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

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 π calculus, notably used in the ProVerif tool. Existing tools and techniques do not allow to verify directly these properties, expressed as behavioral equivalences. We propose a different approach: we design two conditions on protocols which are sufficient to ensure anonymity and unlinkability, and which can then be effectively checked automatically using ProVerif. Our two conditions correspond to two broad classes of attacks on unlinkability, i.e., data and control-flow leaks. This theoretical result is general enough that it applies to a wide class of protocols. In particular, we apply our techniques to provide the first formal security proof of the BAC protocol (e-passport). Our work has also led 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, etc.) 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 notably extensively studied in that context (see e.g., [2] for a survey of attacks on this type of protocols) but they obviously are not limited to it.

One extremely successful approach when designing and analyzing security protocols, is the use of formal methods. The purpose of formal verification is to provide rigorous frameworks and techniques to analyze protocols. For example, a flaw has been discovered 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 are not defined as trace properties. These properties are usually defined as the fact that an observer cannot distinguish between two situations, and requires a notion of behavioral equivalence. Roughly, two protocols $P$ and $Q$ are equivalent if an attacker cannot observe any difference between $P$ and $Q$. Based on such a notion of equivalence, several definitions of privacy-type properties have been proposed (e.g., [8], [9] for unlinkability, e.g., [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 prove 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 case of equivalence properties, the existing tools scale badly and can only analyze protocols for a very limited number of sessions, typically 2 or 3. Another approach consists in implementing a procedure that is not guaranteed to terminate. This is in particular the case of ProVerif, a well-established tool for checking security of protocols. ProVerif is able to check a strong notion of equivalence (called diff-equivalence).
between processes that have the same structure. ProVerif has been recently extended [18] to conclude more often. Despite this recent effort intended to prove unlinkability of the BAC protocol (used in e-passport), ProVerif can still not 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 implyunlinkability and anonymity of the protocol under study. This approach is in the same spirit as the one presented in [9]. However, the class of protocols we target is quite different. For instance, in [9], they are only able to consider a very restricted class of protocols (single-step protocols that only use hash function as cryptographic primitives). We target more complex protocols and 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 used in the e-passport application, as well as the Password Authenticated Connection Establishment (PACE) protocol. It happens that our conditions are rather tight and each time one of our condition was not satisfied, we report on an attack. 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 one on the LAK protocol whereas it is claimed untraceable in [2].

Let us now give an intuitive overview of our 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):

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

This first attempt based on a challenge-response scheme is actually linkable. Indeed, an active attacker who systematically intercepts the nonce $n_R$ and replaces it by a constant will be able to infer whether the same tag has been used in different sessions or not by comparing the answers he receives. Here, the tag is linkable because, for a certain 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 (using a nonce $n_T$) and should achieve mutual authentication by requiring that the reader must answer to the challenge $n_T$. This protocol can be as follows:

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

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) and 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 in a protocol 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 \( \mathcal{V} \). Variables in \( \mathcal{X} \) will typically be used to refer to unknown parts of messages expected by participants, while variables in \( \mathcal{V} \) will be used to store messages learned by the attacker. We assume a signature \( \Sigma \), i.e. a set of function symbols together with their arity. The elements of \( \Sigma \) are split into constructor and destructor symbols, i.e. \( \Sigma = \Sigma_{\text{c}} \cup \Sigma_{\text{d}} \).

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

Example 1: Consider the signature

\[
\Sigma = \{\text{enc}, \text{dec}, \emptyset, \pi_1, \pi_2, @, 0, \text{eq}, \text{ok}\}.
\]

The symbols \( \text{enc} \) and \( \text{dec} \) of arity 2 represent symmetric encryption and decryption. Pairing is modeled using \( \emptyset \) of arity 2, whereas projection functions are denoted \( \pi_1 \) and \( \pi_2 \), both of arity 1. The function symbol \( @ \) of arity 2 and the constant \( 0 \) are used to model the exclusive or operator. Finally, we consider the symbol \( \text{eq} \) of arity 2 to model equality test, as well as the constant symbol \( \text{ok} \). This signature is split into two parts: \( \Sigma_{\text{c}} = \{\text{enc}, \emptyset, @, 0, \text{ok}\} \), and \( \Sigma_{\text{d}} = \{\text{dec}, \pi_1, \pi_2, \text{eq}\} \).

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 \( \equiv \) on \( \mathcal{T}(\Sigma_{\text{c}}, \mathcal{N} \cup \mathcal{X}) \), generated from a set of equations \( \mathcal{E} \) over \( \mathcal{T}(\Sigma_{\text{c}}, \mathcal{X}) \). Thus, \( \equiv \) is closed under substitutions and under bijective renaming. We finally assume that there exist \( u, v \) such that \( u \not\equiv v \).

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

\[
\begin{align*}
    x @ 0 &= x, \\
    x @ x &= 0, \\
    (x @ y) @ z &= x @ (y @ z), \\
    (x @ y) &= (y @ x).
\end{align*}
\]

In such a case, we have that \( \text{enc}(a @ (b @ a)) = \text{enc}(b, k) \).

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 \( \mathcal{T}(\Sigma_{\text{c}}, \mathcal{N}) \times \mathcal{T}(\Sigma_{\text{c}}, \mathcal{N}) \), denoted \( \downarrow \), such that:

- \( n \downarrow n \) for any \( n \in \mathcal{N} \) and, for \( f \in \Sigma_{\text{c}} \) of arity \( k \), if \( t_i \downarrow u_i \) for all \( 1 \leq i \leq k \), then \( f(t_1, \ldots, t_k) \downarrow f(u_1, \ldots, u_k) \);
- if \( t \downarrow u \) then \( t[\overline{v}] \downarrow u[\overline{v}] \) for any bijective renaming \( \rho: \mathcal{N} \rightarrow \mathcal{N} \);
- if \( t \downarrow u \) and \( t'[\overline{u}] \downarrow v' \) then \( t'[\overline{v}] \downarrow v \), for a term \( t \), a context \( t' \) (i.e. a term with a hole) built from \( \Sigma_{\text{c}}, \mathcal{N} \), and messages \( u \) and \( v \);
- if \( t_1 = t_1[\overline{u}_1], t_2 = t_2[\overline{u}_2] \) with \( u_1 \equiv u_2 \) and \( t \) a context built from \( \Sigma_{\text{c}}, \mathcal{N} \), and such that \( t_1 \downarrow v_1 \) then \( t_2 \downarrow v_2 \).

