Epistemic Model Checking for Privacy

Fortunat Rajaona
Surrey Centre for Cyber Security
Univeristy of Surrey
United Kingdom
s.rajaona@surrey.ac.uk

Ioana Boureanu
Surrey Centre for Cyber Security
University of Surrey
United Kingdom
i.boureanu@surrey.ac.uk

R. Ramanujam
IMS Chennai
India
jam@imsc.res.in

Steve Wesemeyer
Surrey Centre for Cyber Security
University of Surrey
United Kingdom
s.wesemeyer@surrey.ac.uk

***Abstract*—We define an epistemic logic or logic of knowledge, PL, and a formalism to undertake privacy-centric reasoning in security protocols, over a Dolev-Yao model. We are able to automatically verify all the privacy requirements that are commonplace in security-protocol verification (i.e., strong secrecy, anonymity, various types of unlinkablity including weak unlinkability), as well as privacy notions that are less studied (i.e., privacy regarding lists' membership). Our methodology does not vary with the property: it is uniform no matter the kind of privacy requirement specified and/or verified. We operate in the setting of a bounded number of protocol-sessions. We also implement Phoebe — a proof-of-concept model checker for this methodology. We use Phoebe to check all the aforementioned properties, and we also show-case it on the "benchmark" anonymity and unlinkability requirements of several well-known protocols.**

I. Introduction

"Privacy" is often used as an umbrella term for numerous and –sometimes– quite different concepts, such as anonymity, non-traceability, unlinkability, pseudonymity [1]. In security-protocol verification, there is a more specific set of privacy concepts that are traditionally explored, as these particular concepts link more strongly to cryptographic inferences: e.g., strong secrecy, anonymity, weak and strong unlinkability (see, e.g., [2], [3], [4], [5], [6], [7]). Meanwhile, in formal-methods lines which are primarily rooted in multi-agent systems and logic-based game theory (e.g., [8], [9], [10]), more varied and more nuanced notions of privacy are also considered, such as undetectability, un-observability, and involvement. These latter notions are also understood from a more generic, "plausibilistic" viewpoint: that is, therein, privacy attacks stem from inferences primarily based on a low level of entropy or uncertainty in a system (e.g., due to all possibilities in a system being exhaustively searchable), rather than from deductions centred on a cryptographic/Dolev-Yao attacker. Perhaps unsurprisingly, all these works [8], [9], [10], and others alike [11], [12], [13], formalise such privacy via epistemic logics, also known as logics of knowledge [14], [15], and an intrinsic formalism for capturing uncertainty. Inspired by the this, we also take such an approach to formalising privacy, only that we enhance its semantics with cryptographic mechanisms. In essence, *we obtain the "epistemic privacy" seminally introduced by Halpern and ONeill [11] in the context of multi-agent systems, and much argued-for by Blaauw [10], but augmented with Dolev-Yao [16] dimensions. In turn, we achieve an encoding of security-protocols' privacy such strong secrecy, anonymity, weak and strong unlinkability, via epistemic logic, in a systematic way.*

We bring together epistemic logic with our carefully crafted Dolev-Yao-based semantics for the treatment of privacy-reasoning in protocols in such a way as to be able to express and verify all privacy properties uniformly: that is, *our technique operates in the same way irrespective of the type/flavour of privacy verified.* This is a small bonus since, in the area of security-protocol verification, sometimes there are different methods to verify, e.g., unlinkabilityvs. anonymity [2] ; the latter is often captured via an equivalence between two traces, whereas the former would naturally require checking a set of traces against another set of traces, which leads to it often being analysed via more mechanisable approximations by [6], [7]. That said, we are not solving this, but rather we are providing an alternative methodology that by-passes this issue entirely. Concretely, epistemic logic has a "possible-worlds" semantics [17], meaning that one can compare a system-state with another possible system-state, considering the whole state-space of the system at once, and without reasoning over (particular) system-executions' traces.

On a different note, our logic-based foundation also allows us to handle the intruder and the honest parties in the same manner. This is useful in various settings, as [11] discusses: e.g., in protocols such as mix-nets [18], where an honest party need not know who an intermediary "relay point" is.

Our formalism work can also be viewed as an extension of in-principle, epistemic approaches [19], [20], [21], [22], where a multi-agent systems semantics was underpinned by a Dolev-Yao intruder model but only for the verification of security properties (e.g., authentication and secrecy). Our model lifts these ideas to reasoning about privacy properties.

All in all, we are the first to provide a mechanised tool for a systematic analysis of security protocols' against Dolev-Yao-based and epistemically-expressed privacy.

We make the following key contributions.

1. We introduce a *logic of knowledge*, called *"privacy logic" ($\mathcal{PL}$)*, concerned with what the intruder or other agents know (in the well-founded, epistemic sense), at a state in a protocol's execution.

2. We give a *privacy-centred semantics* for our $\mathcal{PL}$ logic and for security-protocols' executions. This includes a new *cryptographic indistinguishability* between protocol/system

states, defined on top of a *privacy-driven enrichment of Dolev-Yao [16] deductions*.

3. We encode various flavours of privacy requirements in our $\mathcal{PL}$ logic, including the well-understood notions of Dolev-Yao-induced anonymity and unlinkability. We also include a comprehensive comparison of our privacy expressions with those of other privacy-analysis works.

4. We present `Phoebe` – a *proof-of-concept* model checker for our logic against our semantics. We exemplify its use by *automatically analysing various privacy requirements, including unlinkability*, in several protocols which are the "benchmark" for such analyses, i.e., the Private Authentication Protocol [23], and the RFID protocols in [3], [6], as well as the LoRaWAN Join v.1.1 protocol [24]. We also include a comparison between `Phoebe` and other related tools such as DEEPSEC [25], Sat-Eq [26], both in terms of abilities to handle privacy analyses as well as performance.

II. A Running Example

We start by introducing our running-example, the second Private Authentication Protocol by Abadi [23], which we denote by *PrivAuth*:

Example 1 (*PrivAuth*: Private Authentication Protocol [23])

$$1.\ A \to B : \text{“hello”}, \{\text{“hello”}\ , \mathsf{pubk}(A), N_A\}_{\mathsf{pubk}(B)} \quad \textit{(if } \alpha\textit{)}$$
$$2.\ C \to A : \text{“ack”}, \{\text{“ack”}, \mathsf{pubk}(C), N_A, N_C\}_{\mathsf{pubk}(A)} \quad \textit{(if } \beta\textit{)}$$
$$2'.C \to A : \text{“ack”}, \{N\}_K \quad \textit{(if } \neg\beta\textit{)}$$

where $\{\cdot\}_{\mathsf{pubk}(Y)}$ denotes encryption with the public key of Y, condition α requires $\mathsf{pubk}(B)$ be in a whitelist S_A held by A, condition β that entity C in step 2 be the same as B in step 1 and $\mathsf{pubk}(A)$ be in a whitelist S_C held by C.

In *PrivAuth*, each participant X has a *whitelist* S_X, which is a set of public-keys of parties that X intends to communicate with. In step 1, a participant A acts as *the initiator*: A broadcasts a "hello" message intended for a recipient B who satisfies condition α, that is this B is in A's whitelist S_A. The "hello" message is received by various participants C. A given C acts as *the responder*: C tries to decrypt the second part of the "hello" message, which will decrypt just for A's intended recipient: i.e., when C is B. In this case, the protocol proceeds to step 2: C checks if the public-key $\mathsf{pubk}(A)$ found inside the received message is in C's whitelist S_C. If so, then β is fulfilled and C broadcasts a *real acknowledgement* message: "ack" , $\{\text{“ack”}, \mathsf{pubk}(C), N_A, N_C\}_{\mathsf{pubk}(A)}$. If $\mathsf{pubk}(A)$ is not in S_C, then β is not fulfilled and C broadcasts instead a *decoy acknowledgement* message: a nonce N encrypted with a random key K. If C cannot decrypt the "hello" message (i.e., $C \neq B$ and so β is also not fulfilled), then the protocol proceeds to step $2'$. In this case, C sends the *decoy acknowledgement* message as per the above. All messages are broadcast over a network of protocol participants. Decoy acknowledgement messages are constructed such that they are indistinguishable from real acknowledgements to any observers or protocol participants.

Abadi [23] considers the following privacy goals:

***PrivAuth*'s second goal** is *anonymity:* "A should not have to indicate its identity or presence to any principal outside S_A".

***PrivAuth*'s third goal** is *third-parties' privacy w.r.t. the whitelists*: a responder C should not leak *anything* about *third*-parties found in S_C: even when a participant A can deduce that itself is in S_C (given C's reply in step 2), this A should not know whether a third-party E is in S_C or not.

Note 1: *PrivAuth*'s third goal carries a cryptographic nature due to the protocol it refers to, yet it is also plausibilistic, in its formulation; this is due to the usage of the words "should not leak *anything*", and an example of this could be the size of the whitelists – which is not a concern tacked in cryptographic privacy. Also, *PrivAuth*'s third goal does not refer to an intruder, but to honest participants not inferring certain data. Thus, we chose this running example as it eases the illustration of our introductory aims to tackle both cryptographic and plausibilistic privacy, as well as to look at intruder-centric as well as honest violation of privacy. In this vein, as far as we know, *PrivAuth*'s third goal was also not studied before either.

III. Privacy-centred Protocol Model

Our protocol model. assumes a Dolev-Yao (DY) adversary [16], yet the agents and the DY adversary are systematically enhanced with privacy-centric deductions. To allow for this, our cryptographic deductions operate not only over sets of messages [27] (as is the case in classical Dolev-Yao-based models [28], [29], [30]), but also over hash-tables of algebraic abstract terms and concrete messages.

A. *Protocol Algebra*

The protocol description in our formalism follows the same style as in epistemic-based model [28], strand space model [30], and multiset rewriting model [29].

We refer to the free variables that appear in a protocol description as *protocol variables*, and the cryptographic operations appearing are modelled as *functional symbols*.

***Protocol Signature**:* We use a finite set $\mathcal{V}$ of *protocol variables* partitioned into:

- a set $\mathcal{A}$ of *participant variables* or variables of *sort participant*, e.g., $A, B, C, \ldots$,
- a set $\mathcal{K}$ of *key variables* or variables of *sort key*, partitioned into two subsets: $\mathcal{K}_s$ – denoting short-term keys, and $\mathcal{K}_l$ – denoting long-term keys;
- a set $\mathcal{N}$ of *nonces variables* or variables of *sort nonce*, e.g., $M, N, \ldots$

For some protocols, we distinguish a set $\mathcal{S}$ of variables of sort participant called *whitelist*. In our running-example *PrivAuth*, a whitelist S_A encapsulates the list of participant variables showing to whom A is willing to speak.

As per the usual, functional symbols model cryptographic operations. We consider a set $\mathcal{F}_0$ of *constants/0-ary functional symbols*. Then, we define our set $\mathcal{F}$ of *functional symbols*,

partitioned into *constructor symbols* $\mathcal{F}_c$ and *destructor symbols* $\mathcal{F}_d$, i.e., $\mathcal{F} = \mathcal{F}_c \uplus \mathcal{F}_d$:

$$\mathcal{F}_c = \mathcal{F}_0 \cup \{\{|\cdot|\}_\cdot,\ \{\cdot\}_\cdot,\ \mathsf{seck}(\cdot),\ \mathsf{pubk}(\cdot), \mathsf{hash}(\cdot),\ \langle\cdot,\cdot\rangle\}$$
$$\mathcal{F}_d = \{\mathsf{sdec}(\cdot,\cdot), \mathsf{adec}(\cdot,\cdot),\ \mathsf{check}(\cdot,\cdot), \mathsf{proj}_1(\cdot), \mathsf{proj}_2(\cdot)\},$$

where hash denotes a hash function, $\{|\cdot|\}$ and sdec – symmetric encryption/decryption, $\{\cdot\}$ and adec – asymmetric encryption/decryption, $\langle\cdot,\cdot\rangle$ — pairing, $\mathsf{proj}_1/\mathsf{proj}_2$ – projections on the first and second element of a pair, pubk and seck capture public/private keys, and pairing and projections naturally extend to tuples. $\mathcal{F}_c^*$ denotes the set $\mathcal{F}_c \setminus \{\mathsf{seck}\}$ of functional symbols that are public to all, i.e., any party can apply on the messages it has.

Altogether, our *protocol signature* is the tuple $(\mathcal{V}, \mathcal{F})$ of the set $\mathcal{V}$ of (sorted) *protocol variables*, and the set $\mathcal{F}$ of *functional symbols* as per the above.

***Abstract Terms**: Abstract terms* are the set $\mathcal{T}(\mathcal{F}, \mathcal{V})$ of objects obtained by applying inductively functional symbols in $\mathcal{F}$ to protocol variables in $\mathcal{V}$: e.g., $\mathsf{pubk}(A)$ is an abstract term representing A's public key in a protocol description.

Abstract terms have *types* defined by application of functional symbols to sorts: e.g., the *type pair*, the type *encryption of a pair*, etc. We do not formalise this, as it follows naturally from the constructive definition of terms.

Abstract terms capture messages as per the protocol description. In contrast, *concrete terms* represent concrete-value messages in actual executions. So, concrete terms are formed just like abstract terms, but with actual values instead of variables: e.g., $\mathsf{pubk}(alice)$ vs. $\mathsf{pubk}(A)$.

***Concrete Terms**:* In line with the sorts of variables, we consider $\mathtt{D} = \mathtt{D}_\mathcal{N} \cup \mathtt{D}_\mathcal{A} \cup \mathtt{D}_\mathcal{K}$ to be a *domain for atomic values*, such that:

- $\mathtt{D}_\mathcal{A}$ contains values for variables of sort participant also called *participant' names* or , e.g., $alice, bob, \ldots$;
- $\mathtt{D}_\mathcal{K}$ contains values for variables of sort keys also called *concrete keys* or, denoted $k, k_1, k_2, \ldots$;
- $\mathtt{D}_\mathcal{N}$ contains values for variables of sort nonce also called *concrete nonces*, denoted by $n, n_1, n_2, \ldots$.

Concrete terms are the set $\mathcal{T}(\mathcal{F}, \mathtt{D})$ of objects obtained by applying inductively symbols in $\mathcal{F}$ to atoms $\mathtt{D}$.

The types of abstract terms extend to concrete terms, just as sorts extend from variables to values.

For an abstract/concrete term, we write $Subterms(t)$ for the set of terms constructed out of variables and constructor symbols inductively, up to t and including t.

