Privacy as Reachability

Sébastien Gondron
DTU Compute
Danmarks Tekniske Universitet
Kgs. Lyngby, Danmark
sebastien.gondron@riseup.net

Sebastian Mödersheim
DTU Compute
Danmarks Tekniske Universitet
Kgs. Lyngby, Danmark
samo@dtu.dk

Luca Viganò
Department of Informatics
King's College London
London, United Kingdom
luca.vigano@kcl.ac.uk

2022 IEEE 35th Computer Security Foundations Symposium (CSF) | 978-1-6654-8417-6/22/$31.00 ©2022 IEEE | DOI: 10.1109/CSF54842.2022.9919668

***Abstract*—We show that privacy can be formalized as a reachability problem. We introduce a transaction-process formalism for distributed systems that can exchange cryptographic messages (in a black-box cryptography model). Our formalism includes privacy variables chosen non-deterministically from finite domains (e.g., candidates in a voting protocol), it can work with long-term mutable states (e.g., a hash-key chain) and allows one to specify consciously released information (e.g., number of votes and the result). We discuss examples, e.g., problems of linkability, and the core of the privacy-preserving proximity tracing system DP-3T.**

***Index Terms*—Formal Methods. Protocol security. Transition system. Linkability. DP-3T.**

I. Introduction

Privacy-type properties of security and voting protocols are often specified as *trace equivalence of two processes* in some process calculus, such as the Applied-π calculus [1, 6, 10, 13]. While such approaches have uncovered vulnerabilities in several protocols, they rely on asking whether the intruder can distinguish two variants of a process; for instance, the intruder should not be able to detect a difference between two processes differing only by the swap of the votes of two honest voters. It is quite hard to intuitively understand what such a trace equivalence goal actually entails and what not, and one may wonder if there are other trace equivalences that should be checked. It is a rather technical way to encode the privacy goals of a protocol, and although one can get insights from a failed proof when the goal is too strong, one cannot easily see when it is too weak.

To fill the gap between the intuitive ideas of the privacy goals and the mathematical notions used to formalize and reason about them, (α, β)-*privacy* has been proposed in [20]. It is a declarative approach based on specifying two formulae α and β in first-order logic with Herbrand universes. α formalizes the *payload*, i.e., the "non-technical" information, that we intentionally release to the intruder, and β describes the "technical" information that he has, i.e., his "actual knowledge": what (names, keys, etc.) he initially knows, which actual cryptographic messages he observed and what he infers from them. He may be unable to decrypt a message, but know anyway that it has a certain format and contains certain (protected) information. Consider, for instance, the unlinkability goals in protocols for RFID tags used in electronic passports. In a state where two sessions of the protocol have been initiated, we may have $\alpha \equiv T_1 \in \mathsf{Tags} \wedge T_2 \in \mathsf{Tags}$, where T_1 and T_2 are free variables. This specifies the goal that the intruder does not know more about T_1 and T_2 than: they are tags. In particular, he should not be able to find out whether $T_1 \doteq T_2$. If β (what he learned from observing and interacting with the tags) allows the intruder to derive $\beta \models T_1 \doteq T_2$, then β violates the privacy of α. We will make all of this formal below.

The main difficulty in reasoning about privacy with trace equivalence is that one needs to consider two possible worlds: for every step the first system can make, one has to show that the other system can make a similar step so that they are still indistinguishable. To tame this problem, several works limit themselves to *bi-processes*, i.e., where the two processes can only differ in subterms of messages. Bi-processes allow one to obtain a verification question that is close to a reachability problem, but at the price of drastically limiting the range of protocols that can be considered. What distinguishes (α, β)-privacy from trace equivalence is that it considers *one* possible world rather than two.

This provides a stepping stone for a privacy approach based on reachable states without the limitations of bi-processes. However, until now, (α, β)-privacy is only a static approach that does not reason about the development of a system, like the influence the actions of an intruder can have on the system.