The relation \( \downarrow \) associates, to any ground term \( t \), at most one message up to the equational theory \( \mathcal{E} \). When no such message exists, we say that the computation fails; this is denoted \( t \downarrow \). 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 of the form \( g(u_1, \ldots, u_n) \rightarrow u \) where \( g \) is a destructor, and \( u, u_1, \ldots, u_n \in \mathcal{T}(\Sigma_{\text{c}}, \mathcal{X}) \). A ground term \( t \) can be rewritten into \( t' \) if there is a position \( p \) in \( t \) and a rewriting rule \( g(u_1, \ldots, u_n) \rightarrow u \) such that \( t'_p = g(v_1, \ldots, v_n) \) and \( v_1 \equiv u_1 \theta, \ldots, v_n \equiv u_n \theta \) for some substitution \( \theta \), 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: Continuing Example 1, the properties of symbols in \( \Sigma_{\text{d}} \) are reflected through the following rewriting rules:

\[
\begin{align*}
    \text{dec}(\text{enc}(x, y), y) &= x & \text{eq}(x, x) &= \text{ok} \\
    \pi_1((x, x_{2i})) &= x_i & \text{for } i \in \{1, 2\}. 
\end{align*}
\]

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

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

For modeling purposes, we split the signature \( \Sigma \) into two parts, namely \( \Sigma_{\text{pub}} \) and \( \Sigma_{\text{priv}} \). 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 \( \mathcal{T}(\Sigma_{\text{pub}}, \mathcal{W}) \). 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 \( \Sigma_{\text{c}} \cap \Sigma_{\text{pub}} \).
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.

$$
P, Q := 0 \quad \text{null} \\
| \text{in}(c, x).P \quad \text{input} \\
| \text{out}(c, u).P \quad \text{output} \\
| \text{let } \overline{x} = \overline{\pi} \text{ in } P \text{ else } Q \quad \text{evaluation} \\
| P | Q \quad \text{parallel} \\
| !P \quad \text{replication} \\
| \nu n.P \quad \text{restriction}
$$

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. $\overline{\pi}$) 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 $\text{let } \overline{x} = \overline{\pi} \text{ in } P \text{ else } Q$ that combines several standard constructions, allowing to write computations and conditionals compactly. Such a process tries to evaluate the sequence of terms $\overline{\pi}$ in case of success, i.e. when $\overline{\pi} \parallel \pi$ for some messages $\pi$, the process $P$ in which $\overline{\pi}$ are replaced by $\pi$ 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, “let $x = \text{eq}(u, v)$ in $P$ else $Q$” will execute $P$ only if the computation succeeds on $\text{eq}(u, v)$, that is only if $u \parallel v$, $v \parallel v'$, and $u' =_e v'$ for some messages $u'$ and $v'$.

For brevity, we sometimes omit “else 0” and null processes after outputs. We write $f_v(P)$ 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 $f_v(P) = \emptyset$.

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:

1. $I \rightarrow R: n_I$
2. $R \rightarrow I: \{n_I, n_R\}_k$
3. $I \rightarrow R: \{n_R, n_I\}_k$

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 $R$.

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

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

$$P_I := \text{out}(c_I, n_I).\text{in}(c_I, x_2).$$

let $x_2, x_3 = \text{eq}(n_I, \pi_1(u)), \pi_2(u)$ in

$$\text{out}(c_I, \text{enc}(x_3, n_I)).k$$

$$P_R := \text{in}(c_R, y_1).\text{out}(c_R, \text{enc}(y_1, n_R)).\text{in}(c_R, y_2).$$

let $y_3 = \text{eq}(y_2, \text{enc}(n_R, y_3), k)$ in

C. Semantics

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

- $P$ 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 $\mathcal{V}$, and $u_1, \ldots, u_n$ are messages.

We often write $P \cup P$ instead of $(P; \emptyset)$. The terms in $\phi$ 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 $\emptyset$.

The operational semantics of a process is given by the relation $\rightarrow$ 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 $\overline{\pi} = v_1, \ldots, v_n$; if this succeeds, i.e. if there exist messages $u_1, \ldots, u_n$ such that $v_1 \parallel u_1, \ldots, v_n \parallel u_n$ then variables $\overline{\pi}$ 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.

Example 5: Continuing Example 4. We have that:

$$P_{\text{RFID}} \xrightarrow{\text{tr}} (\emptyset; \phi_0)$$

where $\text{tr}$ and $\phi_0$ are as follows, for fresh $k', n'_I$ and $n'_R$:

$$\text{tr} = \{ \tau.\tau.\tau.\tau.\text{out}(c_I, w_1).\text{in}(c_R, w_1).\text{out}(c_R, w_2).\text{in}(c_I, w_2).\text{out}(c_I, w_3).\text{in}(c_R, w_3).\text{then} \}$$

$$\phi_0 = \{ w_1 \mapsto n'_I, w_2 \mapsto \text{enc}(\{n'_I, n'_I\}, k'), w_3 \mapsto \text{enc}(\{n'_R, n'_I\}, k') \}.$$
in (in(c, x).P ∪ P; φ) \mapsto (in(c, R).P(x → u) ∪ P; φ) where R is a recipe such that Rφ∥u for some message u

out (out(c, u).P ∪ P; φ) \mapsto (out(c, w).P ∪ P; φ ∪ \{w → u\}) with w a fresh variable in \mathcal{W}.

let (let \overline{p} = \overline{p} in P else Q ∪ P; φ) \mapsto (P[\overline{p} → \overline{p}] ∪ P; φ)

let-fail (let \overline{p} = \overline{p} in P else Q ∪ P; φ) \mapsto (Q ∪ P; φ)

new (\nu n.P ∪ P; φ) \mapsto (P ∪ P; φ)

repl (!P ∪ P; φ) \mapsto (P ∪ !P ∪ P; φ)

par ((\{P_1 \mid P_2\} ∪ P; φ) \mapsto ((\{P_1, P_2\} ∪ P; φ)

\fig{2}{Semantics for processes}

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 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 dom(φ) = dom(φ'), and

- for any recipe R such that Rφ∥u for some u, we have that Rφ∥u' for some u';
- for any recipes R_1, R_2 such that R_1φ∥u_1, R_2φ∥u_2, and \u_1 \equiv u_2, we have that R_1φ∥u_1 \equiv R_2φ∥u_2, i.e. there exist \u_1, \u_2 such that R_1φ∥\u_1, R_2φ∥\u_2, and \u_1 \equiv \u_2.

