessible for the design and study of privacy-preserving 2-party protocols. We could also try to translate our conditions into more comprehensive guidelines helping the design of new privacy-enhancing protocols.

[2]For that different attack, we obviously consider that decryption is a constructor, and thus cannot fail.

We also identify a number of research problems aimed at increasing the scope of our technique. Currently, our conditions are checked using ProVerif which, despite its great flexibility, supports only a limited kind of equational theory. In particular, full Diffie-Hellman theory or associative-commutative theories needed for xor (widely used in RFID protocols) are not supported. It seems likely that frame opacity can be checked using ad-hoc methods rather than ProVerif, which could support wider classes of theories. Concerning well-authentication, we could consider various extensions of ProVerif with partial support for xor [41], or other tools such as Tamarin and Maude–NPA. We would also like to investigate the extension of our main theorem to the case of protocols with state. This is certainly technically challenging, but would make it possible to model more protocols, or at least model them more faithfully. Finally, we would like to investigate whether our frame opacity condition could be relaxed to allow one to deal more precisely with primitives that are neither transparent, nor totally opaque in general (*e.g.* zero-knowledge proofs, signatures).

REFERENCES

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

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

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

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

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

[6] S. Meier, B. Schmidt, C. Cremers, and D. Basin, "The Tamarin Prover for the Symbolic Analysis of Security Protocols," in *Proc. 25th International Conference on Computer Aided Verification (CAV'13)*, ser. LNCS, vol. 8044. Springer, 2013, pp. 696–701.

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

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

[9] M. Bruso, K. Chatzikokolakis, and J. den Hartog, "Formal verification of privacy for RFID systems," in *Proceedings of CSF'10*, 2010.

[10] S. Delaune, S. Kremer, and M. D. Ryan, "Verifying privacy-type properties of electronic voting protocols," *Journal of Computer Security*, no. 4, 2008.

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

[12] N. Dong, H. Jonker, and J. Pang, "Formal analysis of privacy in an ehealth protocol," in *Computer Security–ESORICS 2012*. Springer, 2012, pp. 325–342.

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

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

[15] R. Chrétien, V. Cortier, and S. Delaune, "From security protocols to pushdown automata," *ACM Transactions on Computational Logic*, vol. 17, no. 1:3, Sep. 2015.

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

[17] V. Cheval, H. Comon-Lundh, and S. Delaune, "Trace equivalence decision: Negative tests and non-determinism," in *Proceedings of CCS'11*. ACM Press, 2011.

[18] V. Cheval and B. Blanchet, "Proving more observational equivalences with proverif," in *Principles of Security and Trust*. Springer, 2013, pp. 226–246.

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

[20] S. Santiago, S. Escobar, C. Meadows, and J. Meseguer, "A formal definition of protocol indistinguishability and its verification using maude-npa," in *Security and Trust Management*. Springer, 2014, pp. 162–177.

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

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

[23] V. Cortier, S. Delaune, and P. Lafourcade, "A survey of algebraic properties used in cryptographic protocols," *Journal of Computer Security*, vol. 14, no. 1, pp. 1–43, 2006.

[24] M. Feldhofer, S. Dominikus, and J. Wolkerstorfer, "Strong authentication for RFID systems using the AES algorithm," in *Cryptographic Hardware and Embedded Systems-CHES 2004*. Springer, 2004, pp. 357–370.

[25] S. Delaune, S. Kremer, and M. D. Ryan, "Verifying privacy-type properties of electronic voting protocols: A taster," in *Towards Trustworthy Elections – New Directions in Electronic Voting*. Springer, 2010, vol. 6000.

[26] M. Brusó, K. Chatzikokolakis, S. Etalle, and J. Den Hartog, "Linking unlinkability," in *Trustworthy Global Computing*. Springer, 2012, pp. 129–144.

[27] M. Brusó, "Dissecting unlinkability," Ph.D. dissertation, Technische Universiteit Eindhoven, 2014.

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

[29] J. Bender, M. Fischlin, and D. Kügler, "Security analysis of the pace key-agreement protocol," in *Information Security*. Springer, 2009, pp. 33–48.

[30] "Proverif: Cryptographic protocol verifier in the formal model." [Online]. Available: http://prosecco.gforge.inria.fr/personal/bblanche/proverif/

[31] "UKano tool and case studies." [Online]. Available: http://projects.lsv.ens-cachan.fr/ukano/

[32] B. Blanchet, M. Abadi, and C. Fournet, "Automated verification of selected equivalences for security protocols," in *Logic in Computer Science, 2005. LICS 2005. Proceedings. 20th Annual IEEE Symposium on*. IEEE, 2005, pp. 331–340.