The *first contribution* of this paper is to lift (α, β)-privacy from a static approach to a dynamic one. We define a transaction-process formalism for distributed systems that can exchange cryptographic messages (in a black-box cryptography model). Our formalism

- includes privacy variables that can be non-deterministically chosen from finite domains (e.g., the candidates in a voting protocol),
- can work also with long-term mutable states (e.g., modeling a hash-key chain), and
- allows one to specify the consciously released information (e.g., the number of cast votes and the result).

The core of this definition is a semantics as a state-transition system. This keeps track of what the intruder knows about the system, in particular modeling the several possibilities of what *could* have happened that are not (yet) ruled out by observations of the intruder. We define *dynamic* (α, β)-*privacy* to hold if (static) (α, β)-privacy hold in every reachable state of the transition system. Hence, every state is an (α, β)-privacy problem, i.e., a pure reachability problem, that supports a wide variety of privacy goals.

© 2022, Sébastien Gondron. Under license to IEEE.
DOI 10.1109/CSF54842.2022.00009

Formalizing privacy as a reachability problem, as dynamic (α, β)-privacy allows us to do, provides a first step towards automating the analysis, but it does not solve all the challenges of automation: (i) (α, β)-privacy is in general undecidable, but for most reasonable protocols it can be reduced to static equivalence problems (cf. Theorem 3) and is thus decidable for all algebraic theories for which static equivalence is; (ii) the set of reachable states is infinite. Symbolic and abstract interpretation methods still need to be developed, although a first result exists [15].

We argue that our approach is also very helpful for manual analysis, because it is a novel view of privacy that allows us to characterize the reachable states in a declarative logical way, and analyze the dynamic (α, β)-privacy question for them. As a *second contribution* we consider the core of the privacy-preserving proximity tracing system DP-3T [23] as a topical case study. It turns out that the system actually releases slightly more information than we initially thought. This is discovered because for our first specification of α, i.e., what information is deliberately released, the proof of (α, β)-privacy fails: the system actually leaks slightly more information than α. In this situation one has the choice to either change the system or allow for the leak by augmenting the α-release. Such a step-by-step augmentation is indeed a methodology to understand the privacy of a system: we start by considering a minimal α to represent our intention of the released information and augment it until we can finally succeed in proving dynamic (α, β)-privacy—thus obtaining a complete (and rather declarative) characterization of all the information that the system discloses.

As a *third contribution*, we formalize the relationship between our approach and trace equivalence (Th. 2 and 3). This serves two purposes. First, it helps us understand the relative strengths of different approaches, in particular that (α, β)-privacy has at least the expressive power of other approaches, while still allowing one to consider a reachability problem. Second, it paves the road to automation by relating to problems for which algorithms already exist, such as static equivalence for various algebraic theories of the cryptographic operators.

II. Preliminaries

A. Herbrand Logic

(α, β)-privacy is based on specifying two formulae α and β in *First-Order Logic with Herbrand Universes*, or *Herbrand Logic* for short [17]. For brevity, we only list the differences with respect to standard first-order logic (*FOL*).

Herbrand Logic fixes the universe in which to interpret all symbols. We introduce a signature $\Sigma = \Sigma_f \cup \Sigma_i \cup \Sigma_r$ with Σ_f the set of *uninterpreted function symbols*, Σ_i the set of *interpreted function symbols* and Σ_r the set of *relation symbols*. Let $\mathcal{T}_{\Sigma_f}$ be the set of ground terms that can be built using symbols in Σ_f and let $\approx$ be a congruence relation on $\mathcal{T}_{\Sigma_f}$; then we define the *Herbrand Universe* as the quotient algebra $\mathcal{A} = \mathcal{T}_{\Sigma_f}/_{\approx} = \{[\![t]\!]_{\approx} \mid t \in \mathcal{T}_{\Sigma_f}\}$, where $[\![t]\!]_{\approx} = \{t' \mid t \in \mathcal{T}_{\Sigma_f} \wedge t \approx t'\}$. The algebra fixes the "interpretation" of all uninterpreted function symbols: $f^{\mathcal{A}}([\![t_1]\!]_{\approx}, \ldots, [\![t_n]\!]_{\approx}) = [\![f(t_1, \ldots, t_n)]\!]_{\approx}$.