Two frames φ and φ' are in static equivalence, written φ \sim φ', 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 φ_0 ∪ \{w_4 → k'\} \not\sim φ_0 ∪ \{w_4 → k''\}. Indeed, the attacker may observe that the computation R = dec(\u_2, \u_4) succeeds in φ ∪ \{w_4 → k'\} but fails in φ ∪ \{w_4 → 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 = (P; φ), we define trace(K):

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

We define obs(tr) to be the subsequence of tr obtained by erasing all the τ actions (i.e. τ, τ_{then}, τ_{else}).

Definition 3: Let K and K' be two configurations. We say that K is trace included in K', written K \subseteq K', when, for any (tr, φ) ∈ trace(K) there exists (tr', φ') ∈ trace(K') such that obs(tr) = obs(tr') and φ \sim φ'. They are in trace equivalence, written K \approx K', when K \subseteq K' and K' \subseteq K.

Example 7: We may be interested in checking whether K = (!P_h; \phi) and K' = (\nu k.(\nu n_1.P_I \mid \nu n_2.P_{I'}; \phi)) are in trace equivalence. Intuitively, this equivalence models the fact that P_h 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 : \text{out}(c, u).P_R

where c ∈ \mathcal{C}, u ∈ T(\Sigma, \mathcal{N} ∪ \mathcal{X}), and P_R is obtained from the grammar of responder roles:

P_R ::= 0 \mid \text{in}(c, y).let \overline{p} = \overline{p} in P_I else 0 \mid \text{in}(c, y).let \overline{p} = \overline{p} in P_I else \ell : \text{out}(c', u')

where c, c' ∈ \mathcal{C}, y ∈ \mathcal{X}, \overline{p} (resp. \overline{p}') is a (possibly empty) sequence of variables in \mathcal{X} (resp. terms in T(\Sigma, \mathcal{N} ∪ \mathcal{X})), u' ∈ T(\Sigma, \mathcal{N} ∪ \mathcal{X}), and \ell ∈ \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 $\pi, \pi$.

Example 8: Continuing our running example, $P_1$ (resp. $P_R$)
as defined in Example 4 is an initiator (resp. responder) role,
up to the addition of trivial conditionals and distinct labels $\ell_1$,
$\ell_2$, and $\ell_3$ to decorate the output actions that occur in the two
roles.

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 $\phi$ if $\tau_{else} \notin tr$ and $\text{obs}(tr)$ is of the form
\[
\text{out}(\_ \_ w_0) \cdot \text{in}(\_ \_ R_0) \cdot \text{out}(\_ \_ w_1) \cdot \text{in}(\_ \_ R_1) \ldots
\]
for arbitrary channel names, and such that $\forall R_i, \phi \vdash \models E w_i \phi$ for
any action $\text{in}(\_ \_ R_i)$ 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
(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 $u \in E v$
for some $v \in T(\Sigma_c \cap \Sigma_{\text{pub}}, \emptyset)$. 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 $\Pi$ is a tuple $(k, \pi_I, \pi_R, I, R)$
where $k, \pi_I, \pi_R$ are three disjoint sets of names, $I$ (resp. $R$)
is an initiator (resp. responder) role such that $\forall u, \pi_I$ (resp. $\forall u, \pi_R$) $u \notin k \cup \pi_I$ (resp. $\forall u, \pi_R$) $u \notin k \cup \pi_R$). Labels of $I$ and $R$ must be pairwise
distinct. Names $k$ (resp. $\pi_I \cup \pi_R$) are called identity names
(resp. session names).

Let $P_I = \nu k, (\nu \pi_I, I) \models \nu \pi_R, R$. We assume that $P_I \xrightarrow{\text{tr}_h} (\emptyset, \phi_h)$ for some frame $\phi_h$ that does not contain any public
message, and some trace $\text{tr}_h$, that is honest for $\phi_h$.

Example 9: Let $\Pi = (k, n_I, n_R, P_I, P_R)$ as defined in Example 4. We have already seen that $P_I$ is an initiator role whereas $P_R$ is a responder role. Let $P_I = \nu k, (\nu n_I, P_I) \models \nu n_R, P_R$. Let $\text{tr}_h = \text{tr}$, and $\phi_h = \phi_0$
as defined in Example 5. They satisfy the requirements stated
in Definition 6, and therefore $\Pi$ 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 $\Pi$, as defined above, we denote $M_{\Pi I}$
the process that represents an arbitrary number of agents that may
possibly execute an arbitrary number of sessions, whereas $S_{\Pi I}$
represents an arbitrary number of agents that can at most
execute one session each. Formally, we define:
\[
M_{\Pi I} := !\nu k, (\nu n_I, I) \mid !\nu n_R, R; \quad S_{\Pi I} := !\nu k, (\nu n_I, I) \mid !\nu n_R, R.
\]

a) Unlinkability: Informally, a protocol preserves unlinkability w.r.t. the roles $I$ and $R$ 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, would allow the roles $I$ and $R$ 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 $\Pi = (k, \pi_I, \pi_R, I, R)$ be a protocol. We
say that $\Pi$ preserves unlinkability if $M_{\Pi I} \approx S_{\Pi I}$.

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 $\overrightarrow{id} \in E$, we introduce the following process:
\[
M_{\Pi I}^{id} := M_{\Pi I} \mid \nu k, (\nu n_I, I_0) \mid \nu n_R, R_0,
\]
where $I_0 = I \{ \overrightarrow{id} \mapsto \overrightarrow{id} \}$, $R_0 = R \{ \overrightarrow{id} \mapsto \overrightarrow{id} \}$, and $\overrightarrow{id}$ are fresh
constants from $\Sigma_c \cap \Sigma_{\text{pub}}$ (i.e. not used in $\Pi$). In this process, in
addition to the arbitrary number of agents that may execute an
arbitrary number of sessions, there are two agents $I_0$ and $R_0$
that have disclosed (part of) their identity $\overrightarrow{id}$ to the attacker,
and that may also execute an arbitrary number of sessions.

Definition 8: Let $\Pi = (k, \pi_I, \pi_R, I, R)$ be a protocol, $\overrightarrow{id} \in E$.
We say that $\Pi$ preserves anonymity w.r.t. $\overrightarrow{id}$ if $M_{\Pi I} \approx M_{\Pi I}^{id}$.