1) ***Equational Theory**:* Our *equational theory* $=_\mathcal{E}$ is the smallest congruence on $\mathcal{T}(\mathcal{F}, \mathcal{V})$ or on $\mathcal{T}(\mathcal{F}, \mathcal{D})$ containing the following set $\mathcal{E}$ of equations:

$$\mathsf{proj}_1(\langle\tau, \tau'\rangle) =_\mathcal{E} \tau \qquad \mathsf{proj}_2(\langle\tau, \tau'\rangle) =_\mathcal{E} \tau'$$
$$\mathsf{adec}(\{\tau\}_{\mathsf{pubk}(p)}, \mathsf{seck}(p)) =_\mathcal{E} \tau \qquad \mathsf{sdec}(\{|\tau|\}_w, w) =_\mathcal{E} \tau$$
$$\mathsf{check}(\{\tau\}_{\mathsf{seck}(p)}, \mathsf{pubk}(p)) =_\mathcal{E} \tau.$$

where p is either a variable or a value of sort participant, and w is either a variable or a value of sort key.

B. Protocol Actions and Roles

1) ***"Simple" Actions**:* Actions constitute the base of a protocol description. Our *simple protocol actions* are partitioned into *send actions* and *receive actions*, and are syntactically defined as follows.

Definition 1 (Syntax for Simple Sent and Receive Actions.) The constructs below

$$i.A\ !\ : (test(\Phi)) \qquad (\mathcal{V}_\tau)\ \tau \qquad \text{(send)}$$
$$i.A\ ?\ : (test_{pre}(\Psi), test_{post}(\Psi')) \qquad \tau, \qquad \text{(receive)}$$

represent a *simple sent* and *simple receive* action respectively, where:
– $i \in \mathbb{N}$ denotes the protocol step at which the action is performed in the protocol description;
– $\tau \in \mathcal{T}(\mathcal{F}, \mathcal{V})$ denotes the sent/received message;
– $A \in \mathcal{A}$ is a participant variable, denoting the entity in the protocol description sending/receiving the message;
– $\mathcal{V}_\tau \subset Subterms(\tau) \cap \{\mathcal{K}_s \cup \mathcal{N}\}$ is the set of nonces and short-term keys within τ newly generated at this step;
– Φ, Ψ, Ψ' are logical formulae expressed over *logical variables* overloading the protocol variables;
– $test(\Phi)$, $test_{pre}(\Psi)$ are checks for the satisfaction of the Φ and Ψ formulae respectively, encoding the enabling conditions of our actions; $test_{post}(\Psi')$ is a check for the satisfaction of the Ψ' formula, encoding the update resulting from the action;
– $test(\Phi)$, $test_{pre}(\Psi)$, $test_{post}(\Psi')$ are optional: i.e, there are actions where Φ, or Ψ and/or Ψ' are the constant truth.

"Pre"/"Post" Conditions on Simple Actions. $test_{post}(\Psi')$ denotes the following type of condition: a receipt of a message takes places at state of our transition-system semantics, and that state is updated a first time, and then the check encapsulated in $test_{post}(\Psi')$ is performed on this updated state, to trigger a final update (or not). We use such a $test_{post}(\cdot)$ in Example 8. Differently, $test_{pre}(\cdot)$ encodes a test on an incoming message against the state before its receipt, updating the state if the test succeeds. Finally, we use $test(\cdot)$ on sending actions, e.g., to encode checks as to whether an interlocutor belong to a whitelist (see Example 2).

2) ***Branching Actions**:* The tests $test(\Phi)$, $test_{pre}(\Psi)$, $test_{post}(\Psi')$ inside simple actions do not encode "if-then-else", but rather pre/post conditions on actions' executions. However, there are privacy-relevant protocols where actions follow an 'if-then-else' construct semantics. To encapsulate this, we next introduce the notion of *branching actions*; these have the same components as a simple action, except that they present two alternatives at a single step.

Definition 2 (Syntax for Branching Sent and Receive Actions.) A *branching send action* is syntactically defined by:

$$i.A\ !\ : test(\Phi)\ \ (\mathcal{V}_{\tau_0})\ \tau_0$$
$$|A\ !\ : test(\neg\Phi)(\mathcal{V}_{\tau_1})\ \tau_1$$

A *branching receive action* is syntactically defined by:

$$i.A\ ?\ : (\text{optional simple-receive tests})\ \tau_0$$
$$|A\ ?\ : (\text{optional simple-receive tests})\ \tau_1,$$

where the denotations of i, A, $!$, $?$, $(\mathcal{V}_{\tau_i})$, τ_i, formulae of the type Φ is as per Definition 1, and $test(\cdot)$ in not optional.

Definition 2 denotes that if $test(\Phi)$ holds then τ_0 is sent, otherwise τ_1 is sent. Depending on the branch taken in a branching send action, either τ_0 or τ_1 will be received in the corresponding branching receive action (see Example 2).

3) ***Protocol Roles & Protocols****:* The abstract representation of a participant A's messages and actions is called the *role of* A, denoted by η_A. We refer to protocol variables which are not fresh nonces or short-term keys but originate from η_A as *role parameters* and denote this $param(\eta_A)$. As usual, *a protocol* is formalised via a set of roles. We exemplify this on our running example.

Example 2 (Protocol Specification for *PrivAuth*.) *PrivAuth* has two roles: the role η_A of A (initiator) and the role η_C of C (responder). The role η_A is given by

$$\begin{array}{ll} 1.A\ !\ : & test(B \in S_A)(\{N_A\})\{\langle \mathsf{pubk}(A), N_A\rangle\}_{\mathsf{pubk}(B)} \\ 2.A\ ?\ : & \{\langle N_A, N_C, \mathsf{pubk}(B)\rangle\}_{\mathsf{pubk}(A)} \\ |A\ ?\ : & \{N\}_K \end{array}$$

The role η_C of C is given by:

$$\begin{array}{lll} 1.C\ ?\ : & & \{\langle \mathsf{pubk}(A), N_A\rangle\}_{\mathsf{pubk}(B)} \\ 2.C\ !\ : test(\beta) & (\{N_C\}) & \{\langle N_A, N_C, \mathsf{pubk}(B)\rangle\}_{\mathsf{pubk}(A)} \\ |C\ !\ : test(\neg\beta)(\{N, K\})\{N\}_K, & & \end{array}$$

where:
– the set of protocol variables used is $\mathcal{V} = \mathcal{A} \cup \mathcal{N} \cup \mathcal{K}$ with $\mathcal{A} = \{A, B, C\}$, $\mathcal{N} = \{N, N_A, N_C\}$, $\mathcal{K} = \{K\}$;
– the set of whitelist variables is $\mathcal{S} = \{S_A, S_C\}$;
– A, B, S_A are in $param(\eta_A)$, and C, S_C are in $param(\eta_C)$;
– the functional symbols used are given by $\mathcal{F} = (\mathcal{F}_c, \mathcal{F}_d)$;
– β is the formula $has_C(\mathsf{seck}(B)) \wedge A \in S_C$, which formalises that C is the intended recipient and willing to speak to A; the meaning of β will be clarified in Section IV. Constants "hello" and "ack" were omitted in this presentation.

C. Substitutions

Substitutions map protocol variables to concrete terms, to simulate a (potentially adversarial) protocol execution with concurrent instances of role parameters.

Definition 3 (Substitutions.) A *substitution* $\sigma : \mathcal{V} \to \mathcal{T}(\mathcal{F}, \mathtt{D})$ is a total mapping of the protocol variables to concrete terms.

Substitutions are homomorphically lifted from variables to terms, as expected. We consider *substitutions* which are

- *well-sorted* – map abstract terms to concrete terms of the same sort (nonce variables in $\mathcal{N}$ to nonces in $\mathtt{D}_{\mathcal{N}}$, key variables in $\mathcal{K}$ to concrete keys in $\mathtt{D}_{\mathcal{K}}$ etc.)
- *suitable for participant variables* – if A and B are different participant variables attached to a role, then $\sigma(A) \neq \sigma(B)$.

In the initialisation phase of our model, for each protocol role A, we apply substitutions to the $A-$role parameters $param(\eta_A)$, i.e., parts of the role remain uninstantiated. Example 3 below instantiates the role parameters in *PrivAuth*.

Example 3 (Substitutions for *PrivAuth*.) Consider the protocol *PrivAuth*. Assume three values $alice, bob$, and $charlie$ in $\mathtt{D}_{\mathcal{A}}$, and *three substitutions* $\sigma_1, \sigma_2, \sigma_3$ defined by:

σ_1	σ_2	σ_3
$A \mapsto alice$	$A \mapsto alice$	$A \mapsto alice$
$B \mapsto bob$	$B \mapsto bob$	$B \mapsto bob$
$C \mapsto bob$	$C \mapsto bob$	$C \mapsto charlie$
$S_A \mapsto \{\mathsf{pubk}(bob)\}$	$S_A \mapsto \{\mathsf{pubk}(bob)\}$	$S_A \mapsto \{\mathsf{pubk}(bob)\}$
$S_C \mapsto \{\mathsf{pubk}(alice)\}$	$S_C \mapsto \{\mathsf{pubk}(charlie)\}$	$S_C \mapsto \{\mathsf{pubk}(bob)\}$

In the above, $alice$ is the initiator and bob the intended receiver. The substitution σ_1 simulates an execution with bob as responder, and willing to reply to $alice$, σ_2 – a session with bob as responder, but not willing to reply, and σ_3 – a session with $charlie$ as responder.

In our practical protocol-verification (Section VI), we consider a bounded number of substitutions. Variables that are not mapped via these substitutions in the initial setup of the system are generated by considering all possible values[1] for data, including allowing fresh values for the intruder.

We use the notion of *protocol session* as per the usual: it encapsulates any execution of the protocol resulting from the application of several substitutions to the protocol roles.

D. Our Protocol Model: Privacy Systems (PS)

We now define a *Privacy System (PS)*: our multi-agent transition-system for a security-protocol semantics with a Dolev-Yao model enhanced for privacy-centric reasoning.

First, assume an arbitrarily fixed tuple (Pr, Σ, n_{sess}): Pr is a protocol defined over our signature $(\mathcal{V}, \mathcal{F})$; Σ is a finite set of substitutions; n_{sess} is a natural number denoting the bound on the number of sessions. From (Pr, Σ, n_{sess}), the *Privacy System (PS)* $I^{Pr,\Sigma,n_{sess}}$ is constructed, step by step, from the following objects: agents, system states, state updates, and agents' indistinguishability relations on states.

1) ***Agents of a PS****:* Intuitively, an *honest agent* represents an instantiation of a protocol role, attached to a session and to a class of substitutions that agree on the role parameters. The *intruder* $\mathtt{I}$ is a special agent, not attached to any particular role or session.

Definition 4 (Agents of a PS.) Let (Pr, Σ, n_{sess}) be a tuple a protocol Pr, a set Σ of substitutions, and a number n_{sess} of sessions. Then, $Ag = I \cup \{ag_1, \ldots, ag_\omega\}$ are the *agents of the PS* $I^{Pr,\Sigma,n_{sess}}$, where:

- each *honest agent* ag_j is a tuple $(\sigma(param(\eta_A)), i)$, where η_A is a role in Pr, σ is a substitution in Σ, $i \leq n_{sess}$ is a session number, and $j \in \{1, \ldots, \omega\}$;
- $\mathtt{I}$ is the *intruder agent*.

The number ω of honest agents is a function of $|\Sigma|$, n_{sess}, and the sizes of the role parameters $param(\eta_A)$.

Each honest agent $ag_j = (\sigma(param(\eta_A)), i)$ corresponds to a protocol session run by a concrete participant, playing the A-role. The *name of agent* $ag_j = (\sigma(param(\eta_A)), i)$ is the

[1]This is finite, as we use a bounded number of well-sorted substitutions.

participant name that substituted A in session i, denoted by $name(ag_j) = \sigma(A)$.

A participant name *alice* playing several roles will correspond to different agents $ag_j, ag_k, \ldots$ with the same agent's name *alice*, as illustrated by the following example.

Example 4 (Agents for *PrivAuth*.) Consider the roles η_A and η_C of *PrivAuth* and let $n_{sess} = 2$. From the three substitutions $\sigma_1, \sigma_2, \sigma_3$ in Example 3, we obtain eight agents:

$$ag_1 = \sigma_1(\eta_A^{par}, 1) = \sigma_2(\eta_A^{par}, 1) = \sigma_3(\eta_A^{par}, 1)$$
$$ag_2 = \sigma_1(\eta_A^{par}, 2) = \sigma_2(\eta_A^{par}, 2) = \sigma_3(\eta_A^{par}, 2)$$
$$ag_3 = \sigma_1(\eta_C^{par}, 1) \quad ag_4 = \sigma_2(\eta_C^{par}, 1) \quad ag_5 = \sigma_3(\eta_C^{par}, 1)$$
$$ag_6 = \sigma_1(\eta_C^{par}, 2) \quad ag_7 = \sigma_2(\eta_C^{par}, 2) \quad ag_8 = \sigma_3(\eta_C^{par}, 2)$$

where η_A^{par} is a short-hand for $param(\eta_a)$. The substitutions $\sigma_1, \sigma_2, \sigma_3$ coincide on η_A^{par}, hence we have only two A-agents ag_1, and ag_2, for each of the two sessions. Agents are named as follows: ag_1 and ag_2 as $alice = \sigma_1(A) = \sigma_2(A) = \sigma_3(A)$; ag_3, ag_4, ag_6 and ag_7 as $bob = \sigma_1(C) = \sigma_2(C)$; ag_5 and ag_8 as $charlie = \sigma_3(C)$.

*2) **States of a PS**:* Each agent ag of a PS $I^{Pr,\Sigma,n_{sess}}$ is endowed with a *local state* s_{ag}, which keeps track of messages ag receives and sends, but also of the reasoning w.r.t. privacy applied to these messages.

Definition 5 (Local/Global States in a PS.) Let ag be an agent of $I^{Pr,\Sigma,n_{sess}}$. A *local state* s_{ag} *of* ag consists of:

- a set of concrete terms, $terms(s_{ag})$, called *knowledge-set*;
- a multimap $\mathbf{F} = \{\tau_1 \mapsto t_1, \ldots, \tau_s \mapsto t_s\}$, linking abstract terms to concrete terms, called *frame* and denoted $frame(s_{ag})$;
- an integer variable recording the stage in the agent's execution called *protocol step* and denoted $step(s_{ag})$.

A *global state in the PS* is a tuple $s = (s_{ag})_{ag \in Ag}$.