[33] M. Abadi and B. Blanchet, "Computer-assisted verification of a protocol for certified email," in *Static Analysis*. Springer, 2003, pp. 316–335.

[34] A. Juels and S. A. Weis, "Defining strong privacy for RFID," *ACM Transactions on Information and System Security (TISSEC)*, vol. 13, no. 1, p. 7, 2009.

[35] S. Lee, T. Asano, and K. Kim, "RFID mutual authentication scheme based on synchronized secret information," in *Symposium on cryptography and information security*, 2006.

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

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

[38] "Technical advisory group on machine readable travel documents (tag/mrtd)." [Online]. Available: http://www.icao.int/Meetings/TAG-MRTD/TagMrtd22/TAG-MRTD-22_WP05.pdf

[39] J. Bender, Ö. Dagdelen, M. Fischlin, and D. Kügler, "The pace aa protocol for machine readable travel documents, and its security," in *Financial Cryptography and Data Security*. Springer, 2012, pp. 344–358.
[40] L. Cheikhrouhou, W. Stephan, Ö. Dagdelen, M. Fischlin, and M. Ullmann, "Merging the cryptographic security analysis and the algebraic-logic security proof of pace." in *Sicherheit*, 2012, pp. 83–94.
[41] R. Küsters and T. Truderung, "Reducing protocol analysis with XOR to the xor-free case in the horn theory based approach," *J. Autom. Reasoning*, vol. 46, no. 3-4, pp. 325–352, 2011.

APPENDIX A
PROOFS OF SECTION IV

We detail below how to obtain Proposition 1.

Definition 17: Let $u \in \mathcal{T}(\Sigma_c, \mathcal{N})$. We define $h_0(u)$ as the maximum number of nested transparent function symbols in u. Then, $h_t(u)$ is the minimum of all $h_0(v)$ for $u =_{\mathsf{E}} v$.

Proposition 5: For any $u =_{\mathsf{E}} \mathsf{f}(u_1, \ldots, u_n)$ with $\mathsf{f} \in \Sigma_t$, we have $h_t(u) > h_t(u_i)$ for all i.

Proof. We show that, for all $v =_{\mathsf{E}} u$, $h_0(v) > h_t(u_i)$. Since $v =_{\mathsf{E}} \mathsf{f}(u_1, \ldots, u_n)$, and since the equational theory cannot involve f by definition of Σ_t, we have $v = \mathsf{f}(v_1, \ldots, v_n)$ with $v_i =_{\mathsf{E}} u_i$. We conclude: $h_t(u_i) = h_t(v_i) \leq h_0(v_i) < h_0(v)$. □

Definition 18: The relation $\mathcal{R}^{\mathrm{ideal}}$: $\mathcal{T}(\Sigma_c, \mathcal{N}) \times \mathcal{T}(\Sigma_t, \{\Box\})$ is the least relation such that:

- $u \mathcal{R}^{\mathrm{ideal}} \mathsf{f}(t_1, \ldots, t_n)$ if there exist $\mathsf{f} \in \Sigma_t$ and messages u_i for $1 \leq i \leq n = \mathrm{ar}(\mathsf{f})$, such that $u =_{\mathsf{E}} \mathsf{f}(u_1, \ldots, u_n)$ and $u_i \mathcal{R}^{\mathrm{ideal}} t_i$ for all $1 \leq i \leq n$;
- $u \mathcal{R}^{\mathrm{ideal}} \Box$ otherwise.

Proposition 6: For all message u there exists a v such that $u \mathcal{R}^{\mathrm{ideal}} v$. Furthermore, whenever $u =_{\mathsf{E}} u'$, $u \mathcal{R}^{\mathrm{ideal}} v$ and $u' \mathcal{R}^{\mathrm{ideal}} v'$, then we have that $v = v'$.

Proof. We proceed by induction over $h_t(u)$. If u cannot be equated to a message with a transparent function symbol at toplevel, then the result is obvious with $v = \Box$. Otherwise, assume $u =_{\mathsf{E}} \mathsf{f}(u_1, \ldots, u_n)$. By induction hypothesis we obtain $u_i \mathcal{R}^{\mathrm{ideal}} v_i$ for all i, and thus $u \mathcal{R}^{\mathrm{ideal}} \mathsf{f}(v_1, \ldots, v_n)$. Consider now u', v and v' such that $u =_{\mathsf{E}} u'$, $u \mathcal{R}^{\mathrm{ideal}} v$ and $u' \mathcal{R}^{\mathrm{ideal}} v'$. Observe that $u' =_{\mathsf{E}} \mathsf{f}'(u'_1, \ldots, u'_m)$ is only possible if $\mathsf{f} = \mathsf{f}'$, $n = m$ and $u_i =_{\mathsf{E}} u'_i$ for all i. Thus $v = \mathsf{f}(v_1, \ldots, v_n)$ and $v' = \mathsf{f}(v'_1, \ldots, v'_n)$. Moreover, $v_i = v'_i$ by induction hypothesis on u_i, which concludes the proof. □