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

Discussion: A flurry of alternative definitions of un-linkability 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 main 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 event has occurred, 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 \( A(\bar{k}, \pi) \) where \( A \in \{I, R\} \). An annotated action is either \( \tau \) or \( \alpha[a] \) where \( \alpha \) is an annotation, \( "\tau" \) is an action other than \( \tau \) (possibly \( \tau_{\text{then}} \) or \( \tau_{\text{else}} \)) 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 \( II = (\bar{k}, \pi, \pi_R, I, R) \), consider any execution of \( \mathcal{M}_{II}^{\text{id}}, \mathcal{M}_{II} \) or Domain. In such an execution, \( \tau \) 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 \( I \) and \( R \) with fresh names. Actions other than \( \tau \) (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 \( \tau \) 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 ouptutted term so that we can refer to them when needed.

**Example 10:** Considering the protocol of Example 9, process \( S_{II} \) can essentially perform the execution seen in Example 5. The annotated execution has the trace \( ta \) given below, where \( k', n'_1 \) and \( n'_R \) are fresh names, \( a_I = I(k', n'_1) \) and \( a_R = R(k', n'_R) \):

\[
\begin{align*}
&\begin{array}{c}
&\text{ta} = \tau.\tau.\tau.\tau.\tau.\\text{out}(c_1, w_1)(a_I).\text{in}(c_R, w_1)(a_R).\
&\quad \text{out}(c_R, w_2)(a_R).\text{in}(c_I, w_2)(a_I).\tau_{\text{then}}[a_I].\
&\quad \text{out}(c_I, w_3)(a_I).\text{in}(c_R, w_3)(a_R).\tau_{\text{then}}[a_R].
\end{array}
\end{align*}
\]

After the initial \( \tau \) actions, the annotated configuration is 

\[
\{I \sigma_I(a_I), R \sigma_R(a_R), S_{II}\}; \phi_0
\]

where \( \sigma_I \) is \( \{k \mapsto k', n_1 \mapsto n'_1\} \) 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 \( \phi_0 \) are decorated with \( \ell_1, \ell_2 \) and \( \ell_3 \) respectively.

Note that annotations of the specific agents whose identity contains constants \( id_\alpha \) will contain those constants (i.e. they are of the form \( A'\overline{k}, \pi) \) with \( id_\alpha \in \overline{k} \).

B. Frame opacity

In light of attacks based on leakage from messages where non-trivial relations between outputted messages are exploited by the attacker to trace an agent, our first condition will 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 \( \Sigma_t \subseteq \Sigma_c \cap \Sigma_{\text{pub}} \) is said to be transparent if it satisfies the following conditions:

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

In the rest of our theoretical development, we assume an arbitrary transparent set \( \Sigma_t \). Our results are stronger with a larger set, but still hold if some constructor symbols fail to be identified as transparent.

**Example 11:** In the signature of Example 1, the largest set of transparent constructors is \( \{\langle\rangle, 0, \text{ok}\} \).

We now define the idealization of (the observable parts of) messages and frames: we first replace non-transparent subterms by holes (denoted by \( \square \), 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

\[
[u]^{\text{ideal}} : \mathcal{T}(\Sigma_c, N) \rightarrow \mathcal{T}(\Sigma_t, \{\square\})
\]

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

**Definition 10:** A concretization of \( u_t \in \mathcal{T}(\Sigma_t, \{\square\}) \) is any term obtained by replacing each hole of \( u_t \) by a fresh nonce. We denote by \( \text{inst}(u_t) \) the set of all concretizations of \( u_t \). Finally, for a message \( u \), we let \( [u]^{\text{nonce}} \) be the set \( \text{inst}\{[u]^{\text{ideal}}\} \).

Those definitions are extended to frames in the 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 \( u' \in [u]^{\text{nonce}} \) (resp. \( \phi' \in [\phi]^{\text{nonce}} \)), no nonce appears twice in \( u' \) (resp. \( \phi' \)), and therefore for all frames \( \psi, \phi_1, \phi_2 \in [\psi]^{\text{nonce}} \), one has \( \phi_1 \sim \phi_2 \).

**Example 12:** Let \( u \) be \( \langle n_1, \text{enc}(\text{ok}, n_3), k \rangle \). We have that \( [u]^{\text{ideal}} = \langle \square, \square \rangle \) and \( [u]^{\text{nonce}} = \{\langle n_1, n_2 \rangle, n_1 \neq n_2 \in N\} \).

We are now ready to state our first condition:
Definition 11: The protocol $\Pi$ ensures frame-opacity if, for any execution $(M^H_\Pi; \phi) \xrightarrow{\tau} (Q; \phi)$, we have that:
1) $\phi \sim \psi$ for some $\psi \in [\phi]_{\text{nonce}}$, and
2) for any $w_i, w_j \in \text{dom}(\phi)$ that carry the same label $\ell \in \mathcal{L}$, we have that $[w_1 \phi]_{\text{ideal}} = [w_2 \phi]_{\text{ideal}}$.

Example 13: Consider the frame $\phi_0$ as defined in Example 5. We have that $[\phi_0]_{\text{ideal}} = \{w_1 \mapsto \square, w_2 \mapsto \square, w_3 \mapsto \square\}$. We have that $\phi_0 \sim \phi$ for any $\phi \in [\phi_0]_{\text{nonce}}$.

However, in case, $n'_R = n'_1$, static equivalence between $\phi_0$ and its idealized version $[\phi_0]_{\text{nonce}}$ 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 conditions. 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 let $\pi = \tau \in P$ else $Q$ occurring in $A \in (\mathcal{I}, \mathcal{R})$ is safe if $\pi \in \mathcal{T}(\Sigma_{\text{pub}} \cup \{x_1, \ldots, x_n\} \cup \{u_1, \ldots, u_m\})$, where the $x_i$ are the variables bound by the previous inputs of that role, and $u_i$ are the messages used in the previous outputs of that role.

Example 14: Consider the process given below:

\[
\text{out}(c, u). \text{in}(c, x). \text{let } z = \text{neq}(x, u) \text{ in } P \text{ else } Q
\]

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 the grammar forced us to add 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 $ta|_{a,a'}$ the subsequence of $ta$ that consists of actions of the form $o[\alpha]$ or $o[\alpha']$.