***Knowledge-sets & Frames**: The knowledge-set* $terms(s_{ag})$ contains the concrete terms/values that agent ag initially knows via the role instantiation, as well as those that ag generates, sends and receives.

The frame $frame(s_{ag})$ tracks concrete messages that ag gathers by their abstract-term signature. Consider that ag receives a concrete message $f(t)$, which ag may not even be able decrypt. From the protocol description, ag knows the message is an instance of the abstract term $f(\tau)$. In that case, ag adds $f(\tau) \mapsto f(t)$ to its frame; in Section III-D4, we formalise how such mappings are systematically added into an agent's frame following a protocol action.

An honest agent's frame is a map, i.e., each abstract term uniquely maps to a message, since an honest agent is linked to a single session. However, the intruder's frame is a multimap, i.e., one abstract term can be associated to several concrete terms, to account for the intruder being present in several sessions.

***Initial States**:* Every agent is endowed with an initial state. Within, the step variable is initialised at zero. Then, an initial knowledge-set stores the agent's initial knowledge (e.g., names, long-term keys). And, an initial frame captures the agent's role setup (e.g., intended recipient).

Definition 6 (Initial Local/Global States of a Privacy System.) The *initial local state* s_{ag}^0 of an honest agent $ag = (\sigma(param(\eta_A)), i)$ consists of:

- the knowledge-set $Public \cup Secret_{ag} \cup \sigma(param(\eta_A))$,
- the frame $\{\tau \mapsto \sigma(\tau) \,|\, \tau \in param(\eta_A)\} \cup \{self \mapsto name(ag)\}$,
- the step $step(s_{ag}^0) = 0$,

such that: $Secret_{ag}$ contains ag's secret key, and all (long-term) keys that ag shares with other agents; $Public$ contains constants in $\mathcal{F}_0$, names in $\mathtt{D}_\mathcal{A}$, and the public keys in $D_\mathcal{K} \cup \mathsf{pubk}(\mathtt{D}_\mathcal{A})$. The *initial state* $s_\mathtt{I}^0$ *of the intruder* consists of:

- the knowledge-set $Public \cup Secret_\mathtt{I}$
- an empty frame
- the step $step(s_\mathtt{I}^0) = 0$, where

$Secret_\mathtt{I}$ contains $\mathtt{I}$'s secret key, and – when modelling agents being corrupted by $\mathtt{I}$ – $Secret_\mathtt{I}$ also includes the secret key of such agents.

An *initial global state* is a tuple $s^0 = (s_{ag}^0)_{ag \in Ag}$.

*3) **Deductions on PS Local States**:* On the knowledge-set of a local state, we apply Paulson's well-known "synth" and "analz" operators [27]; these model agents' ability to construct or destruct terms from/into their components. We recast Paulson's definitions over our protocol signature $(\mathcal{V}, \mathcal{F})$ and equation theory $=_\mathcal{E}$.

Definition 7 (Analz and Synth [27].) Let $\mathbf{T}$ be a set of concrete terms. *Closure of* $\mathbf{T}$ *under synthesis/constructors*, $synth(\mathbf{T})$, is the least set that contains $\mathbf{T}$, and if $synth(\mathbf{T})$ contains the terms $t_1, \ldots, t_n$ (for some $n \geq 1$) and $f \in \mathcal{F}_c^*$ of arity n, then $synth(\mathbf{T})$ contains $f(t_1, \ldots, t_n)$.

Closure of $\mathbf{T}$ *under analysis/destructors*, $analz(\mathbf{T})$, is the least set that contains $\mathbf{T}$, and if $analz(\mathbf{T})$ contains constructor terms $t_1, \ldots, t_n$ (for some $n \geq 1$) and there is a destructor symbol $g \in \mathcal{F}_d$ of arity n such that $g(t_1, \ldots, t_n) =_\mathcal{E} t$, then $analz(\mathbf{T})$ contains t.

The *closure under synthesis and analysis* of a set of concrete terms $\mathbf{T}$, denoted $\overline{\mathbf{T}}$, is the set given by $synth(analz(\mathbf{T}))$.

We introduce $analz^P$, which lifts the standard $analz$ operator to our notion of frame. Intuitively, $analz^P$ allows an agent which has the mapping $\tau \mapsto t$ already in its frame, to augment its frame with the mapping $\tau' \mapsto t'$, for subterms t' of t and appropriate τ's.

Definition 8 (Privacy-driven Analysis: $analz^P$) Given a tuple $(\mathbf{F}, \mathbf{T})$ where $\mathbf{F}$ is a frame and $\mathbf{T}$ is a knowledge-set, $analz^P(\mathbf{T}, \mathbf{F})$ is the smallest frame such that:

1) $\mathbf{F} \in analz^P(\mathbf{T}, \mathbf{F})$
2) if $(f(\tau_1, \ldots, \tau_n) \mapsto f(t_1, \ldots, t_n)) \in analz^P(\mathbf{T}, \mathbf{F})$, with $f \in \mathcal{F}_c^*$, and $t_1, \ldots, t_n \in analz(\mathbf{T})$, then $(\tau_i \mapsto t_i) \in analz^P(\mathbf{T}, \mathbf{F})$ for each $i \in \{1, \ldots, n\}$.
3) if $f(\tau_1, \ldots, \tau_n) \mapsto f(t_1, \ldots, t_n) \in analz^P(\mathbf{T}, \mathbf{F})$, with $f \in \mathcal{F}_c$,

a) if there is a unary destructor function $g \in \mathcal{F}_d$ and there is a j in $\{1,\ldots,n\}$, such that $g(f(t_1,\ldots,t_n)) =_E t_j$, then $(\tau_j \mapsto t_j) \in analz^P(\mathbf{T},\mathbf{F})$.
b) if there is a binary destructor function $g \in \mathcal{F}_d$ and there is a j in $\{1,\ldots,n\}$, such that $g(f(t_1,\ldots,t_n),t') =_E t_j$ for some $t' \in analz(\mathbf{T})$, then $\tau_j \mapsto t_j \in analz^P(\mathbf{T},\mathbf{F})$.

Definition 8-2) captures a deduction by "exhaustively" applying public constructor symbols in $\mathcal{F}_c^*$ to known messages, similarly to a dictionary attack, as per the next example.

Example 5 ($analz^P$ via public constructors.) Assume that, *alice* sends $m = \{alice\}_{\mathsf{pubk}(bob)}$ to *bob*. The intruder intercepts m, and adds the mapping $\{A\}_{\mathsf{pubk}(B)} \mapsto m$ into its frame. The intruder cannot decrypt m. However, having the name *alice* and the key $\mathsf{pubk}(bob)$, it can reconstruct m. Thus, via $analz^P$ (Definition 8.(2)), the intruder deduces the mappings $A \mapsto alice$ and $\mathsf{pubk}(B) \mapsto \mathsf{pubk}(bob)$.

Definition 8-(3) captures a deduction based on applying destructor symbols in $\mathcal{F}_d$, when all the necessary "destructing" ingredients are $\mathbf{T}$, as illustrated by Example 6 below.

Example 6 ($analz^P$ via destructors.) Consider an instance of *PrivAuth*, where an agent ag named *alice* sends the message $m = \{alice, n\}_{\mathsf{pubk}(bob)}$. If an agent ag' named *bob* receives m, it stores the mapping $\{A, N_A\}_{\mathsf{pubk}(B)} \mapsto m$. As ag' can also decrypt m, m and n are added into its knowledge-set. Then, by applying $analz^P$ (Definition 8.(3)), ag' deduces the mappings $A \mapsto alice$, $N_A \mapsto n$.

*4) **State-Updates in a PS**:* A global state encapsulates the result of several substitutions of Σ. So, a global state s can be updated via the same action act in different ways, depending on the substitution σ considered when applying act to s. To this end, we introduce the notion of event.

Definition 9 (Event.) An *event* e in a PS $I^{\Sigma,Pr,n_{sess}}$ is a tuple (act, σ, n), where act is an action in Pr, σ is a substitution in Σ, and $1 \le n \le n_{sess}$. If act is a send action, then e is a *send event*. If act is a receive action, then e is a *receive event*.

An event e acts on a global state s producing a *state-update from state s to s'*, written $s' = update(s,(act,\sigma,n))$. The function $update$ is projected on every local state s_{ag} of s, and the event (act,σ,n) is applied to these local states s_{ag}. We formalise $update(s,e)$ below.

Definition 10 (The $update$ Function for a Send Event.) Let s be a state and $e = (act,\sigma,n)$ be a send event with $act = i.A\ !\ :\ test(\Phi)\ (\mathcal{V}_\tau)\ \tau$. Let $ag = \sigma(param(\eta_A), n)$ be the sending agent, $\sigma(\mathcal{V}_\tau)$ denote the concrete instantiations of the fresh terms $\mathcal{V}_\tau$, and $t = \sigma(\tau)$ – the concrete, sent value for the abstract term τ. The *enabling conditions for* $update(s,e)$ are:

$$t \in \overline{terms(s_{ag}) \cup \sigma(\mathcal{V}_\tau)} \quad (1)$$
$$step(s_{ag}) = i - 1 \quad (2)$$
$$s \models_\sigma \Phi \quad (3)$$

and the *updated state* $s' = update(s,e)$ is as follows:

- for the sending agent ag:

$$step(s'_{ag}) := step(s_{ag}) + 1 \quad (4)$$
$$terms(s'_{ag}) := analz(terms(s_{ag}) \cup \sigma(\mathcal{V}_\tau) \cup \{t\}) \quad (5)$$
$$frame(s'_{ag}) := analz^P(terms(s'_{ag}), F), \quad (6)$$

where $F = frame(s_{ag}) \cup \{\tau_f \mapsto \sigma(\tau_f) \mid \tau_f \in \mathcal{V}_\tau\} \cup \{\tau \mapsto t\}$

- for the intruder $\mathtt{I}$:

$$terms(s'_{\mathtt{I}}) := analz(terms(s_{\mathtt{I}}) \cup \{t\}) \quad (7)$$
$$frame(s'_{\mathtt{I}}) := analz^P(terms(s'_{\mathtt{I}}), frame(s_{\mathtt{I}}) \cup \{\tau \mapsto t\}) \quad (8)$$

- for all the other agents $ag' \notin \{\mathtt{I}, ag\}$:

$$s'_{ag'} := s_{ag'} \quad (9)$$

Condition (1) for sending t is standard: t can be sent only if it can be composed out of the terms in ag's knowledge-set $terms(s_{ag})$. Condition (2) on $step$ is self-explanatory. Condition (3) requires the formula Φ to hold[2] at the state s. In the state-updates (4), (5), (6) the agent increases his protocol steps, and records the data generated and sent in his state. The updates (7) and (8) mean that the intruder intercepts the message being sent, as per the usual Dolev-Yao intruder model.

After adding a new term to the knowledge-set and mapping it into the frame, all state-updates include the $analz$ operator being applied to the new knowledge-set and the $analz^P$ operator being applied to the new frame, both in the case of the sending agent ag and of the intruder $\mathtt{I}$.

Definition 11 (The $update$ Function for a Receive Event.) Let s be a state and $e = (act,\sigma,n)$ be a receive event with $act = i.\ A\ ?\ :\ (test_{pre}(\Psi), test_{post}(\Psi'))\ \tau$. Let the receiving agent be $ag = \sigma(param(\eta_A), n)$, and $t = \sigma(\tau)$ denote the concrete, received value for the abstract term τ. Let $\mathcal{V}_\tau$ be a possibly empty set of subterms of τ chosen by the intruder, i.e., $\mathcal{V}_\tau = \{T \in Subterms(\tau) | \sigma(T) \in terms(s_I)\}$.

Then, the *enabling conditions for* $s' = update(s,e)$ are

$$t \in \overline{terms(s_{\mathtt{I}}) \cup \sigma\mathcal{V}_\tau} \quad (10)$$
$$step(s_{ag}) = i - 1 \quad (11)$$
$$s \models_\sigma \Psi \text{ and } s' \models_\sigma \Psi' \quad (12)$$
$$\begin{cases} \text{for all } (\tau' \mapsto t') \text{ in } analz^P(terms(s_{ag}) \cup \{t\}, \{\tau \mapsto t\}) \\ \text{if } (\tau' \mapsto t'') \text{ in } frame(s_{ag}), \text{ then } t' =_E t'' \end{cases} \quad (13)$$

and the *updated state* $s' = update(s,e)$ is as follows,

- for the receiving agent ag:

$$step(s'_{ag}) := step(s_{ag}) + 1 \quad (14)$$
$$terms(s'_{ag}) := analz(terms(s_{ag}) \cup \{t\}) \quad (15)$$
$$frame(s'_{ag}) := analz^P(terms(s'_{ag}), frame(s_{ag}) \cup \{\tau \mapsto t\}) \quad (16)$$

- for the intruder $\mathtt{I}$:

$$terms(s'_{\mathtt{I}}) := analz(terms(s_{\mathtt{I}}) \cup \sigma(\mathcal{V}_\tau) \cup \{t\}) \quad (17)$$
$$frame(s'_{\mathtt{I}}) := analz^P(terms(s'_{\mathtt{I}}), F), \quad (18)$$

where $F = frame(s_{\mathtt{I}}) \cup \{\tau_c \mapsto \sigma(\tau_c) \mid \tau_c \in \mathcal{V}_\tau\} \cup \{\tau \mapsto t\}$

[2]This satisfaction is formally defined later, via the semantics of our logic (Definition 19), and Φ is evaluated at a state together with a run.

- for all the other agents $ag' \notin \{\mathtt{I}, ag\}$:

$$s'_{ag'} := s_{ag'} \tag{19}$$

Condition (10) stipulates that all that is received on the network is injected/forwarded by the intruder. Condition (11) on $step$ is self-explained. Then, condition (12) requires that the state s before the receive satisfies ψ and the future state s' would satisfy ψ'. The enabling condition (13) denotes a "matching-receive" à la [31], and says that an agent accepts only messages that match their already-formed view of the execution. This is enforced by requiring any mapping $(\tau' \mapsto t')$ that could result from adding the incoming term t, deducible by the agent via $analz^P$, should be consistent with the previous entries in the frame.

The updates (14) on $step$, (15) on adding t to the knowledge-set, and (16) on adding $\tau \mapsto t$ to the receiver's frame are self-explained. Updates (17) and (18) show that a receive by an agent is in tandem with a send by the intruder. Note that $analz$ and $analz^P$ are applied to the updated knowledge-set and updated frame, respectively.