The interpreted function symbols Σ_i and the relation symbols Σ_r behave as in FOL, i.e., as function and relation symbols on the universe. To highlight the distinction between uninterpreted and interpreted function symbols, we write $f(t_1, \ldots, t_n)$ if $f \in \Sigma_f$ and $f[t_1, \ldots, t_n]$ if $f \in \Sigma_i$. Given a signature Σ, a set $\mathcal{V}$ of variables distinct from Σ, and a congruence relation $\approx$, and thus fixing a universe A, we define an *interpretation* $\mathcal{I}$ (with respect to Σ, $\mathcal{V}$, and $\approx$) as a function such that: $\mathcal{I}(x) \in A$ for every $x \in \mathcal{V}$; $\mathcal{I}(f)\colon A^n \mapsto A$ for every $f/n \in \Sigma_i$ of arity n; and $\mathcal{I}(r) \subseteq A^n$ for every $r/n \in \Sigma_r$ of arity n. Note that the functions of Σ_f are determined by the quotient algebra. We define a *model relation* $\mathcal{I} \models \phi$ (in words: ϕ holds under $\mathcal{I}$) as is standard and use notation like $\phi \models \psi$.

Let Σ_f contain the constant 0 and the unary function s, and let Σ_i contain the binary function $+$, i.e., the universe contains the natural numbers $0, s(0), s(s(0)), \ldots$, which we also write as $0, 1, 2, \ldots$ We characterize $+$ by the axiom $\alpha_{ax} \equiv \forall x, y.\ x + 0 = x \ \wedge \ x + s(y) = s(x + y)$[1].

We employ standard syntactic sugar and write, e.g., $\forall x.\ \phi$ for $\neg\exists x.\ \neg\phi$, and $x \in \{t_1, \ldots, t_n\}$ for $x = t_1 \vee \ldots \vee x = t_n$. Slightly abusing notation, we will also consider a substitution $\{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$ as a formula $x_1 = t_1 \wedge \ldots \wedge x_n = t_n$.

B. Encoding of Frames

We use, as it is customary in security protocol analysis, a black-box algebraic model. We choose a subset $\Sigma_{op} \subseteq \Sigma_f$ of uninterpreted functions to be the *operators* available to the intruder. For instance, we generally require $0, s \in \Sigma_{op}$, so the intruder can "generate" any natural number. We call these symbols *public* and we call *private* the symbols from $\Sigma \setminus \Sigma_{op}$. In order to represent the intruder's knowledge, we use frames.

Definition 1 (Frame)**.** *A* frame *is written as* $F = \{\!|m_1 \mapsto t_1, \ldots, m_l \mapsto t_l|\!\}$*, where the* m_i *are distinguished constants called* labels *and the* t_i *are terms that do not contain any* m_i*. We call* $m_1, \ldots, m_l$ *the* domain *and* $t_1, \ldots, t_l$ *the* image *of the frame. We write* $F\{\!|t|\!\}$ *for replacing in the term* t *every occurrence of* m_i *with* t_i*, i.e.,* F *works like a substitution.*

The labels m_i can be regarded as *memory locations* of the intruder, representing that the intruder knows the messages t_i. The set of *recipes* is the least set that contains $m_1, \ldots, m_l$ and that is closed under all the cryptographic operators in Σ_{op}.