Definition 13: Two agents $A_1(\mathcal{K}_1, \pi_1)$ and $A_2(\mathcal{K}_2, \pi_2)$ are associated in $(ta, \phi)$ if:
- the agents are dual, i.e. $A_1 \neq A_2$ and $\mathcal{K}_1 = \mathcal{K}_2$;
- the interaction $ta|_{A_1(\mathcal{K}_1, \pi_1), A_2(\mathcal{K}_2, \pi_2)}$ is honest for $\phi$.

Example 15: Continuing Example 10, the agents $I(k', n'_i)$ and $R(k', n'_R)$ are associated in $(ta, \phi_0)$.

We can finally state our second condition:

Definition 14: The protocol $\Pi$ is well-authenticating if, for any execution $(M^H_{\Pi}; \phi) \xrightarrow{\tau \eta_{\text{in}}} (A(\mathcal{K}, \pi_1)) (P; \phi)$ either the last action corresponds to a safe conditional, or there exists $A'$ and $\pi_2$ such that (i) $A(\mathcal{K}, \pi_1)$ and $A'(\mathcal{K}, \pi_2)$ are associated in $(ta, \phi)$, and (ii) $A'(\mathcal{K}, \pi_2)$ is only associated with $A(\mathcal{K}, \pi_1)$ in $(ta, \phi)$.

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 systematically 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 $\Pi = (\mathcal{K}, \pi_1, \pi_2, \mathcal{I}, \mathcal{R})$ and some identity names $\overline{id} \in \mathcal{K}$. If the protocol is well-authenticating and ensures frame opacity, then $\Pi$ ensures unlinkability and anonymity w.r.t. $\overline{id}$.

Note that, since $M_{\Pi}^H = M_{\Pi}$ when $\overline{id} = \overline{\phi}$, we have as a corollary that if $M_{\Pi}$ ensures well-authentication and frame opacity, then $\Pi$ is unlinkable.

Before establishing this result 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. Our tool concludes within 1 minute for the first five examples, and it takes a bit more time (around 20 minutes) to conclude on the other examples.

V. PROOFS

We provide in this section the proof of Theorem 1. Our main argument consists in showing that, for any execution of $M_{\Pi}^H$, there is an indistinguishable execution of $S_{\Pi}$.

Instead of working with $M_{\Pi}^H$, $M_{\Pi}$ and $S_{\Pi}$, 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 $M_{\Pi}^H$, $M_{\Pi}$ or $S_{\Pi}$ by launching a few sessions (performing $\tau$ 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 \(A(\overline{k}, \overline{m})\), the session parameters \(\overline{m}\) are names and the identity parameters \(\overline{k}\) are made of names or constants \(\text{dom}_0\);

- 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 \(\mathcal{P}_I \cup \mathcal{P}_R\) where

\[
\mathcal{P}_I = \{ I(\overline{k} \mapsto \overline{l}, \overline{m} \mapsto \overline{m}) | I(\overline{l}, \overline{m}) \in S \}
\]

and similarly for \(\mathcal{P}_R\), 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 \(A(\overline{k}, \overline{m})\) and \(A(\overline{k}, \overline{m})\) occur in \(\mathcal{P}\) then \(\overline{m} = \overline{m}\) and \(\overline{m}\) does not occur in it. Any ground configuration can be reached from \(\mathcal{M}_I\); single-session ground configurations are those which can also be obtained from \(\mathcal{S}_I\).

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

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

Note that the renaming on parameters induced by a renaming of agents may conflict: this happens, for example, when \(\rho(A(\overline{k}, \overline{m})) = A(\overline{k_1}, \overline{m})\) 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 \(\phi\). However, given an execution \(ta\) that yields \(\phi\), each handle \(w \in \text{dom}(\phi)\) is uniquely associated in \(ta\) to an output, and thus an agent \(a_w\). We can then define \(\phi\rho\) (omitting the mention of \(ta\) as a slight abuse of notation) as

\[
\{ w \mapsto w\sigma | w \in \text{dom}(\phi), \sigma\text{ induced by }\rho\text{ on }a_w \}.
\]

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 \(I\) be a well-authenticating protocol, and \(\mathcal{P}\) a ground configuration of \(I\) such that

\[
(P; \phi) \xrightarrow{ta; \tau_x[A(\overline{k}, \overline{m})]} (P'; \phi)
\]

and the last action is performed by an unsafe conditional. We have \(\tau_x = \tau_{\text{then}}\) iff there exists \(\overline{m}_2\) such that \(\overline{A}(\overline{k}, \overline{m}_2)\) is associated to \(\overline{A}(\overline{k}, \overline{m})\) in \((ta, \phi)\).

**Proof sketch.** The \(\Rightarrow\) direction is a direct consequence of well-authentication. For the other direction, we essentially 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 \(I\) modulo a renaming of parameters, thus the considered test must be successful. Assuming that \(a_1 = A(\overline{k}, \overline{m}_1)\) and \(a_2 = A(\overline{k}, \overline{m}_2)\) are associated in \((ta, \phi)\), we shall prove that \(\tau_x = \tau_{\text{then}}\). By hypothesis, \(ta|_{a_1, a_2}\) is honest: its observable actions are of the form \(\text{out}(c_1, w_1), \text{in}(c_1', M_1), \ldots, \text{out}(c_n, w_n), \text{in}(c_n', M_n)\) with possibly an extra output at the end, such that \(M_i\phi = \bigwedge w_i \phi\) for all \(1 \leq i \leq n\). Consider \(ta'\) obtained from \(ta\) 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 \((P; \phi)\) and yields the same configuration \((P'; \phi)\). But now \(ta'_|_{a_1, a_2}\) is a self-contained execution, i.e. if \(P\) and \(Q\) are the processes respectively annotated \(a_1\) and \(a_2\) in \(P\), we have

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

In that execution, everything is deterministic (up to the theoretical) and thus the execution is actually a prefix of the honest execution of \(I\), up to a 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 \(\tau_{\text{then}}\). \(\Box\)

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 (do so names) and thus frames are modified. However, the last requirement of 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. Note that, by frame opacity again, this implies that the frames are statically equivalent.