Remark 1 Our protocol semantics above assumes a standard Dolev-Yao attacker. A possibly odd-looking aspect appears only in (10), where we explicitly name the new values $\sigma(\tau_c)$ generated by $\mathtt{I}$ for composing the sent message $\sigma(t)$. This is just so that, in (18), we can concretely record inside $\mathtt{I}$'s updated frame the mappings $\tau_c \mapsto \sigma(\tau_c)$, corresponding to these generated values $\sigma(\tau_c)$.

Definition 12 (The $update$ Function for an Event via a Branching Send Action.) Let s be a state, let $ag = \sigma(param(\eta_A), n)$ be the sending agent with A in an event $e = (act, \sigma, n)$ as follows:

$$act = i.A\ !\ :test(\Phi)\ \ (\mathcal{V}_{\tau_0})\ \tau_0\ \mid\ A\ !\ :test(\neg\Phi)(\mathcal{V}_{\tau_1})\ \tau_1.$$

Then, the *enabling conditions for* $s' = update(s, e)$ are:

$$step(s_{ag}) = i - 1 \tag{20}$$
$$\text{if } s \models_\sigma \Phi \text{ then } \ \sigma(\tau_0) \in \overline{terms(s_{ag}) \cup \sigma(\mathcal{V}_{\tau_0})} \tag{21}$$
$$\text{if } s \not\models_\sigma \Phi \text{ then } \ \sigma(\tau_1) \in \overline{terms(s_{ag}) \cup \sigma(\mathcal{V}_{\tau_1})} \tag{22}$$

and the *updated state* $s' = update(s, e)$ is

- for the honest agent ag and agents $ag' \in Ag \setminus \{ag, \mathtt{I}\}$:

the same as for simple send action using $t = \sigma(\tau_0)$ and $\mathcal{V}_{\tau_0}$ or $t = \sigma(\tau_1)$ and $\mathcal{V}_{\tau_1}$, depending on the truth of $s \models_\sigma \Phi$, with the exception of the frame of the intruder.

- for the intruder $\mathtt{I}$, irrespective of whether $s \models_\sigma \Phi$:

$$frame(s'_{\mathtt{I}}) := analz^P(terms(s'_{\mathtt{I}}, frame(s_{\mathtt{I}}) \cup \{\tau_0 \mapsto t, \tau_1 \mapsto t\})) \tag{23}$$

The update (23) stipulates that the intruder cannot predict the correct signature of an intercepted term, when the latter comes from a branching send. Hence, the intruder records both possible mappings. Afterwards, the intruder may make further deductions with one of the mappings.

Similarly, a receiving agent cannot know upfront the correct signature of an incoming term. Thus, the semantics of a branching receive follows the fashion of (23). Without repeating details to formally give this definition, we note that the frame update for the receiving agent (16) becomes:

$$frame(s'_{ag}) := analz^P(terms(s_{ag}, frame(s_{ag}) \cup \{\tau_0 \mapsto t, \tau_1 \mapsto t\}))$$

5) ***Executions****:* A protocol *execution* ξ is a sequence of events. Starting at an initial global state s^0 and applying in series the events in a sequence ξ results in a (global) system-state denoted $infstate(s^0, \xi)$, formalised below.

Definition 13 (States Updated via Executions.) Let s^0 be an initial global state, ξ be an execution, e be an event and the function $update$ be as per Definitions 10, 11, 12. The function $infstate$ is defined inductively as follows:

$$infstate(s^0, \varnothing) \ = s^0$$
$$infstate(s^0, \xi \cdot e) = update(infstate(s^0, \xi), e),$$

where $\varnothing$ denotes an empty sequence of events.

6) ***States Indistinguishability****:* To define state indistinguishability, we encode the "sense an agent makes" out of the terms, given his current state. For this, we use the notion of *pattern* [32], which intuitively says that "undecipherable" terms have the same "pattern"/meaning to an agent. We state the definition of patterns in Appendix A.

Definition 14 (Equivalences of Concrete Terms) We say that *two sets* $\mathbf{T}$ *and* $\mathbf{T}'$ *of concrete terms are equivalent*, denoted $\mathbf{T} \equiv \mathbf{T}'$, if $pat(\mathbf{T}) = pat(\mathbf{T}')$.

We are now in the position to define, what it means for an agent to tell apart two of their states. Agents reason on patterns of terms and can see decipherable terms for what they are, and all undecipherable terms are equal.

Definition 15 (States' Indistinguishability.) Let ag be an agent, and $s_{ag} = (\mathbf{T}, \mathbf{F}, step)$, $s'_{ag} = (\mathbf{T}', \mathbf{F}', step')$ be two of its local states. Then, s_{ag} and s'_{ag} are *indistinguishable* by ag, written $s_{ag} \approx s'_{ag}$, if there is a bijection θ on nonces and short-term keys such that:

1) $step = step'$, $\mathbf{T}\theta \equiv \mathbf{T}'$, and $\mathsf{dom}(\mathbf{F_T}) = \mathsf{dom}(\mathbf{F'}_{\mathbf{T}'})$
2) for all $d \in \mathsf{dom}(\mathbf{F_T})$, $\mathsf{card}(\mathsf{im}_{\mathbf{F_T}}(d)) = \mathsf{card}(\mathsf{im}_{\mathbf{F'}_{\mathbf{T}'}}(d))$
3) for all $d \mapsto t \in \mathbf{F_T}$, there exists $d \mapsto t' \in \mathbf{F'}_{\mathbf{T}'}$ such that $pat(t\theta, \mathbf{T}\theta) \ = \ pat(t', \mathbf{T}')$
4) for all $d \mapsto t' \in \mathbf{F'}_{\mathbf{T}'}$, there exists $d \mapsto t \in \mathbf{F_T}$ such that $pat(t\theta, \mathbf{T}\theta) \ = \ pat(t', \mathbf{T}')$,

where $\mathsf{card}(X)$ is the cardinality of a set X.

We say that two *global states* s *and* s' *are indistinguishable by agent* ag, written $s \sim_{ag} s'$, if the respective local state of ag are indistinguishable, i.e., if $s_{ag} \approx s'_{ag}$.

The definition above says that two states are indistinguishable to ag, when the three components of the states are indistinguishable: the step, the knowledge-set, and the frame. Recall that at each state-update frames are closed under $analz^P$; then, two frames are equivalent if they have the same domain of

abstract terms, the same size, and –as per points 3 and 4 in Definition 15– for each mapping in one, one can find an equivalent mapping in the other.

Example 7 Consider two local states $(\mathbf{T}, \mathbf{F}, step)$ and $(\mathbf{T}', \mathbf{F}', step)$ of some agent ag. Assume that:

$$\mathbf{T} = \{x, y, \{0\}_{\mathsf{pubk}(f(x,y))}\},$$
$$\mathbf{T}' = \{x, y, \{1\}_{\mathsf{pubk}(f(x,y))}\},$$
$$\mathbf{F} = \{X \mapsto x, Y \mapsto y, \{Z\}_{\mathsf{pubk}(f(X,Y))} \mapsto \{0\}_{\mathsf{pubk}(f(x,y))}\},$$
$$\mathbf{F}' = \{X \mapsto x, Y \mapsto y, \{Z\}_{\mathsf{pubk}(f(X,Y))} \mapsto \{1\}_{\mathsf{pubk}(f(x,y))}\}.$$

Then, knowledge-sets $\mathbf{T}$ and $\mathbf{T}'$ are equivalent according to Definition 14. To compare $\mathbf{F}$ and $\mathbf{F}'$, observe that all ingredients $0, x, y$ of the message $\{0\}_{f(x,y)}$ are in $\overline{\mathbf{T}}$ (since 0 and 1 are constants). By Definition 8-(2) of $analz^P$ (i.e., by public construction), $(Z \mapsto 0)$ is in $analz^P(\mathbf{T}, \mathbf{F})$. Similarly, $(Z \mapsto 1)$ is in $analz^P(\mathbf{T}', \mathbf{F}')$. Hence, the mapping of Z makes the two states distinguishable by ag.

However, if $\mathbf{T} = \{x, y, \{n\}_{f(x,y)}\}$ and $\mathbf{T}' = \{x, y, \{n'\}_{f(x,y)}\}$ for some nonces n, n' that are not in the knowledge-sets $\mathbf{T}, \mathbf{T}'$, then the two above states are indistinguishable because Definition 8-(2) would not apply (as nonces are not public constants).

IV. Epistemic Logic for Protocols' Privacy

We now present the syntax of $\mathcal{PL}$, our logic of knowledge for encoding privacy. Some of the illustrations are based on the *PrivAuth* protocol given in Example 1, and others on the so-called "Basic-Hash" protocol [7] described in Example 8.

A. Privacy Logic $\mathcal{PL}$ – Syntax

The syntax of $\mathcal{PL}$ is on top of our protocol signature $(\mathcal{F}, \mathcal{V})$ and domain $\mathtt{D}$ of concrete terms.

Logic Variables: First, we consider a set $\mathcal{X}_{Ag}$ *of logic agent variables* ag which are handles to agent. Second, we consider a set $\mathcal{X}$ *of logic protocol variables* $x, y, \ldots$. Variables in $\mathcal{X}$ reflect data linked to a protocol description, and include sorted variables overloading protocol variables. The latter are particularly useful when encoding tests in protocol actions (e.g., formula β in Example 2).

Logic Terms: Because we overload protocol variables to logic variables, our algebraic signature $(\mathcal{F}, \mathcal{V})$ becomes also a logic signature by natural extension, i.e., any functional symbol f becomes a first-order logic predicate f. So, any algebraic abstract term in $\mathcal{T}(\mathcal{F}, \mathcal{X})$ also has a corresponding *logic term*. The Greek letter θ denotes a generic logic term.

Index Logic Terms & Indexed Predicates: We define another set of terms called *index logic terms*: agent in Ag or logical variables in $\mathcal{X}_{Ag}$. These are used to index predicates and epistemic operators to create an indexed logic.

We introduce families of *predicates indexed on indexed terms*: $(has_{ag}(\cdot))_{ag \in Ag}$, $(link_{ag}(\tau, \cdot))_{ag \in Ag, \tau \in \mathcal{V}}$ and $(\cdot \in S_{ag})_{ag \in Ag}$. The predicate $has_{ag}(t)$ reads as "agent ag has the (concrete) term t", $link_{ag}(\tau, n)$ reads as "agent ag links the protocol variable τ to the (concrete) term t, $t \in S_{ag}$ reads as "the (concrete) term t belongs to ag's whitelist". Their meanings will be clarified in the semantics.

Formulas in $\mathcal{PL}$: These are given over the above indexed predicates, augmented with the family of *epistemic operators* $(K_{ag})_{ag \in Ag}$, logical connectives, and quantifications. The formula "$K_{ag}\, \phi$" reads: "agent ag knows ϕ".

Definition 16 ($\mathcal{PL}$ Syntax.) A $\mathcal{PL}$ *formula* is given by:

$$\varphi ::= has_u(\theta) \mid link_u(\tau, \theta) \mid \theta \in S_u \mid K_u \varphi \mid \neg\varphi \mid \varphi \wedge \varphi \mid \forall x : \mathtt{D}_X \cdot \varphi \mid \forall x : Ag \cdot \varphi$$

where θ is a logic term, u an index term, τ a logic term, x a logic variable in $\mathcal{X} \cup \mathcal{X}_{Ag}$, $\mathtt{D}_X$ one of $\mathtt{D}_{\mathcal{A}}, \mathtt{D}_{\mathcal{K}}$, and $\mathtt{D}_{\mathcal{N}}$.

B. Privacy Logics $\mathcal{PL}$ – Semantics

Variables Assignment: An *assignment* α *of logic variables* is the tuple $\alpha = (\alpha_1, \alpha_2)$ of two functions: the sort-respecting $\alpha_1 : \mathcal{X} \to \mathtt{D}$, and $\alpha_2 : \mathcal{X}_{Ag} \to Ag$.

Agents Active at a State: Assignments are used as per the usual in interpretations of logical formulae. In this vein, we note that logical protocol variables $x \in \mathcal{X}$ are interpreted over constant domain $\mathtt{D}$, whereas logical agent variables $x \in \mathcal{X}_{Ag}$ are interpreted over a varying domain, which is state-dependent, as per the following definition.

Definition 17 (Active Agents at State s.) The *domain* $Ag^{act}(s)$ *for agent variables' values at a state* s is the subset of Ag given by the agents active at s:

$$Ag^{act}(s) = \{ag \in Ag \mid s_{ag} \neq s^0_{ag}\}$$

The intruder $\mathtt{I}$ is part of the set $Ag^{act}(s)$ for any state s, whenever a protocol execution has started.

Values for Logic Terms: As per Section IV-A, our algebraic signature $(\mathcal{V}, \mathcal{F})$ is mapped to a logic signature, and our logic terms largely overload algebraic abstract terms. So, as algebraic abstract terms get mapped to concrete terms (i.e., values), so do logic terms, as defined below.

A *valuation* V^α *for logic terms and index logic terms under a variable assignment* α is a function, as follows:

- $V^\alpha(ag) = ag$, if $ag \in \mathcal{X}_{Ag}$;
- $V^\alpha(x) = \alpha(x)$, if $x \in \mathcal{X}$;
- $V^\alpha(f(\theta_1, \theta_2, \ldots)) = f(V^\alpha(\theta_1), V^\alpha(\theta_2), \ldots)$, for $f \in \mathcal{F}$.

Any concrete algebraic term t just becomes a *concrete logic term*, i.e., $V^\alpha(t) = t$.

Interpretation of $\mathcal{PL}$ formulas: First, we need to explain how the states of our transition-system in the protocol semantics are unravelled, in the shape needed for $\mathcal{PL}$ logic formulas to be interpreted.

Definition 18 (Kripke Models for a PS) The *unwound model of a* PS produced from initial states via executions is a Kripke structure $M = (W, (\sim_{ag})_{ag \in Ag})$, where W is the set of global states reachable via the *infstate* function, and $\sim_{ag}$ is the indistinguishable relation for each agent ag.

Now, we can give the truth valuation of $\mathcal{PL}$ formulas.

Definition 19 (Truth Valuation of a $\mathcal{PL}$ Formula.) Let $I^{\Sigma,Pr,n_{sess}}$ be a PS, $M = (W, (\sim_{ag})_{ag\in Ag})$ be its unwound model, and $s \in W$ be a global state in M. Let θ be a logical term, u be an agent index term, x be a logical variable, d be an abstract term, and α be a variable assignment.