The above results immediately allow to show Proposition 1, by defining $[u]^{\mathsf{ideal}}$ to be the unique v such that $u \mathcal{R}^{\mathrm{ideal}} v$. The last point is an easy consequence of the definition of $[\cdot]^{\mathsf{nonce}}$.

APPENDIX B
PROOFS OF SECTION V

Proposition 3: Let Π be a protocol ensuring frame opacity. Let $\mathcal{P}$ be a ground configuration of Π, ta an annotated trace, and ρ an arbitrary renaming of agents. If $(\mathcal{P}; \varnothing) \xrightarrow{\mathsf{ta}} (\mathcal{P}_1; \phi_1)$ and $(\mathcal{P}\rho; \varnothing) \xrightarrow{\mathsf{ta}\rho} (\mathcal{P}_2; \phi_2)$, then $[\phi_1]^{\mathsf{ideal}} = [\phi_2]^{\mathsf{ideal}}$.

Proof. Intuitively, this is a consequence of the second point (ii) of frame opacity for the execution where ta and $\mathsf{ta}\rho$ are executed in sequence, because any handle w in $\mathrm{dom}(\phi_1) = \mathrm{dom}(\phi_2)$ must carry the same label in ϕ_1 and ϕ_2. Formally, we consider a bijective renaming of names (instead of agents) σ such that no agent is both in $\mathcal{P}$ and $\mathcal{P}\rho\sigma$, so that $\mathcal{P} \cup (\mathcal{P}\rho\sigma)$ is still a ground configuration. We also consider a bijection θ over $\mathcal{W}$ such that $\mathrm{dom}(\phi_1) \cap \mathrm{dom}(\phi_2\theta) = \varnothing$ and θ is the identity on $\mathrm{dom}(\phi_1)$. Let ρ' be $\rho\sigma$ and $\mathsf{ta}' = \mathsf{ta}\rho'\theta$. We can deduce from the execution of $\mathsf{ta}\rho$ by $\mathcal{P}\rho$ that $(\mathcal{P}\rho'; \varnothing) \xrightarrow{\mathsf{ta}'} (\mathcal{P}'_2; \phi'_2)$ with $\mathcal{P}'_2 = \mathcal{P}_2\sigma$ and $\phi'_2 = \theta^{-1}\phi_2\sigma$. By concatenating the above executions, we obtain

$$(\mathcal{P} \cup (\mathcal{P}\rho'); \varnothing) \xrightarrow{\mathsf{ta}.\mathsf{ta}'} (\mathcal{P}_1 \cup \mathcal{P}'_2; \phi_1 \cup \phi'_2).$$

Further, we have $(\mathcal{M}^{\mathsf{id}}_{\Pi}; \varnothing) \xrightarrow{\mathsf{ta}_0} (\mathcal{P} \cup (\mathcal{P}\rho'); \varnothing)$ for some ta_0 because $\mathcal{P} \cup (\mathcal{P}\rho')$ is a ground configuration. Applying frame opacity on the execution of $\mathsf{ta}_0.\mathsf{ta}.\mathsf{ta}'$, we obtain that for all $w, w' \in \mathrm{dom}(\phi_1 \cup \phi'_2)$ that carry the same label $\ell \in \mathcal{L}$, one has $[w(\phi_1 \cup \phi'_2)]^{\mathsf{ideal}} = [w'(\phi_1 \cup \phi'_2)]^{\mathsf{ideal}}$. In particular, for all $w \in \mathrm{dom}(\phi_1)$ carrying a label $\ell \in \mathcal{L}$, since $w\theta \in \mathrm{dom}(\phi'_2)$ must carry the same label ℓ,

$$w[\phi_1]^{\mathsf{ideal}} = (w\theta)[\phi'_2]^{\mathsf{ideal}} = w[\phi_2\sigma]^{\mathsf{ideal}}.$$

Finally, since σ is a bijection on names, we have $[\phi_2]^{\mathsf{ideal}} = [\phi_2\sigma]^{\mathsf{ideal}}$ and thus $w[\phi_1]^{\mathsf{ideal}} = w[\phi_2]^{\mathsf{ideal}}$ as expected. □

Proposition 4: For any protocol and any ground configuration $\mathcal{P}$ of the protocol such that

$$(\mathcal{P}; \varnothing) \xrightarrow{\mathsf{ta}} K,$$

there exists an agent renaming ρ satisfying the hypothesis of Lemma 1 and such that $\mathcal{P}\rho$ is single-session.