**Proposition 3:** Let II be a protocol ensuring frame opacity. Let \( P \) be a ground configuration of II, \( \tau a \) an annotated trace, and \( \rho \) an arbitrary renaming of agents. If \( (P; \emptyset) \nrightarrow^\tau (P_1; \phi_1) \) and \( (P; \emptyset) \nrightarrow^\tau P_2; \phi_2 \), then \([\phi_1]_\text{ideal} = [\phi_2]_\text{ideal}\).

**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 \( (\tau a, \phi) \) if they are associated in \( (\tau a_\emptyset, \phi) \) for some prefix \( \tau a_\emptyset \) of \( \tau a \) such that \( \tau a_{a,a'} \) contains at least one \( \tau \) then action of an unsafe conditional.

**Lemma 1:** Let II be a well-authenticating protocol ensuring frame opacity, and \( \tau a \) be an annotated trace executed by some ground configuration \( P \):

\[
(P; \emptyset) \nrightarrow^\tau K
\]

Let \( \rho \) be a renaming of agents whose domain contains the annotations of \( P \), and such that \( \rho(a) \) and \( \rho(a') \) are duals iff \( a \) and \( a' \) are connected in \( (\tau a_\emptyset, \phi(K)) \). Then we have:

\[
(P; \emptyset) \nrightarrow^\tau_\rho K_\rho \quad \text{and} \quad \phi(K) \sim \phi(K_\rho).
\]

**Proof.** We shall focus on establishing that \( \tau a_\rho \) is executable; once this is known, static equivalence is a direct consequence of Proposition 3. We thus prove that, for any prefix \( \tau a_\emptyset \) of \( \tau a \), we have

\[
(P; \emptyset) \nrightarrow^\tau_\rho \tau K_\rho
\]

with an additional invariant: \( \rho(a) \) and \( \rho(a') \) are associated in \( (\tau a_\rho, \phi(K_\rho)) \) iff \( a \) and \( a' \) are associated in \( (\tau a_\emptyset, \phi(K)) \) and connected in \( (\tau a, \phi(K)) \).

We proceed by induction on \( \tau a_\emptyset \). If it is empty, then \( \tau a_\rho \) can also obviously be executed. For empty traces, association coincides with duality, thus the hypothesis on \( \rho \) implies our invariant.

Consider now a prefix of \( \tau a \) of the form \( \tau a_\rho, \alpha[a] \). By induction hypothesis we have \( K_\rho \) (resp. \( K_\rho_\epsilon \)) resulting from the execution of \( \tau a_\emptyset \) by \( P \) (resp. \( \tau a_\rho \) by \( P_\rho \)) and our invariant satisfied for \( \tau a_\emptyset \). Moreover, by Proposition 3, we know that \( \phi(K_\rho) \sim \phi(K_\rho_\epsilon) \). The action \( \alpha \) performed by the process annotated \( a \) in \( P \) may be an input, an output, or a test. In any case, the corresponding process in \( P_\rho \) can perform an action of the same nature. To conclude, we distinguish the three kinds of actions:

- If \( \alpha \) is an output, we only have to check our invariant for \( \tau a_\rho, \alpha[a] \). 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 \( (\tau a_\rho, \alpha[a]) \rho \) iff they are associated in \( \tau a_\rho \), and similarly without \( \rho \).
- If \( \alpha \) 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 \( \tau a_\emptyset \) iff \( \rho(a) \) and \( \rho(a') \) are associated in \( \tau a_\rho \). This is simply because, if \( a \) and \( a' \) are associated in \( \tau a_\emptyset \), then they are having an honest interaction, thus the outcome of the test will be positive, and \( a \) and \( a' \) are connected in \( \tau a \).

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.

\( \square \)

**D. Proof of Theorem 1**

Thanks to our lemma, we can change any execution of \( \mathcal{M}_\Pi^{\text{id}} \) 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 \( P \) of the protocol such that

\[
(P; \emptyset) \nrightarrow^\tau K,
\]

there exists an agent renaming \( \rho \) satisfying the hypothesis of Lemma 1 and such that \( P_\rho \) 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 and complete proof can be found in Section B. \( \square \)

We can now easily conclude.

**Proof of Theorem 1.** It is easy to see that \( \mathcal{S}_\Pi \subseteq \mathcal{M}_\Pi \subseteq \mathcal{M}_\Pi^{\text{id}} \), so it only remains to establish \( \mathcal{M}_\Pi^{\text{id}} \subseteq \mathcal{S}_\Pi \). Consider an execution \( \mathcal{M}_\Pi^{\text{id}} \nrightarrow^\tau K \). Without loss of generality we can assume that session creations are all performed at the beginning of \( \tau a \), i.e. it is of the form \( \tau^* \cdot \tau a' \) with no occurrence of \( \tau \) in \( \tau a' \); otherwise we can modify \( \tau a \) to satisfy this condition, without
changing its observable actions and the resulting frame. Thus we have a ground configuration $P$ of $H$, such that

$$(P;\emptyset) \xrightarrow{ta'} K.$$ 

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

$$(S_{H};\emptyset) \xrightarrow{\tau'} (P\rho \cup S_{H};\emptyset) \xrightarrow{ta'\rho} K\rho$$

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

VI. MECHANISATION

We now discuss how to delegate the verification of frame-opacity and well-authentication to a fully automatic tool. We show that it is possible to use ProVerif [30] to do so. Everything that is explained in this section has 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. 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 [32]. For each role $\varLambda \in \{I,R\}$, 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 $Out_{\varLambda_i}(k,\overline{n},m)$ and $In_{\varLambda_j}(k,\overline{n},m)$, whose arguments contain:

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

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

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 the condition of Definition 14 (i). It can be expressed as a correspondence property using events as explained next. Given a role $\varLambda \in \{I,R\}$ and a conditional of this role whose event is $Test_{\varLambda_i}(k,\overline{n})$, 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 $Test_{\varLambda_i}(k,\overline{n})$ is fired,
2) there must be a previous event $In_{\varLambda_j}(k,\overline{n},m)$ ($In_{\varLambda_j}$ corresponding to the input just before the conditional),
3) and a previous event $Out_{\varLambda_k}(k,\overline{n'},m')$ ($Out_{\varLambda_k}$ corresponding to the output that fed the input $In_{\varLambda_j}$ in the honest execution),
4) and a previous event $In_{\varLambda_l}(k,\overline{n'},m')$ ($In_{\varLambda_l}$ corresponding to the first input before $Out_{\varLambda_k}$),
5) and a previous event $Out_{\varLambda_n}(k,\overline{n},m')$ ($Out_{\varLambda_n}$ corresponding to the output that fed the input $In_{\varLambda_l}$ 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 our equational theory $\Sigma$.