The *truth valuation of a formula* $\varphi \in \mathcal{PL}$ *at* (M, s), denoted by $(M, s) \models \varphi$, is given as follows:

1. $(M,s) \models_\alpha \neg\Phi$ iff $(M,s) \not\models_\alpha \Phi$
2. $(M,s) \models_\alpha \Phi \wedge \Psi$ iff $(M,s) \models_\alpha \Phi$ and $(M,s) \models_\alpha \Psi$
3. $(M,s) \models_\alpha has_u(\theta)$ iff $V^\alpha(\theta) \in terms(s_{V^\alpha(u)})$
4. $(M,s) \models_\alpha \theta \in S_u$ iff $V^\alpha(\theta) \in S_{V^\alpha(u)}$
5. $(M,s) \models_\alpha link_u(d,\theta)$ iff $(d \mapsto V^\alpha(\theta)) \in frame(s_{V^\alpha(u)})$
6. $(M,s) \models_\alpha K_u\varphi$ iff for all $s' \in W$ with $s' \sim_{V^\alpha(u)} s$, $(M,s') \models_\alpha \varphi$
7. $(M,s) \models_\alpha \forall x{:}\mathtt{D}_X \cdot \varphi$ iff for all $t \in \mathtt{D}_X$ $(M,s) \models_{\alpha\cup\{x\mapsto t\}} \varphi$
8. $(M,s) \models_\alpha \forall x{:}Ag \cdot \varphi$ iff for all $ag \in Ag^{act}(s)$ $(M,s) \models_{\alpha\cup\{x\mapsto ag\}} \varphi$

where $\alpha \cup \{x \mapsto t\}$ is the extension of the variable assignment α with the mapping $x \mapsto t$ for x being as defined and t being a sort-respecting value in $\mathcal{D}$.

C. Informal Meaning of Formulas in $\mathcal{PL}$

Predicates has ***and*** $link$*:* The predicates has expresses a "possession-type" knowledge. For a concrete name $alice$, $has_{ag}(alice)$ says that agent ag possesses (can deduct) the name $alice$, and this is trivially true if names are public knowledge. On the other hand, $link$ associates a protocol variable to its instantiation in a protocol run, and so – it naturally expresses privacy. For instance, the statement $link_{ag}(vote_V, alice)$ holds when ag can deduct via $analz^P$ that the agent of role V voted for $alice$.

Knowledge Operators*:* Our epistemic operators are standard: an agent knows about the actual global state only as much as it can "see" from its local state. Thus, an agent ag knows a fact in a given global state s (in one run) iff that fact holds in any reachable global state s' that agent ag cannot distinguish from s (even across other runs).

Quantifications*:* Quantifications over one of the finite subdomains of $\mathtt{D}$ reduce to finite disjunctions and conjunctions.

When quantifying over agents, recall that we only quantify over *active agents* $Ag^{act}(s)$ (Definition 17) rather than the entire set Ag. To see the need for this, consider the formulas $\varphi_0 = \exists x : Ag \cdot named_x(alice) \wedge plays_x(A)$. If we let x to range over the whole Ag, then φ_0 would hold at any state. This is because the entire Ag contains every possible agent, for any name, playing any role. By restricting to $Ag^{act}(s)$, we get more information from φ_0: i.e., that there exists an agent named $alice$ who sent or received a message up to state s.

D. Equality in $\mathcal{PL}$ via $link$

Our definition of the semantics of $link$ (Definition 19-5) as a set-membership allows us to emulate term-equality with our logic. The following example illustrates this.

Example 8 (The Basic-Hash Protocol [7].) In this protocol, each tag T has a secret key KT. A reader R maintains a list S_R containing the secret keys of tags that the reader accepts. A tag T initiates a session by sending a fresh nonce NT and the hash of the pair $\langle NT, KT\rangle$. After receiving a message $\langle NT, \mathsf{hash}(\langle NT, KT\rangle)\rangle$, a reader computes $\mathsf{hash}(NT, x)$ for every key x in its database, until it matches $\mathsf{hash}(\langle NT, KT\rangle)$.

In our formalism, this protocol is modelled with a tag role, which consists of the send action

$$1.T\ !\ :\qquad (\{NT\})\langle NT, \mathsf{hash}(\langle NT, KT\rangle)\rangle,$$

and a reader role, which consists of the receive action

$$1.R\ ?\ : test(\mathsf{True}, \phi_{\text{lookup}})\quad \langle NT, \mathsf{hash}(\langle NT, KT\rangle)\rangle,$$

where ϕ_{lookup} is the formula
$\exists x : \mathtt{D}_\mathcal{K} \cdot x \in S_R \wedge link_R(\mathsf{hash}\langle NT, KT\rangle, \mathsf{hash}\langle NT, x\rangle)$.

As we said, we will explain that the $link$-based postcondition ϕ_{lookup} used in our encoding of this protocol emulates checking term equality.

Concretely, equality for hash checking needed in this this protocol is modelled via evaluating the postcondition ϕ_{lookup}, at the state of the receiving tag agent. To see this, assume that a tag-role agent ag is receiving $\sigma(\mathsf{hash}(\langle NT, KT\rangle)$ at one of its state s_{ag}. For agent ag to progress, the post-condition formula ϕ_{lookup} has to hold at one of ag's intermediate state s'_{ag} where $(\mathsf{hash}(\langle NT, KT\rangle)\rangle \mapsto \sigma(\mathsf{hash}\langle NT, KT\rangle)) \in frame(s'_{ag})$ and $\sigma(NT), \sigma(\mathsf{hash}\langle NT, KT\rangle) \in terms(s'_{ag})$. And, $s'_{ag} \models \phi_{\text{lookup}}$ holds only if there exists $k \in S_{ag}$, such that $(\mathsf{hash}(\langle NT, KT\rangle)\rangle \mapsto \mathsf{hash}\langle\sigma(NT), k\rangle)) \in frame(s'_{ag})$. This will be true only if the equality $\sigma(\mathsf{hash}\langle NT, KT\rangle) = \mathsf{hash}\langle\sigma(NT), k\rangle)$ holds. If the post-condition ϕ_{lookup} fails, then the receiving tag-role agent ag does not update its state.

Example 9, in Appendix B, gives an alternative modelling of hash/equality checking, with an explicit lookup-fail.

E. $\mathcal{PL}$'s Indexing over Agents & Names

In this subsection, we show how using quantification over agents of the same name can help us encode other privacy expressions. First, we define some "helper" predicates.

Helper Predicates*:* Just for ease of explaining our logics constructs, we define two helper predicates, defined based on the $link$ predicate, namely $named$ and $plays$, such that

$$named_{ag}(a) := link_{ag}(self, a)$$
$$plays_{ag}(A) := link_{ag}(A, name(ag)) \qquad (\text{for}^3\ ag \neq \mathtt{I}.)$$

The predicate $named_{ag}(a)$ means that the agent ag is named a, for $a \in \mathtt{D}_\mathcal{A}$. The predicate $plays_{ag}(A)$ means that the agent ag plays the role A.

Name-indexed Predicates*:* Let a be an agent name in $\mathtt{D}_\mathcal{A}$, θ a concrete logic term, and d an abstract term. We define:

$$has_a(\theta) := \exists x : Ag \cdot named_x(a) \wedge has_x(\theta)$$
$$link_a(d,\theta) := \exists x : Ag \cdot named_x(a) \wedge link_x(\tau, \theta)$$
$$\theta \in S_a := \exists x : Ag \cdot named_x(a) \wedge \theta \in S_x$$
$$plays_a(A) := \exists x : Ag \cdot named_x(a) \wedge plays_x(A)$$

First, the predicate has_a combines the knowledge-sets of (active) agents named a at the state where has_a is interpreted; so, $has_a(\theta)$ pins down the group of agents named a that have the term θ. Second, the predicate $link_a(\tau, \theta)$ denotes that some agent named a links the abstract term τ to the concrete term θ. Third, the predicate $\theta \in S_a$ is true if θ is in the whitelist of some (and every) agent named a that was active at the state where $\theta \in S_a$ is interpreted.

V. Encoding Privacy in $\mathcal{PL}$

In this section, we exemplify our encodings of privacy, anonymity and unlinkability via $\mathcal{PL}$ formulae. As before, we illustrate via the *PrivAuth* protocol given in Example 1, and the Basic-Hash protocol [7] shown in Example 8. Stemming from these two protocols, we speak of agents of role A and C (i.e., ag^A, ag^C) and agents of role tag T and reader R (i.e., ag^T, ag^R).

In Section VI, we verify these expressions shown here, as well as others.

A. *Anonymity*

Anonymity hides the identity of actions' performers. The *second goal of PrivAuth* is a "role-based anonymity" for the protocol-roles A and C. I.e., *anonymity of role* C means observers (possibly other than the intruder) cannot relate the C-role to the name of the agent who performs it.

We encode this via *minimal anonymity* and *total anonymity* à la [11]. "Minimal anonymity" corresponds to an identity remaining hidden with respect to an anonymity set of just two elements. In "total anonymity", this hiding is with respect to the largest "anonymity set" possible.

The minimal anonymity of C is expressed as follows.

Property 1 (*PrivAuth*'s 2nd Goal: Minimal Anonymity of C)

$$\forall x : Ag \cdot \forall a : \mathtt{D}^*_{\mathcal{A}} \cdot (\mathsf{pubk}(x) \notin S_a \wedge \neg named_x(a)) \Rightarrow \neg K_x(plays_a(C))$$

where $\mathsf{pubk}(x) \notin S_a := (\nexists b \cdot named_x(b) \wedge \mathsf{pubk}(b) \in S_a)$.

The implicant in Property 1 specifies the set of observers to be any agent not named a and outside a's whitelist; these should not know that agents named a play the responder role. The logic variable a is taken from the domain $\mathtt{D}^*_{\mathcal{A}}$, meaning we look at the anonymity of parties other than the intruder. However, as expected, the variable x for the observers ranges over active agents, which include the intruder.

In general: for an observer x not know $plays_a(C)$ at a state s, it suffices that x cannot distinguish s with another state s' where $plays_a(C)$ does not hold: at s', there is a name d, $d \neq a$, such that $plays_d(C)$. However, the concept of maximality in Property 2 below requires that *any other* name d may be playing C.

Property 2 (*PrivAuth*'s 2nd Goal: Total Anonymity of C)

$$\forall x : Ag \cdot \forall d : \mathtt{D}^*_{\mathcal{A}} \cdot \neg named_x(d) \Rightarrow \neg K_x(\neg plays_d(C))$$

Note the difference between the set of observers in Property 1 and 2. In Property 1, an agent x named b such that $\mathsf{pubk}(b) \in S_a$ may know that $plays_a(C)$ (by successfully executing the protocol). In contrast, in Property 2, such an agent should not know that a is not playing C.

B. *Privacy of Interlocutors.*

Now, we formulate the third goal of *PrivAuth*, which says that if $A \in S_B$, only B and A should know this: B by virtue of owning S_B, and A by executing the protocol. For this, we encode that observers from outside of their b's whitelist should not know *who is in* b's whitelist S_b:

Property 3 (*PrivAuth*'s Third Goal: Privacy of Interlocutors)

$$\forall x : Ag \cdot \forall a : \mathtt{D}_{\mathcal{A}} \cdot \forall b : \mathtt{D}^*_{\mathcal{A}} \cdot$$
$$(\neg named_x(a) \wedge \neg named_x(b)) \Rightarrow \neg K_x(\mathsf{pubk}(a) \in S_b)$$

This part of *PrivAuth*'s third goal says the following: when $\mathsf{pubk}(a) \in S_b$, an agent named a is allowed to know this; and whoever is not named a or b is not allowed to know this. The name b ranges over $\mathtt{D}^*_{\mathcal{A}}$, which (again) excludes the intruder, as we do not require the intruder's privacy.

Now we express the second part of the goal, i.e., that observers should not know *who is not in* b's whitelist S_b:

Property 4 (*PrivAuth*'s Third Goal: Privacy of Interlocutors)

$$\forall x : Ag \; \forall a : \mathtt{D}_{\mathcal{A}} \; \forall b : \mathtt{D}^*_{\mathcal{A}} \cdot$$
$$(\neg named_x(b) \wedge \neg named_x(a) \Rightarrow \neg K_x(\mathsf{pubk}(a) \notin S_b))$$

Property 4 requires that when $a \notin S_b$, the agent a should not know that. Intuitively, this is achieved in *PrivAuth* when a plays A and b plays C: the a-agent cannot distinguish between a decoy message saying "I am not b" and a decoy message saying "I am b, but I am not interested".

C. *Unlinkability*

In the ISO/IEC-15408 standard, unlinkability is expressed as the guarantee that a party may make multiple "appearances", without others being able to link them, e.g., an intruder not linking different "presences" of the same participant in different protocol sessions.

- To exemplify unlinkability, we use the Basic-Hash protocol [7] shown in Example 8, and we speak agents of tag role T (i.e., ag^T) and of reader role R (i.e., ag^R) .
- In the following expressions of unlinkability, we consider only the intruder to be the observer.

To ease the notation, we use the following syntactic sugar for quantifying over agents playing a specific role:

$$\exists x : Ag^T \cdot \phi \quad := \quad \exists x : Ag \cdot plays_x(A) \wedge \phi.$$

Strong Unlinkability*: Strong unlinkability* [2] requires that the intruder cannot distinguish a protocol execution comprised of *any* n concurrent sessions from an idealised execution formed by n sessions executed with n different parties. Strong unlinkability fails when the intruder knows that at least two, among n sessions, are linked. To ensure strong unlinkability, the following formula must hold for any n-sessions execution.

Note 2: If the attacker was –for instance– to link two sessions amongst themselves because a static information in each (e.g.,

a long-term of the party executing it), then strong unlinkability would fail trivially. Synonymously, a valid attack against this property can be "easy" to mount. We will detail this further in Note 3 and Note 4, in the context of weak unlinkability as well.

Property 5 (Strong Unlinkability by Name.)

$$\neg K_I\big(\exists t_1,\ldots,t_{n-1} : \mathtt{D}_{\mathcal{A}} \cdot \forall x : Ag^T \cdot \bigvee_{i\in\{1,\ldots,n-1\}} (named(x,t_i))\big)$$

Property 5 fails if the intruder knows that at most $n-1$ names of tags could be playing T, in the n observed sessions.

We continue with *strong session-unlinkability by key*. F or this, consider that a group of tags (playing role T) may share the same key KT.