Proof. We first define $\mathcal{C}\mathrm{o}(\overline{k})$ as the set of all $(\overline{n_1}, \overline{n_2})$ such that $I(\overline{k}, \overline{n_1})$ and $R(\overline{k}, \overline{n_2})$ are connected in $(\mathsf{ta}, \phi(K))$. Next, we assume for each $(\overline{k}, \overline{n_1}, \overline{n_2})$ a vector of names $k^c(\overline{k}, \overline{n_1}, \overline{n_2})$ of the length of identity parameters of Π. These name vectors are assumed to be all disjoint and not containing any name already occurring in the annotations of $\mathcal{P}$. This gives us a mean to pick fresh identity parameters for each combination of $\overline{k}, \overline{n_1}, \overline{n_2}$ taken from the annotations of $\mathcal{P}$. We also assume name vectors $k^1(\overline{k}, \overline{n_1})$ which are again disjoint and not overlapping with annotations of $\mathcal{P}$ and any $k^c(\overline{k'}, \overline{n'_1}, \overline{n'_2})$, and similarly for $k^2(\overline{k}, \overline{n_2})$ which should also not overlap with k^1 vectors. These last two collections of identity parameters will be used to give fresh identities to initiator and responder agents, independently. We then define ρ as follows:

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

By construction, agents that were connected in ta are renamed into agents sharing same identity names $k^c(\overline{k}, \overline{n_1}, \overline{n_2})$. Other agents have distinct, fresh identities. Finally, we have not used $\overline{\mathsf{id}_0}$, and the image of ρ obviously has at most one session per identity and role: our renaming is single-session. □

APPENDIX C
EXAMPLE

In this section, we give a protocol that does not preserve unlinkability according to the definition we used in this paper (see Definition 7). However, it appears that this protocol would be considered secure w.r.t. a game-based definition of unlinkability suitable for direct verification using diff-equivalence.

Description of the protocol: The protocol can be presented in Alice & Bob notation as follows:

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

The protocol is between a tag T and a reader R that share a symmetric key k. Moreover, we assume that T aborts in case the nonce n_R he receives is equal to the nonce n_T he sent previously (in the same session). We consider the term algebra introduced in Example 1, and the equational theory introduced in Example 2 with in addition the following equation:

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

Attack against unlinkability (Definition 7): To show that the property formally stated in Definition 7 does not hold, consider the following scenario.

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

A same tag starts two sessions and therefore generates two nonces n_T and n'_T. The attacker answers to these requests by sending back the two encrypted messages to the tag who will accept both of them, and sends on the network two messages that are actually equal (the exclusive or operator is commutative). Therefore the attacker observes a test (the equality between the two last messages), and this equality has no counterpart in the single session scenario. In practice, this can be very harmful when *e.g.* tags are distributed among distinct groups (*e.g.* for access control policies) sharing each the same key k. By interacting with two tags, the attacker would then be able to know if they belong to the same group.

Game-based definition: We will not give any formal definition but instead briefly give its general idea. In such a definition, the two scenarios under study will be made of two phases:

1) *Learning phase:* During this phase, the attacker can trigger an arbitrary number of sessions of the two roles (namely tag and reader) with the identity of his choice. This allows him to gain some knowledge.
2) *Guessing phase:* This phase starts once the previous one is finished. The challenger chooses an identity x among two identities id_1 and id_2, and the attacker is allowed to interact again with x (an arbitrary number of times). It may also interact with tags and readers of identities different from id_1 and id_2.

The attacker wins the game if he can infer whether x is id_1 or id_2, *i.e.* if he is able to distinguish between these two scenarios.

This is typically the kind of scenario that can be checked relying on the diff-equivalence notion implemented in several automatic tool (*e.g.* ProVerif, Tamarin). However, here we failed to prove it using ProVerif due to the $\oplus$ operator that ProVerif can not handle. The attack scenario described in the previous paragraph can be done in the guessing phase with tag id_1, and can be mimicked on the other side using two sessions of the tag with identity id_2. Actually, we believe that these two scenarios are indistinguishable, *i.e.* the resulting processes are in trace equivalence.

This example shows that game-based variants of unlinkability that are amenable to automation relying on the diff-equivalence notion is rather weak.