Example 16: Those kinds of correspondence properties are better explained by showing the ProVerif code we produce. We depict in Figure 4 the query we produce for checking well-authentication (i) on the first conditional of $P_I$ from our running example.

```proverif
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. 4. Example of ProVerif query for checking well-authentication

B. Frame Opacity

Assuming well-authentication (i), we now explain how to check frame opacity using the diff-equivalence feature of ProVerif [33]. Diff-equivalence is a property of bi-processes. A bi-process is a process in which some terms are replaced by bi-terms, denoted $choice[u_1, u_2]$. 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.

Frame opacity (see Definition 11) requires that for any execution $M^I_{id}[\phi]$ and a modified version of this process where each syntactical output $u$ (identified by a label) has been replaced by its static idealization, i.e. the idealization of some message that this output may produce.

We first explain how to determine, for each syntactical output $\ell : \text{out}(c, u)$, its static idealization $u_{\ell}^{\text{ideal}}$. If the output is not in an else branch, then it is executed in the honest execution producing some message $m$. In that case we set $u_{\ell}^{\text{ideal}} = [m]^{\text{ideal}}$. Otherwise, the output is the only action of the else branch of some conditional executed (positively) in the honest execution. Since we assume well-authentication (i), it is possible to reach that else branch (and thus this output) by choosing a recipe which breaks any association for the input
just before that conditional. We can thus obtain an execution performing this output, producing some message \( m \), and we set \( u_{\ell}^{\text{ideal}} = [m]^{\text{ideal}} \).

Given a protocol II ensuring well-authentication (i), we check that frame opacity holds by checking diff-equivalence for some bi-process \( \text{biproc}(M_{\text{id}}^2) \). As a first approximation, the bi-process is defined from \( M_{\text{id}}^2 \) by replacing each \( \ell : \text{out}(c, u) \) by \( \text{out}(c, \text{choice}[u, u_{\text{nonce}}]) \) where \( u_{\text{nonce}} \) is obtained from \( u_{\ell}^{\text{ideal}} \) by filling its holes with fresh names. A crucial point is to consider fresh names from messages \( u_{\text{nonce}} \) as new session names of the bi-process so that they will be different for each session. The only remaining problem 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 \( \text{biproc}(M_{\text{id}}) \) 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 this point: we show in Figure 5 (part of) the bi-process resulting from the application of our transformation to our running example. More examples can be found in the ProVerif files associated to our case studies, available online [31].

Assuming that diff-equivalence holds for \( \text{biproc}(M_{\text{id}}^2) \), frame opacity holds. Indeed, for any execution \( M_{\text{id}}^2 \xrightarrow{\tau} (P, \phi) \), there exists some execution \( \text{biproc}(M_{\text{id}}^2) \xrightarrow{\tau} (Q, [\phi_l, \phi_r]) \) with \( \phi_l \sim \phi_r \). By construction of the bi-process, we have \( \phi_r \in [\phi_l]^{\text{nonce}} \), which implies item (1) of frame opacity, and the idealization only depends on the output labels, which implies item (2).

Finally, we will see that checking condition (ii) of well-authentication is trivial once the other conditions have been checked. More precisely, we claim that if well-authentication (i) and frame opacity hold, then well-authentication (ii) holds if, and only if, the first input of the responder \( R \) is not immediately followed by an unsafe test. Indeed, if that test of \( R \) is unsafe, condition (ii) is immediately broken by considering that the first output of some agent \( Z(k, \pi) \) can be fed to the first input of two readers \( R(k, \pi_1) \) and \( R(k, \pi_2) \), both of which will pass the test. Conversely, assume that the first test of \( R \) is safe. Consider an execution where two agents \( a \) and \( a' \) are associated and \( a' \) has performed an unsafe test. 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 \( \ell \) 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 \( \ell \) 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 the ProVerif tool (as explained in Section VI) to check automatically whether the protocol under study satisfies frame opacity and well-authentication as defined in Section IV. All ProVerif files can be found in [31]. We also discuss some variations of the protocols to examine how privacy is affected.

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.

B. Hash-Lock protocol

We consider an RFID protocol, called Hash-Lock (more precisely the Improved Randomized Hash-Locks), as described in [34] 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.

\[
\begin{align*}
\text{Reader} & \rightarrow \text{Tag} : & n_R \\
\text{Tag} & \rightarrow \text{Reader} : & n_T, h(n_R, n_T, k)
\end{align*}
\]

This protocol falls into our generic class of 2-party protocols, and frame opacity and well-authentication can be established using ProVerif. 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 \( h(k) \) (they always store the two last keys). Our framework does not allow one to model protocol that rely on a mutable
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 \((I(Tag))\) by injecting previously eavesdropped messages.

\[
\begin{align*}
I(Reader) &\rightarrow Tag : \quad r_1 \\
Tag &\rightarrow I(Reader) : \quad r_2, h(r_1 \oplus r_2 \oplus k) \\
Reader &\rightarrow I(Tag) : \quad r'_1 \\
I(Tag) &\rightarrow Reader : \quad r_2', h(r_0 \oplus r_1 \oplus k) \\
Reader &\rightarrow I(Tag) : \quad h(h(r_0 \oplus r_1 \oplus k) \oplus k \oplus r'_1)
\end{align*}
\]

where \( r'_2 = r'_1 \oplus r_1 \oplus r_2 \), thus \( h(r_1 \oplus r_2 \oplus k) = h(r'_1 \oplus r'_2 \oplus k) \).

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 \( \tau_{then} \) 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 \( I(Tag) \) 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 ProVerif. 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 Tag and 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.

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 \((m, mac(m, k))\). 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 left instructions, and therefore satisfy our definition of being a responder role. Then, we established frame opacity and well-authentication relying on the ProVerif tool. 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

\(^1\)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.