We use two frames $struct = \{\!|l_1 \mapsto t_1, \ldots, l_n \mapsto t_n|\!\}$ and $concr = \{\!|l_1 \mapsto t'_1, \ldots, l_n \mapsto t'_n|\!\}$ that always share the same domain D in any formula. Let $concr[\cdot]$ and $struct[\cdot]$ be unary function symbols, and $gen(\cdot)$ a unary relation symbol defined by the following axioms:

$$\phi_{gen} \equiv \forall r.gen(r) \Leftrightarrow \big(r \in D \vee \bigvee\nolimits_{f^n \in \Sigma_{op}} \exists r_1, \ldots, r_n.\ r = f(r_1, \ldots, r_n) \wedge gen(r_1) \wedge \ldots \wedge gen(r_n)\big)$$

[1]This characterization is only possible due to the expressive power of Herbrand logic (in FOL one cannot characterize the universe appropriately).

$$\phi_{hom} \equiv \bigwedge_{f^n \in \Sigma_{op}} \forall r_1, \ldots, r_n.\ gen(r_1) \wedge \ldots \wedge gen(r_n) \implies concr[f(r_1, \ldots, r_n)] = f(concr[r_1], \ldots, concr[r_n]) \wedge struct[f(r_1, \ldots, r_n)] = f(struct[r_1], \ldots, struct[r_n])$$

$$\phi_{dom} \equiv struct[l_1] = t_1 \wedge \ldots \wedge struct[l_n] = t_n \wedge concr[l_1] = t'_1 \wedge \ldots \wedge concr[l_n] = t'_n$$

$$\phi_{\sim} \equiv \forall r, s.\ gen(r) \wedge gen(s) \implies concr[r] = concr[s] \Leftrightarrow struct[r] = struct[s]$$

ϕ_{gen} defines the predicate gen to be exactly the set of recipes for the frames $concr$ and $struct$. ϕ_{hom} and ϕ_{dom} specify the two frames $concr$ and $struct$ with domain $D = \{l_1, \ldots, l_n\}$ and formalize that $struct[t]$ and $concr[t]$ is the result of applying recipe t to the frames, i.e., replacing every occurrences of a label m_i by the corresponding t_i in t. Finally, $\phi_{\sim}$ means that $concr$ and $struct$ are *statically equivalent* (we write $concr \sim struct$): for any pair of recipes r and s that the intruder can generate, $concr$ agrees on r and s iff $struct$ does.

C. Alpha-Beta-Privacy

The distinction between payload and technical information is at the core of (α, β)-privacy. We formalize it by a distinguished subset $\Sigma_0 \subset \Sigma$ of the alphabet, where Σ_0 contains only the non-technical information, such as votes and addition, while $\Sigma \setminus \Sigma_0$ includes cryptographic operators. The formula α is always over just Σ_0, whereas β is over the full Σ.

Definition 2 (Static (α, β)-privacy [20])**.** *Consider a countable signature Σ and a payload alphabet $\Sigma_0 \subset \Sigma$, a formula α over Σ_0 and a formula β over Σ s.t. $fv(\alpha) \subseteq fv(\beta)$, both α and β are consistent and $\beta \models \alpha$. We say that* (α, β)-privacy holds (statically) *iff every Σ_0-model of α can be extended to a Σ-model of β, where a Σ-interpretation $\mathcal{I}'$ is an extension of a Σ_0-interpretation $\mathcal{I}$ if they agree on all variables and all the interpreted function and relation symbols of Σ_0.*

In contrast to [20], we allow β to have more free variables than α. All α formulae we consider in this paper are *combinatoric*, meaning that Σ_0 is finite and contains only uninterpreted constants. Then α has only finitely many models.

The common equivalence-based approaches to privacy are about the distinguishability between two alternatives. In contrast, (α, β)-privacy represents only one single situation that can occur, and it is the question what the intruder can deduce about this situation. To model this, we formalize that the intruder not only knows some concrete messages, but also that the intruder may know something about the structure of these messages, e.g., that a particular encrypted message contains a vote v_1, where v_1 is a free variable of α. Hence, we define the intruder knowledge by two frames $concr$ and $struct$, where $struct$ formalizes the *structural knowledge* of the intruder and thus may contain free variables of α, and the frame for the *concrete knowledge* $concr$ is the same except that all variables are instantiated with what really happened, e.g., $v_1 = 1$. The main idea is that we require as part of β that $struct$ and $concr$ are statically equivalent, which means that if the intruder knows that two concrete constructible messages are equal, then also their structure has to be equal, and vice versa.