Session-unlinkability by key prevents the intruder to know if the same key was used twice in n sessions; this is expressed in $\mathcal{PL}$ as follows:

Property 6 (Strong Unlinkability by Key.)

$$\neg K_I\big(\exists k_1,\ldots,k_{n-1} : \mathtt{D}_{\mathcal{A}} \cdot \forall x : Ag^T \cdot \bigvee_{i\in\{1,\ldots,n-1\}} (link_x(KT,k_i))\big)$$

Note 3: Strong unlinkability by name/by key fails when the intruder knows there are at least two T-sessions with the same name/key, although it does not know which sessions are linked. In other words, for strong unlinkability to fail, the intruder need not know the name/key explicitly or in a meaningfully-identifiable way. So, this unlinkability notion is "strong", since its failure is conceived by a rather "weak" attack.

Weak unlinkability: Weak unlinkability requires that the intruder does not know whether any two given sessions are linked to each other. To express weak unlinkability, we extend[4] our logic with equality between (honest) agents. Then, weak unlinkability of the tag T is given by:

Property 7 (Weak Unlinkability.)

$$\neg\big(\exists x_1 \neq x_2 : Ag^T \cdot K_I\big(\exists t : \mathtt{D}_{\mathcal{A}} \cdot \textstyle\bigwedge_{i\in\{1,2\}} named_{x_i}(t)\big)\big)$$

Property 7 is weaker (than the earlier notion of strong unlinkability), as it fails only when intruder can pinpoint the exact two sessions that are linked. To better see this contrast between weak and strong unlinkability, perhaps observe the following reformulation of strong unlinkability using agents (in)equality, below:

$$\neg\big(K_I\big(\exists x_1 \neq x_2 : Ag^T \cdot \exists t : \mathtt{D}_{\mathcal{A}} \cdot \textstyle\bigwedge_{i\in\{1,2\}} named_{x_i}(t)\big)\big)$$

Strong unlinkability, now as per the above formulation, fails if the intruder knows the existence of a link between two sessions, although it may not know which two sessions are actually linked (i.e., although, weak unlinkability as per formula 7 would not fail). So

Note 4: See that, in our weak-unlinkability encoding as per formula 7, the tag agents x_1, x_2, are existentially quantified *outside* the scope of K, whereas –in our weak-unlinkability encodings as per the formula 6– the tag agents x_1, x_2, are existentially quantified *inside* the scope of K. So, from a logic-formalism viewpoint, refuting the strong-unlinkability formula is easier compared to the weak strong-unlinkability; in the latter, (checking) its refutation would span the whole reachable state-space.

Remark 2 (**Weak and Strong Anonymity**) Analogously to the notions of weak and strong unlinkability, in fact, we can also distinguish between weak and strong anonymity (despite not doing so earlier in this section). In fact, in our earlier encodings of anonymity in Property 1 and Property 2, each corresponds to a notion of stronger anonymity. To see this, let us rewrite, e.g., Property 1 as a negation of an attack, and replace the syntactic sugar $plays_a(A)$ with its meaning; then, we get:

$$\neg\big(\exists x : Ag \cdot \exists a : \mathtt{D}^*_{\mathcal{A}} \cdot \neg named_x(a) \wedge \mathsf{pubk}(x) \notin S_a \wedge K_x(\exists y : Ag^A \cdot named_y(a))\big)$$

In the above, note that the A-agent y is existentially quantified *inside* the scope of K_x; so, strong anonymity here may fail simply when the observer x knows a played the role A, even if x does not know for a fact which A-agent y is identifiable as a (i.e., in which session a played).

Contrary to the above, weak anonymity is encoded as:

$$\neg\big(\exists x : Ag \cdot \exists a : \mathtt{D}^*_{\mathcal{A}} \cdot \exists y : Ag^A \cdot \neg named_x(a) \wedge \mathsf{pubk}(x) \notin S_a \wedge K_x(named_y(a))\big)$$

Now, note that for weak anonymity to fail, one needs a rather strong attack: i.e., this attack implies that x knows exactly that a fixed A-agent y is identifiable as a given a.

Like with unlinkability, strong anonymity is can be more readily embodied as plausibilistic privacy, whereas weak anonymity is less so.

VI. Mechanised Privacy Analysis

Now, we present `Phoebe`, an epistemic model checker for privacy analysis of cryptographic protocols, based on the protocol model and the logic in Sections III and IV.

The source code for `Phoebe` and use-cases are available at https://github.com/UoS-SCCS/phoebe.

A. *Overview of Our Model Checker* `Phoebe`

Our epistemic model checker `Phoebe` is written in `Haskell`. It takes a protocol specification (i.e., given via `Haskell` data types) and a privacy property expressed in $\mathcal{PL}$, and returns the truth value of the formula (or counter-example traces) on the model of the protocol given at input.

`Phoebe` works in the setting of bounded number of agents and domain $\mathtt{D}$. Agents in Ag are enumerated by exhausting combinations of agent parameters. We choose the domain-sizes for nonces and session-keys such that agents' logical omniscience [15] is not the case. For a given number of sessions, all possible executable traces between the agents are generated (with loose inspiration from [33]), creating session parameters (nonces and session-keys) on-the-fly.

[4]This can be done easily (e.g., by checking their assigned id in the modelling), and we omitted the formalisation of agents' equality, for sake of clarity.

Protocol	Formula	Scenario	Agents	Result	Time
PrivAuth	Neg. of Goal 3 Privacy of whitelists (who is in)	$\mathtt{D}_{\mathcal{A}}$=[a,b]	$4 \times A;\ 4 \times C$	unsat/private	17s
	Neg. of Goal 3' Privacy of whitelists (who is not in)	$\mathtt{D}_{\mathcal{A}}$=[a,b]	$4 \times A;\ 4 \times C$	unsat/private	34s
	Neg. of Goal 2A (Strong Minimal) Anonymity of Initiator A	$\mathtt{D}_{\mathcal{A}}$=[a,b]	$4 \times A;\ 4 \times C$	unsat/anonymous	34s
	Neg. of Goal 2C (Strong Minimal) Anonymity of Responder C	$\mathtt{D}_{\mathcal{A}}$=[a,b]	$4 \times A;\ 4 \times C$	unsat/anonymous	27s
PrivAuthX (without decoy)	Neg. of Goal 3 Privacy of whitelists (who is in)	$\mathtt{D}_{\mathcal{A}}$=[a,b]	$4 \times A;\ 4 \times C$	sat/attack	0.05s
	Neg. of Goal 3' Privacy of whitelists (who is not in)	$\mathtt{D}_{\mathcal{A}}$=[a,b]	$4 \times A;\ 4 \times C$	unsat/private	1.44s
	Neg. of Goal 2A (Strong Minimal) Anonymity of Initiator A	$\mathtt{D}_{\mathcal{A}}$=[a,b]	$4 \times A;\ 4 \times C$	unsat/anonymous	0.4s
	Neg. of Goal 2C (Strong Minimal) Anonymity of Responder C	$\mathtt{D}_{\mathcal{A}}$=[a,b]	$4 \times A;\ 4 \times C$	sat/attack	2.16s
BasicHash	Neg. of Strong Unlinkability by name	$\mathtt{D}_{\mathcal{A}}$=[t1,t2,r1]	$6 \times T;\ 1 \times R$	sat/attack	0.2s
	Neg. of Strong Unlinkability by name	$\mathtt{D}_{\mathcal{A}}$=[t1,t2,t3,r1]	$9 \times T;\ 1 \times R$	unsat/unlinkable	20s
	Neg. of Strong Unlinkability by name	$\mathtt{D}_{\mathcal{A}}$=[t1,t2,t3,r1]	$12 \times T;\ 1 \times R$	sat/attack	10s
	Neg. of Strong Unlinkability by name	$\mathtt{D}_{\mathcal{A}}$=[t1,t2,t3,t4,r1]	$16 \times T;\ 1 \times R$	unsat/unlinkable	51m
	Neg. of Strong Unlinkability by name	$\mathtt{D}_{\mathcal{A}}$=[t1,t2,t3,t4,t5,r1]	$25 \times T;\ 1 \times R$	time-out	>2h
	Neg. of Weak Unlinkability by name	$\mathtt{D}_{\mathcal{A}}$=[t1,t2,r1]	$6 \times T;\ 1 \times R$	unsat/unlinkable	0.2s
	Neg. of Weak Unlinkability by name	$\mathtt{D}_{\mathcal{A}}$=[t1,t2,t3,r1]	$9 \times T;\ 1 \times R$	unsat/unlinkable	2s
	Neg. of Weak Unlinkability by name	$\mathtt{D}_{\mathcal{A}}$=[t1,t2,t3,r1]	$12 \times T;\ 1 \times R$	unsat/unlinkable	31m
TagReader0	Neg. of Strong Unlinkability by key	$\mathtt{D}_{\mathcal{A}}$=[t1,t2,r1] $\mathtt{D}_{\mathcal{K}}$=[k1,k2]	$8 \times T;\ 1 \times R$	sat/attack	14s
	Neg. of Strong Unlinkability by name	$\mathtt{D}_{\mathcal{A}}$=[t1,t2,r1] $\mathtt{D}_{\mathcal{K}}$=[k1,k2]	$8 \times T;\ 1 \times R$	unsat/unlinkable	45s
LoRaWAN Join v1.1	Neg. of Unlinkabity of DevEUI (via DevAddr)	$\mathtt{D}_{\mathcal{A}}$=[d1,d2,s1]	$2 \times D;\ 1 \times S$	sat/attack	0.39s

TABLE I: Illustrative Results of Privacy Verification with Our Model Checker `Phoebe`

B. *Verifying Privacy & Unlinkability in* `Phoebe`

Now, we report the results and performance of our verification for our case studies. All the computations were performed on a MacBook Pro 2.6 GHz 6-Core Intel Core i7 with 16 GB of RAM. All results are summarised in Table I.

Let us first describe aspects of Table I, in general. We check formulae in $\mathcal{PL}$ which express the negation of privacy goals on well-studied, privacy-relevant protocols. If this negation is satisfied (e.g., "sat" in column 5), then there is a privacy attack, and if the negation holds (e.g., "unsat" in column 5), then the privacy goal mentioned in column 2 holds on the model considered (and we mark "private"/"anonymous", etc., in column 5). If an attack is not found, larger-bound models[5] could be checked (since our verification is bounded, thus not complete). Yet, clearly, if an attack is found, there is no more to check w.r.t. the protocol in case. Recall that $\mathtt{D}_{\mathcal{A}}$ means the domain of the names of the agents (e.g., 'a' – as in "alice", 'b' – as in "bob", 't1' – as in some tag), etc.

Note 5: The number of agents considered (e.g., column 4) is higher than needed for the verification undertaken, in order to report comparative timings. I.e., when the verification result is positive, we report the times for a large number of agents: e.g., 16, 12 for the Basic-Hash protocol.

All the verification results (i.e., privacy holding or failing) in Table I are in line with prior results in the literature, or in line with the expectation.

1) ***Privacy Verification for PrivAuth****:* We specified *PrivAuth* in `Phoebe`, in line with our encoding in Example 2 (see `PrivAuth.hs` in our source code). To see the value of decoy messages, we also encoded a "weakened" version of *PrivAuth*, without decoy messages. We denote this by *PrivAuthX*. We verified both *PrivAuth* and *PrivAuthX* against privacy properties written in $\mathcal{PL}$ in Section V. We report the results only for strong minimal and anonymity, and the two statement for the privacy of whitelists.

Results. `Phoebe` found no attack on *PrivAuth*, in the settings considered as per Table I. But, it did, as expected, on *PrivAuthX*; without decoy messages, in *PrivAuthX*, the presence of the responder C and the privacy of the responder's interlocutors list are compromised.

[5]This is the case of other tools in this domain, such as [25].

2) ***Unlinkability Analysis for the Basic-Hash Protocol****:* We specified and verified in `Phoebe` the *"Basic Hash" (BH)* protocol [34], [3] as given in our formalism (see Example 8). For this protocol, we verified "strong unlinkability" as well as "weak unlinkability" (by name), i.e., Property 5 and 7 as presented in Section V-C. Representative results are reported in Table I.

Results. Strong unlinkability fails only when the intruder observers more sessions than there are number of possible tags. An example attack-trace is found at our repository.However, weak unlinkability is never breached.

3) ***Further Case Studies****:* Two more case studies are presented in Table I and covered in Appendix C and Appendix D: a "Tag-Reader" protocol in [6] and the LoRaWAN Join v1.1 [24] against privacy goals recently stated in [35].

Our repository (https://github.com/UoS-SCCS/phoebe) also contains some example attack-traces.

C. *Limitations of Our Method and* `Phoebe`

The tool and the method verify bounded-size models, so they are incomplete, i.e., they may miss attacks. The method is sound, due to the Kripke formalisation: i.e., if an attack is found, it is correct. Hand in hand with this, `Phoebe` performs better on finding attacks, as it may search the whole state-space to prove a formula holds.

Removing the limitation on bounded-size models would not remove the incompleteness. This is due to two sources of undecidability: (a) Dolev-Yao privacy-analysis [22]; (b) epistemic satisfiability in infinite-state systems [36].

`Phoebe` can be further improved in maturity (as it is, above all, a proof-of-concept implementation for a new privacy-verification approach), and efficiency. One source of complexity is our naive implementation of model checking K. In the worst case, the evaluation of K scans the entire state-space of a privacy system. Next, we will check formulas with K at equivalence classes, not individual states. To bound less and improve performance, we will switch to lazy model-checking [36].

VII. Related Work

a) ***Principled Verification of Privacy with Epistemic Logics****:* Halpern and O'Neill [11] gave an epistemic characterisation of anonymity, without explicitly defining a cryptographic

indistinguishability relation. Similarly, the epistemic logic framework by Brusó et al. [3] formally expresses and compares different notions of unlinkability in a model that abstracts away cryptography. Syverson et al. [37] axiomatised a bespoke logic of knowledge for anonymity in a cryptographic setting. All forego model checking. To allow for verification, Tiplea et al. [22] reduced epistemic "minimal anonymity", in one-session, to a reachability property. On a parallel track, [38] and [39] proposed non-mechanised ideas of verifying epistemic privacy, in the specific domains of anonymous protocols, and e-voting, using a permutation-based cryptographic indistinguishability relation [38], or one based on pattern-matching of terms [39].
• Our cryptographic indistinguishability relation differ from both approaches, as it is defined on patterns' equivalence rather on frames.

• We summarise the relations to these epistemic-based works in Table II.