We consider the signature given in Example 1 augmented the two cases of failure, i.e. this symbol. We also assume public constants to model error mac key. There is no rewriting rule and no equation regarding code, taking as arguments the message to authenticate and the {

\[
T(k_E, k_M) = \nu T, \nu k_T. \text{out}(c_T, n_T). \text{in}(c_T, x).
\]

let \(x = \pi_1(x), x_M = \pi_2(x), \text{eq}(x, \text{mac}(x_E, k_M))\) in
let \(\text{error} = \text{eq}(\nu T, \pi_1(\text{dec}(x_E, k_E))))\) in
let \(\text{else out} \text{(error}_{\text{mac}})\) otherwise

We consider the signature given in Example 1 augmented with a function symbol \(\text{mac}\) of arity 2. This is a public construct 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. \(\text{error}_{\text{mac}}\) and \(\text{error}_{\text{nonce}}\) are the same constant, whereas the French version does.

![Fig. 6. Description of the BAC protocol](image)

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.).

**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 proof about privacy were 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 steps:

- 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 \(g^{n_R, n_T}\).

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

![Fig. 7. PACE in Alice & Bob](image)

- 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).

More specifically, 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:

\[\Sigma_c = \{\text{enc, dec, dh, mac, gen, g, ok}\}\]
\[\Sigma_d = \{\text{neq}\}\]

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

\[\text{dec}(\text{enc}(x, y), y) = x\]
\[\text{dh}(\text{dh}(x, y), z) = \text{dh}(\text{dh}(x, z), y)\]

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

\[\Pi_{\text{PACE}} = (k; \{s_T, n_T, n'_T\}, \{n_R, n'_R\}, I_{\text{PACE}}, \mathcal{R}_{\text{PACE}})\]

where the \(\mathcal{R}_{\text{PACE}}\) 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 \(I_{\text{PACE}}\) can be obtained in a similar way.

Unfortunately, ProVerif cannot handle the equation above on the \(\text{dh}\) operator (due to some termination issues). Instead, we consider the following equational theory that ProVerif handle, where \(g\) is a constant:

\[\text{dh}(\text{dh}(g, y), z) = \text{dh}(\text{dh}(g, z), y)\]
\[\text{dh}(\text{dh}(\text{gen}(x_1, x_2), y), z) = \text{dh}(\text{dh}(\text{gen}(x_1, x_2), z), y)\]

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

Firstly, 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 in $G = \text{gen}(\text{dec}(y_1, k), \text{dh}(y_2, n_R))$ 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 $c^0 = \text{enc}(s^0_T, k^0)$ of a certain tag $T^0$ and then, in a future session, it would let the reader optically scan a tag $T^0$ but replace its challenge $\text{enc}(s_T, k)$ by $c^0$. Whatever key $k$ is equal or $k^0$, the reader answers $g^{s^0}$. The attacker then plays the two rounds of Diffie-Hellman by reusing messages from the reader (he actually performs reflection attacks). More precisely, he replies with $g^{s_T} = g^{s^0}$, $G^{g^{s_T}} = G^{g^{s^0}}$ and $\text{mac}(G^{g^{s_T}}, k') = \text{mac}(G^{g^{s_0}}, k')$. 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 $\text{enc}(s_T, k)$ from a tag $T$ ($T'\text{ provides to two readers}$ $R(k, n^1_R), R(k, n^2_R)$ 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 $S_{II}$; if the attacker makes two readers talk to each other in $S_{II}$ 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 two readers, an attacker may learn which ones it had initiated by forwarding messages from one to the other. 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 $\text{mac}((c_t, G^{c_t}), k')$ and $\text{mac}((c_t, G^{c_t}), k')$ where $c_t, c_t$ are public constants, and adding the corresponding checks. Frame opacity and well-authentication can be automatically established using our tool UKano. 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. 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
e-passport protocols. 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 two-party protocols. We could also try to translate our conditions into more comprehensive guidelines helping the design of new privacy-enhancing protocols.

We also identify a number of research problems aimed at generalizing the impact 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-automation, we could consider various extensions of ProVerif with partial support for xor [41], or other tools such as Tamarin and Maude-NPA. We also would 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 proof, signature).

REFERENCES

We detail below how to obtain Proposition 1.

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

Proposition 5: For any \( u \in \Sigma_i \) with \( f \in \Sigma_i \), we have \( h_i(u) \leq h_i(u) \) for all \( i \).

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

Definition 18: The relation \( R_{\text{ideal}} : (T(\Sigma, \mathcal{N})) \times T(\Sigma, \{ e \}) \) is the least relation such that:

- \( u R_{\text{ideal}} f(t_1, \ldots, t_n) \) if there exist \( f \in \Sigma_i \) and messages \( u_i \) for \( 1 \leq i \leq n \) and \( u \in \Sigma_i \) such that \( u = E f(u_1, \ldots, u_n) \).
- \( u R_{\text{ideal}} t_0 \) for all \( 1 \leq i \leq n \).

Proposition 6: For all \( u \) there exists a \( v \) such that \( u R_{\text{ideal}} v \). Furthermore, whenever we have \( u \in E u' \), \( u R_{\text{ideal}} v \), and \( u R_{\text{ideal}} v' \), it must be that \( v = v' \).

Proof. We proceed by induction over \( h_i(u) \). If \( u \) cannot be equated to a message with a transparent function symbol at toplevel, then the result is obvious with \( v = \{ e \} \). Otherwise, assume \( u = E f(u_1, \ldots, u_n) \). By induction hypothesis we obtain \( u_i R_{\text{ideal}} v_i \) for all \( i \), and thus \( u R_{\text{ideal}} f(v_1, \ldots, v_n) \). Consider now \( u' \) and \( v' \) such that \( u = E u' \), \( v R_{\text{ideal}} v \), and \( v R_{\text{ideal}} v' \). Observe that \( u' R_{\text{ideal}} f(u'_1, \ldots, u'_m) \) is only possible if \( f = f' \), \( n = m \), and \( u_i R_{\text{ideal}} u'_i \) for all \( i \). Thus \( v = f(v_1, \ldots, v_n) \) and \( v' = f(v'_1, \ldots, v'_n) \). 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 \( u_{\text{ideal}} \) to be the unique \( u \) such that \( u R_{\text{ideal}} v \).
introduced in Example 1, and the equational theory introduced in Example 2 with in addition the following equation:
\[
\text{dec}(\text{enc}(x, y), y) = x
\]

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

1. \(T \rightarrow R: \{n_T\}_k\)
2. \(1'. T \rightarrow R: \{n'_T\}_k\)
3. \(I(R) \rightarrow T: \{n'_T\}_k\)
4. \(2'. I(R) \rightarrow T: \{n_T\}_k\)
5. \(3'. T \rightarrow R: \{n_T \oplus n'_T\}_k\)

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).

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.