*b) **Privacy Verification with Non-Epistemic Logics**:* "α-β privacy" characterises privacy using logics [40]; Their attacker is mainly operating on an augmented frame, on which it attempts to "link" concrete (cryptographic) messages to abstract (cryptographic) ones. Their formula α specifications encapsulate what the attacker is cleared to know. We capture this directly in our privacy-system I, amounts to, e.g., domains of variables, initial-states setting. In the way of α holding in [40], we can query whether the attacker "K-knows" general, non-cryptographic facts. Also, the attacker will "K-know" privacy-relevant facts implicitly (e.g., due to determinism not due to cryptography): e.g., property 5/Example **??** fails due to this. W.r.t. the β formulae in [40], they amount to the links our attacker creates via $analz^P$, and thus via our indistinguishability relation $\sim$.

*c) **Automated Security and Privacy Verification with Epistemic Logics**:* Boureanu et al. [41], [21] developed automatable verification of epistemic properties. However, they cater only for security/authentication and not privacy. Tool-supports for epistemic encoding of privacy properties were given in [42] and [43]. [42] introduced the Dynamic Epistemic MOdelling tool for verifying anonymity, but not in a Dolev-Yao model. [43] analyses vote-privacy, via an ad-hoc compilation to a "non-Dolev-Yao" model-checker; thus, the loss/gap is unclear.

*d) **Automatic Tools for Privacy Verification in Security Protocols via Trace Equivalences**:* Privacy properties, such as anonymity and vote privacy, are often practically verified once expressed as trace equivalences. The most-known tools for checking trace-equivalence include DEEPSEC [25], AKiSs [44], Sat-Equiv [26], APTE [45], SPEC [46].

• These tools all differ on the class of protocols and the cryptographic primitives that they cover, as surveyed in [25]. All these tools consider a bounded number of sessions, as does `Phoebe`. Like DEEPSEC, `Phoebe` covers protocols with else branches. However, `Phoebe` is defined under a fixed equational theory.

*e) **Automatic Tools for Privacy Verification in Security Protocols via Approximations**:* Arapinis et al. [2] introduced the notion of strong unlinkability which cannot be encode as an equivalence between two traces. For 2-party protocols, Hirschi et al. [6], [7] reduced this unlinkability notion to two trace properties, plus "frame opacity" – a series of quantified static equivalences; yet, it is recognised that some observational equivalences (e.g., weak unlinkability [2]) cannot be reduced in these ways. In this domain, [47] recently put forward new process-algebraic observational equivalences, which are coarser than the classical ones [48], but possibly more amenable to direct verification of notions such as strong unlinkability by Arapinis et al.

• An epistemic formula in $\mathcal{PL}$ is generally stronger than trace equivalence, as it compares sets of traces ("congregated" in one state) to sets of traces ("congregated" in another state) [15]. For that reason, an epistemic formula in $\mathcal{PL}$ can be generally more expressive than Arapinis' unlinkability [49] too, as that compares a *specific* set of traces to a *specific* set of traces; i.e., that forms just a sub-class of models for our logic, not the full set of models for the logic [15].

*f) **Automatic Tools for Privacy Verification in Security Protocols via Diff-Equivalences**:* ProVerif [50], Tamarin [51] and Maude NPA [52] have been extended to verify privacy using diff-equivalence [53], [54], [55]. Whilst these tools allow for unbounded number of sessions, diff-equivalence is an over-approximation of observational equivalence, which may lead to false attacks w.r.t. privacy, or be impractical for properties such as unlinkability [4].

*g) **Summative Comparison between Automated Verification of Dolev-Yao Privacy**:* Tamarin and ProVerif can do verification of a bounded number of sessions, but privacy is only approximated therein. A new approach for verifying equivalence properties in an unbounded number of sessions was recently proposed in [56]. Meanwhile, DEEPSEC and `Phoebe` *handle only a bounded number of sessions, and* `Phoebe` *is also well-typed (i.e., it excludes type-flaw attacks).*

All the tools listed thusfar can only specify privacy against what the intruder know. In turn, Paverd et al., using CASPER, verified privacy expressed as trace properties against *honest-but-curious attacker* (legitimate parties who execute honestly the protocol but want to gain knowledge about other participants) [57].

As far as we know, only `Phoebe` can verify privacy w.r.t. the knowledge of legitimate parties, as well as precisely express the anonymity goals of the *PrivAuth*, (using the list of interlocutors) as they were originally stated in [23].

Table III summarises the comparison between `Phoebe` and some of the aforesaid tools. Figure 7 from [25] compares DEEPSEC with AKiSs [44], Sat-Equiv [26], APTE [45], SPEC [46], and DEEPSEC can handle the largest set of privacy properties, overall, and generally outperforms the other tools in efficiency (see Figure 7 from [25]). So, in our Table III, we can safely only compare with DEEPSEC. We reiterate that DEEPSEC and `Phoebe` are the only two tools, in Table III, catering primarily for privacy verification.

Appr. / Indist.		Syverson *et al.* [37] axiom. crypto.	Halpern *et al.* [11] semantics no crypto.	Garcia *et al.* [38] semantics crypto.	Cohen & Dam [20] semantics + axiom. crypto.	Baskar *et al.* [39] semantics crypto.	Brusó *et al.* [3] semantics no crypto.	$\mathcal{PL}$ semantics + tool crypto.
Anonym.	Min.	✓	✓	✓	✓	?	N/A	✓
	Tot.	✓	✓	✓	✓	?	N/A	✓
Unlink.	Weak	?	N/A	?	?	N/A	✓	✓
	Strong	?	N/A	?	?	N/A	✓	✓
Privacy of interloc.		?	N/A	✓	✓	?	N/A	✓
Vote Privacy		?	?	✓	✓	✓	N/A	✓

TABLE II: $\mathcal{PL}$ vs. Epistemic Logic Frameworks for Expressing and Verifying Privacy

Our Notion		Tamarin	ProVerif	DEEPSEC	Phoebe
Anonym.	Min.	✓O.A.,I.O. d-e[54]	✓O.A.,I.O. d-e [53]	✓I.O.	✓
	Tot.	?	?	?	✓
Unlink.	Weak	N/A	?	?	✓
	Strong	✓O.A. (3p. [7])	✓O.A. ([2]) ✓O.A. (3p. [6])	✓O.A. [58]	✓
Privacy of interloc.		✓O.A.,I.O. d-e	✓O.A.,I.O. d-e	✓I.O.	✓
Vote Privacy		✓O.A. d-e	✓O.A. d-e	✓	✓

TABLE III: `Phoebe` vs. Privacy Verification Tools for Protocols ("O.A." stands for "Over-Approximated"; "N/A" for "not applicable"; "?" for "never attempted"; "I.O." for privacy against intruder only; "d.-e." for diff-equivalence; "3p" for the 3 conditions for strong unlinkability by [6], [7])

VIII. Conclusions

We defined an epistemic logic $\mathcal{PL}$, syntactically and semantically, to express and verify privacy properties in cryptographic protocols, in a Dolev-Yao model. Our logic can express privacy goals w.r.t an intruder as well as legitimate protocol participants. Then, $\mathcal{PL}$ is a generic logic and expresses uniformly different flavours and strength privacy properties, from strong/weak minimal/total anonymity to strong/weak unlinkability. Finally, $\mathcal{PL}$ allows for automation. To this end, we provided a prototype model checker for $\mathcal{PL}$, named `Phoebe`. We tested `Phoebe` on several use cases, and found expected privacy attacks on Abadi's Private Authentication protocol [23], as well as expected strong unlinkability attacks against the Basic-Hash protocol [34].

References

[1] A. Pfitzmann and M. Hansen, "A terminology for talking about privacy by data minimization: Anonymity, Unlinkability, Undetectability, Unobservability, Pseudonymity, and Identity Management," https://dud.inf.tu-dresden.de/literatur/, Tech. Rep., 2010.

[2] M. Arapinis, T. Chothia, E. Ritter, and M. Ryan, "Analysing unlinkability and anonymity using the applied pi calculus," in *2010 23rd IEEE computer security foundations Symp.*, 2010, pp. 107–121.

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

[4] S. Delaune and L. Hirschi, "A survey of symbolic methods for establishing equivalence-based properties in cryptographic protocols," *J. of Logical and Algebraic Methods in Programming*, vol. 87, 2017.

[5] R. Horne, "A bisimilarity congruence for the applied pi-calculus sufficiently coarse to verify privacy properties," 2018. [Online]. Available: https://arxiv.org/abs/1811.02536

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

[7] D. Baelde, S. Delaune, and S. Moreau, "A method for proving unlinkability of stateful protocols," in *2020 IEEE 33rd Computer Security Foundations Symp. (CSF)*, 2020, pp. 169–183.

[8] I. Goriac, "Plausibilistic entropy and anonymity." *J. Wirel. Mob. Networks Ubiquitous Comput. Dependable Appl.*, vol. 5, no. 1, pp. 64–83, 2014.

[9] Y. Tsukada, H. Sakurada, K. Mano, and Y. Manabe, "On compositional reasoning about anonymity and privacy in epistemic logic," *Annals of Mathematics and Artificial Intelligence*, vol. 78, no. 2, pp. 101–129, July 2016.

[10] M. Blaauw, "The epistemic account of privacy," *Episteme*, vol. 10, no. 2, p. 167–177, 2013.

[11] J. Halpern and K. O'Neill, "Anonymity and information hiding in multiagent systems," in *16th IEEE Computer Security Foundations Workshop*, July 2003, pp. 75–88.

[12] Y. Tsukada, K. Mano, H. Sakurada, and Y. Kawabe, "Anonymity, privacy, onymity, and identity: A modal logic approach," in *2009 Int. Conference on Computational Science and Engineering*, vol. 3. IEEE, 2009, pp. 42–51.

[13] H. L. Jonker and W. Pieters, "Receipt-freeness as a special case of anonymity in epistemic logic," in *IAVoSS Workshop On Trustworthy Elections (WOTE 2006)*, 2006.

[14] J. Hintikka, *Knowledge and Belief, An Introduction to the Logic of the Two Notions*. Cornell University Press, 1962.

[15] R. Fagin, J. Halpern, Y. Moses, and M. Vardi, *Reasoning about Knowledge*. MIT Press, 1995.

[16] D. Dolev and A. Yao, "On the Security of Public-Key Protocols," *IEEE Transactionson Information Theory 29*, vol. 29, pp. 198–208, 1983.

[17] S. Kripke, "Semantic Analysis of Modal Logic (Abstract)," *J. of Symbolic Logic*, vol. 24, pp. 323–324, 1959.

[18] G. Danezis, "Better anonymous communications," Ph.D. dissertation, 2004.

[19] J. Y. Halpern and R. Pucella, "Modeling Adversaries in a Logic for Security Protocol Analysis," in *the Workshop on Formal Aspects of Security (FASec'02)*, ser. LNCS, vol. 2629, 2002, pp. 115–132.

[20] M. Cohen and M. Dam, "A complete axiomatization of knowledge and cryptography," in *LICS 2007*, 2007, pp. 77–88.

[21] I. Boureanu, P. Kouvaros, and A. Lomuscio, "Verifying security properties in unbounded multiagent systems," in *the 2016 Int. Conference on Autonomous Agents & Multiagent Systems*, 2016, pp. 1209–1217.

[22] F. Tiplea, L. Vamanu, and C. Varlan, "Reasoning about minimal anonymity in security protocols," *Future Generation Computer Systems*, vol. 29, no. 3, pp. 828 – 842, 2013.

[23] M. Abadi, "Private authentication," in *Int. Workshop on Privacy Enhancing Technologies*, vol. 2482, 2002, pp. 27–40.

[24] LoRa Alliance Technical Committee, "LoRaWAN Specification v1.1," LoRa Alliance, January 2017. [Online]. Available: https://lora-alliance.org/resource_hub/lorawan-specification-v1-1/

[25] V. Cheval, S. Kremer, and I. Rakotonirina, "DEEPSEC: Deciding Equivalence Properties in Security Protocols Theory and Practice," in *2018 IEEE Symp. on Security and Privacy (SP)*, 2018.

[26] V. Cortier, A. Dallon, and S. Delaune, "Sat-equiv: an efficient tool for equivalence properties," in *2017 IEEE 30th Computer Security Foundations Symposium (CSF)*. IEEE, 2017, pp. 481–494.

[27] L. Paulson, "The Inductive Approach to Verifying Cryptographic Protocols," *J. of Computer Security*, vol. 6, no. 1-2, pp. 85–128, Jan. 1998.

[28] R. Ramanujam and S. P. Suresh, "Deciding knowledge properties of security protocols," in *the 10th Int. conference on Theoretical Aspects of Rationality and Knowledge (TARK'05)*, 2005, pp. 218–235.

[29] N. Durgin, P. Lincoln, J. Mitchell, and A. Scedrov, "Multiset Rewriting and the Complexity of Bounded Security Protocols," *J. of Computer Security*, 2003.
[30] T. Fábrega, J. Herzog, and J. Guttman, "Strand Spaces: Proving Security Protocols Correct," *J. of Computer Security*, vol. 7, pp. 191–230, 1999.
[31] M. Rusinowitch and M. Turuani, "Protocol Insecurity with Finite Number of Sessions is NP-complete," in *the 14th IEEE Workshop on Computer Security Foundations (CSFW'01)*, 2001, pp. 174–187.
[32] M. Abadi and P. Rogaway, "Reconciling Two Views of Cryptography," in *Theoretical Computer Science: Exploring New Frontiers of Theoretical Informatics*, ser. LNCS, 2000, pp. 3–22.
[33] D. Basin, "Lazy infinite-state analysis of security protocols," in *Int. Exhibition and Congress on Network Security*, 1999, pp. 30–42.
[34] S. A. Weis, S. E. Sarma, R. L. Rivest, and D. W. Engels, "Security and privacy aspects of low-cost radio frequency identification systems," in *Security in pervasive computing*, 2004, pp. 201–212.
[35] K. Budykho, I. Boureanu, S. Wesemeye, F. Rajaona, D. Romero, Lewis, Y. Rahulan, and S. Schneider, "Fine-Grained Trackability in Protocol Executions," in *Network and Distributed System Security Symposium (NDSS) 2023*, 2023.
[36] A. Cimatti, M. Gario, and S. Tonetta, "A lazy approach to temporal epistemic logic model checking," in *AAMAS*, 2016, pp. 1218–1226.
[37] P. Syverson and S. Stubblebine, "Group principals and the formalization of anonymity," in *Formal Methods (FM), Volume I*, 1999, pp. 814–833.
[38] F. Garcia, I. Hasuo, W. Pieters, and P. Van Rossum, "Provable Anonymity," in *FMSE 2005*. ACM Press, 2005.
[39] A. Baskar, R. Ramanujam, and S. Suresh, "Knowledge-based modelling of voting protocols," in *the 11th conference on Theoretical aspects of rationality and knowledge*, 2007, pp. 62–71.
[40] S. Mödersheim and L. Viganò, "Alpha-beta privacy," *ACM Trans. Priv. Secur.*, vol. 22, no. 1, pp. 7:1–7:35, 2019.
[41] I. Boureanu, M. Cohen, and A. Lomuscio, "Model Checking Detectability of Attacks in Multiagent Systems," in *AAAMAS'10*, 2010.
[42] J. van Eijck and S.Orzan, "Epistemic verification of anonymity," *Electr. Notes Theor. Comput. Sci.*, vol. 168, pp. 159–174, 2007.
[43] I. Boureanu, A. V. Jones, and A. Lomuscio, "Automatic verification of epistemic specifications under convergent equational theories," in *AAMAS 2012*. IFAAMAS, 2012, pp. 1141–1148.
[44] R. Chadha, Ş. Ciobâcă, and S. Kremer, "Automated verification of equivalence properties of cryptographic protocols," in *Programming Languages and Systems*, 2012, pp. 108–127.
[45] V. Cheval, "APTE: An Algorithm for Proving Trace Equivalence," in *Tools and Algorithms for the Construction and Analysis of Systems*. Springer Berlin Heidelberg, 2014, pp. 587–592.
[46] A. Tiu, N. Nguyen, and R. Horne, "SPEC: an equivalence checker for security protocols," in *Programming Languages and Systems: 14th Asian Symposium, APLAS 2016, Hanoi, Vietnam, November 21-23, 2016, Proceedings 14*. Springer, 2016, pp. 87–95.
[47] R. Horne, S. Mauw, and S. Yurkov, "Compositional analysis of protocol equivalence in the applied-pi-calculus using quasi-open bisimilarity," in *Theoretical Aspects of Computing – ICTAC 2021*. Springer Int. Publishing, 2021, pp. 235–255.
[48] M. Abadi and C. Fournet, "Mobile values, New Names, and Secure Communication," in *the 28th ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages (POPL'01)*, 2001, pp. 104–115.
[49] M. Arapinis, L. I. Mancini, E. Ritter, and M. D. Ryan, "Analysis of privacy in mobile telephony systems," *Int. J. Inf. Sec.*, vol. 16, no. 5, pp. 491–523, 2017.
[50] B. Blanchet, "An efficient cryptographic protocol verifier based on prolog rules," in *the 14th IEEE Workshop on Computer Security Foundations*, ser. CSFW '01. IEEE Computer Society, 2001, pp. 82–. [Online]. Available: http://dl.acm.org/citation.cfm?id=872752.873511
[51] S. Meier, B. Schmidt, C. Cremers, and D. Basin, "The tamarin prover for the symbolic analysis of security protocols," in *the 25th Int. Conference on Computer Aided Verification*, ser. CAV 2013. Springer-Verlag, 2013, pp. 696–701.
[52] S. Escobar, C. Meadows, and J. Meseguer, *Maude-NPA: Cryptographic Protocol Analysis Modulo Equational Properties*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 1–50.
[53] B. Blanchet, M. Abadi, and C. Fournet, "Automated verification of selected equivalences for security protocols," *The Journal of Logic and Algebraic Programming*, vol. 75, no. 1, pp. 3–51, 2008.
[54] D. Basin, J. Dreier, and R. Sasse, "Automated symbolic proofs of observational equivalence," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, 2015, pp. 1144–1155.
[55] S. Santiago, S. Escobar, C. Meadows, and J. Meseguer, "A formal definition of protocol indistinguishability and its verification using maude-NPA," in *International Workshop on Security and Trust Management*. Springer, 2014, pp. 162–177.
[56] V. Cheval and I. Rakotonirina, "Indistinguishability beyond diff-equivalence in proverif," in *2023 2023 IEEE 36th Computer Security Foundations Symposium (CSF)(CSF)*. IEEE Computer Society, 2023, pp. 552–567.
[57] A. Paverd, A. Martin, and I. Brown, "Modelling and automatically analysing privacy properties for honest-but-curious adversaries," *Tech. Rep*, 2014.
[58] V. Cheval, C. Jacomme, S. Kremer, and R. Künnemann, "SAPIC+: protocol verifiers of the world, unite!" in *USENIX Security'22*, Boston, MA, Aug. 2022, pp. 3935–3952.

APPENDIX A
PATTERNS OF TERMS

Definition 20 (Patterns of Concrete Terms.) Consider a set $\mathbf{T}$ of concrete terms and a concrete term $t \in \mathbf{T}$. The *pattern* pat *of* t *at* $\mathbf{T}$ is a function defined on $\mathbf{T}$ such that

$$
\begin{aligned}
pat(t, \mathbf{T}) &= \begin{cases} b, \text{ if } b \in \mathcal{D} \cap \mathbf{T} \\ \square, \text{ if } b \in \mathcal{D} \setminus \mathbf{T} \end{cases} \\
pat((t_1, t_2), \mathbf{T}) &= (pat(t_1, \mathbf{T}), pat(t_2, \mathbf{T})) \\
pat(\mathsf{pubk}(t), \mathbf{T}) &= \mathsf{pubk}(pat(t, \mathbf{T})) \\
pat(\mathsf{seck}(t), \mathbf{T}) &= \mathsf{seck}(pat(t, \mathbf{T})) \\
pat(\{t\}_{\mathsf{pubk}(a)}, \mathbf{T}) &= \begin{cases} \{pat(t)\}_{\mathsf{pubk}(a)}, \text{ if } \mathsf{seck}(a) \in \mathbf{T} \\ \square, \text{ if } \mathsf{seck}(a) \notin \mathbf{T} \end{cases} \\
pat(\{|t|\}_k, \mathbf{T}) &= \begin{cases} \{pat(t)\}_k, \text{ if } k \in \mathbf{T} \\ \square, \text{ if } k \notin \mathbf{T} \end{cases} \\
pat(\mathsf{hash}(t), \mathbf{T}) &= \begin{cases} \mathsf{hash}(pat(t)), \text{ if } t \in \mathbf{T} \\ \square, \text{ if } t \notin \mathbf{T} \end{cases}
\end{aligned}
$$

The *patterns* $pat(\mathbf{T})$ *of a set* $\mathbf{T}$ *of concrete terms* is given as $pat(\mathbf{T}) = \{pat(t, analz(\mathbf{T})) \mid t \in analz(\mathbf{T})\}$.

APPENDIX B
SPECIFYING THE BASIC-HASH PROTOCOL [34]

Now, we show an alternative modelling of the Basic-Hash protocol [34] in our formalism.

Example 9 To make the lookup fail more explicit in the Basic-Hash protocol (Example 8), one can model the receiving agent to do separately the receive action and the lookup action, instead of a single post-conditioned receive action. In that case, the role of the reader would contain the following actions

$$
\begin{aligned}
&1.R\ ?\ : && \langle NT, \mathsf{hash}(\langle NT, KT\rangle)\rangle, \\
&2.R\ !\ : test(\phi_{\text{lookup}}) && \text{ok} \\
&|R\ !\ : test(\neg\phi_{\text{lookup}}) && \text{error},
\end{aligned}
$$

where ϕ_{lookup} is the formula
$\exists x : \mathsf{D}_{\mathcal{K}} \cdot x \in S_R \wedge link_R(\mathsf{hash}\langle NT, KT\rangle, \mathsf{hash}\langle NT, x\rangle)$, and where ok and test are constant strings.

APPENDIX C
VERIFYING THE TAG-READER PROTOCOL [6] IN PHOEBE

Another use-case is the a tag-reader (*TR*) protocol (Example 17 in [6]). This is as follows. A tag `T` first sends the encryption of a fresh nonce `NT` with the shared key `KT`. Upon receiving `{NT}_KT`, a reader also sends the encryption of its fresh nonce `NR` with the same shared key `KT`. The tag accepts `{NR}_KT` only if `NR` is not a term that it already has, after which it sends the encryption of the pair $\langle$`NR,NT`$\rangle$. The reader accepts the second encryption message if it matches its expected message.

The TR Protocol in `Phoebe`: For `TagReader0`, we verified the properties (strong) "unlinkability by name" and "unlinkability by key", i.e., our properties 5 and 6 in VI-B. We allowed names t_i to play only the tag's role, and names r_i to play only the reader's role. We report the results in Table I with two sessions and, two tag names `{t1, t2}` such that `t1` and `t2` may have the same or different keys from `{k1, k2}`.

Results: For this protocol, `Phoebe` found no attack on "strong unlinkability by name", for all the tests. [6] informally discussed the fact that this protocol might be exploited by the intruder to track of a group of tags using the same key. We formally defined this property as "unlinkability by key" in Section V. As expected, attacks were found on "unlinkability by key" for `TagReader0`.

APPENDIX D
VERIFYING LORAWAN JOIN V1.1 [24] IN PHOEBE

We now present the LoRaWAN Join and a summary of its verification in `Phoebe`.

1) The LoRaWAN Join: LoRaWAN in the most used IoT specification. When LoRAWAN devices join an IoT network, they run the protocol in Figure 1, i.e., the LoRaWAN Join v1.1 protocol [24]. We describe it here succinctly (due to space constraints), but sufficiently for privacy concerns.

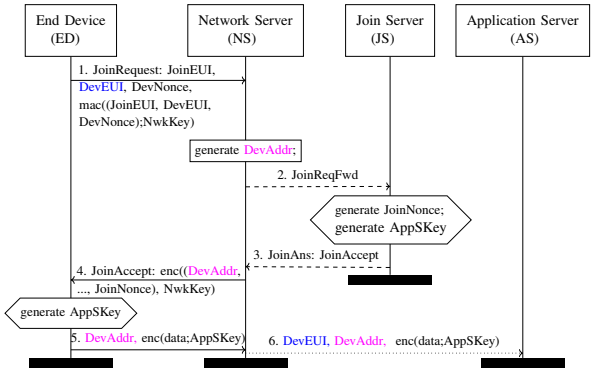

Fig. 1: LoRaWAN Join v1.1[6]

In this protocol, an End Device (ED) with a long-term identifier *DevEUI* contacts a *Join Server (JS)* via a *Network Server (NS)*, such that the ED establishes a new session key, *AppSKey*, to use with an *Application Server (AS)*.

[6]Dashes vs lines: secure vs insecure channels; dots: down to implementer

In this process, the NS generates an ephemeral identifier, *DevAddr*, for the device, which the device will use in all the headers of its application messages until it rejoins (e.g., days or weeks later).

2) LoRaWAN-Join's Privacy Provision.: Figure 1 shows which messages are encrypted and which are not, and we see that the long-term identifier *DevEUI* appears in clear in message 1. We also see the in message 6, on the "backend" between the NS and the AS, the *DevEUI* and the *DevAddr* appear jointly.

Privacy question: Assuming that the attacker does not have access to the backend, i.e., not to message 6, is the attacker able to link a DevEUI to its assigned DevAddr?

This requirement can be expressed in $\mathcal{PL}$ as follows.

$$\neg(\exists x : \mathtt{D}_{\mathcal{A}}, y : \mathtt{D}_{\mathcal{K}}, ag : Ag \cdot K_I(\, plays_{ag}(ED) \wedge link_{ag}(DevEUI, x) \wedge link_{ag}(DevAddr, y)))$$

We model LoRaWAN Join in `Phoebe`, as one would expect from the description above and our formalism.

We verified this is `Phoebe` and we report that in Table I.

Note 6: LoRaWAN Join v1.1 protocol [24] was not specified with the privacy in mind, but this attack was recently discussed in [35], and now this is current topic of interest for the upcoming v1.2. specifications (see https://resources.lora-alliance.org/, talks at Expo 2022).

APPENDIX E
LIST OF SYMBOLS

Below, we provide a non-exhaustive list of the symbols used in this work and their meaning to help the reader.

Terms

	Abstract terms	Concrete terms
Names	$A, B, C, T, R, \ldots (\in \mathcal{A})$	$alice, bob, t_1, t_2, \ldots (\in \mathtt{D}_{\mathcal{A}})$
Nonces	$N_A, N_B, \ldots (\in \mathcal{N})$	$n, n_a, n_b, \ldots (\in \mathtt{D}_{\mathcal{N}})$
List	$S_A, S_B, \ldots (\in \mathcal{S})$	$\{\mathsf{pubk}(bob)\}, \{k_1, k_2\}, \ldots$
Shared key	$K_{AB}, K_T, \ldots (\in \mathcal{K})$	$k_{ab}, k, \ldots (\in \mathtt{D}_{\mathcal{K}})$
Generic	$\tau \in \mathcal{T}(\mathcal{F}, \mathcal{V})$	$t \in \mathcal{T}(\mathcal{F}, \mathtt{D})$

Protocol Model

$\mathcal{V}$	protocol variables $\mathcal{A} \cup \mathcal{N} \cup \mathcal{K}$
$\mathcal{F}$	functional symbols ($= \mathcal{F}_c \cup \mathcal{F}_d$)
$\mathcal{F}_0$	public constants
$\mathcal{F}_c^*$	public constructor symbols
$\mathcal{V}_\tau$	a subset of $\mathcal{V}$
$\mathbf{T}, \mathbf{M}$	set of concrete terms (knowledge-set)
$\mathbf{F}$	frame
$\mathtt{D}$	domain of concrete terms $\mathtt{D}_{\mathcal{N}} \cup \mathtt{D}_{\mathcal{A}} \cup \mathtt{D}_{\mathcal{K}}$
σ	a substitution, i.e., in $\mathcal{V} \to \mathcal{T}(\mathcal{F}, \mathtt{D})$
Σ	a (finite) set of substitutions
ag	an agent (representing a session of a participant)
s	a global state of the system
s_{ag}	local state of agent ag

Logic Syntax

x, y, z	logic variables
θ	logic term $\mathcal{T}(\mathcal{F}, \mathtt{D} \cup \mathcal{X})$

Logic Semantics

α	an assignment of logic variables
V^α	valuation of logic terms under α
$M = (W, \{\sim_{ag}\}_{ag \in Ag})$	an epistemic model
W	a set of global states (possible-worlds)
$\sim_{ag}$	indistinguishability relation of agent ag