Example 1. *As an example, let us consider a simplistic voting protocol. The voters choose their vote v_i from the payload alphabet $\Sigma_0 = \{0, 1\}$. Let $h \in \Sigma_{op}$ (h does not have a destructor), and, as part of the protocol, the voting server publishes messages of the form $h(v_i)$. Then, for two voters, we have the frames $struct = \{m_1 \mapsto h(v_1), m_2 \mapsto h(v_2)\}$ and $concr = \{m_1 \mapsto h(0), m_2 \mapsto h(1)\}$. Suppose now that we have the following α, and that β is constructed from the conjunction of α and the axioms we have introduced:*

$$\alpha \equiv v_1, v_2 \in \{0, 1\}$$
$$\beta \equiv \alpha \wedge \phi_{gen} \wedge \phi_{hom} \wedge \phi_{dom} \wedge \phi_{\sim}$$

α expresses that the intruder knows that the voters choose the votes from the set $\{0, 1\}$. β contains α, the specification of the frames $struct$ and $concr$, and the ability to compare them. Then, from β follows the property $v_1 \neq v_2$. (α, β)-privacy is violated, since, for instance, $v_1 = 0, v_2 = 0$ is a model of α, but cannot be extended to a model of β. In this situation, one can choose to allow for the leak, i.e., set $\alpha \equiv v_1, v_2 \in \{0, 1\} \wedge v_1 + v_2 = 1$, then all models of α are compatible with β and privacy holds. Rather than allowing the leak, better would be to change the system, e.g., by adding a fresh nonce as part of the published message: $h(n_i, v_i)$.

In the following, we assume β in every state to be implicitly augmented by the respective α and by the axioms ϕ_{gen}, ϕ_{hom}, ϕ_{dom} and $\phi_{\sim}$ where D is the set of labels occurring in β.

III. Transition Systems for Alpha-Beta-privacy

We lift the definition of static (α, β)-privacy to a dynamic one with transition systems. In §III-A, we describe the syntax of a protocol specification, notably the syntax of processes. We give the operational semantics for transition systems in §III-B and define the state with, among other things, the following formulae: the *payload formula* α, the *technical information formula* β and the *truth formula* γ. We also define a sequence of *conditional updates* δ on memory cells. In §A, we show how to derive a legitimate linkability attack on the OSK protocol.

A. Syntax

We consider a number of *transaction processes* and a number of families of *memory cells*, which allow us to model the stateful nature of some protocols. These cells can be used, for instance, to store the status of a key (e.g., valid or revoked).

We define *protocol specifications* in Definition 3 below. A specification must fix Σ, Σ_0 and fix an interpretation of all the interpreted symbols in Σ_i except for the built-in ones $struct[\cdot]$, $concr[\cdot]$ and $gen(\cdot)$. Moreover, we forbid the use of $struct[\cdot]$, $concr[\cdot]$ and $gen(\cdot)$ in process specifications.

In the processes, we talk about privacy variables. Each of them has a domain $D = \{c_1, \ldots, c_n\}$, where $c_1, \ldots, c_n$ are constants, i.e., a variable will be instantiated to one of these values. We consider only finite domains. This is not a restriction, since it is possible to leave the size of the model as a parameter in all definitions.

Definition 3 (Syntax)**.** *A* protocol specification *consists of:*

- *a number of families of* memory cells*, e.g.,* $\mathsf{cell}(\cdot)$*, together with an initial value which is a ground context* $k([\cdot])$*, so that initially* $\mathsf{cell}(t) = k([t])$*,*
- *a number of* transaction processes *of the form* $\mathcal{P}_l$*, where* $\mathcal{P}_l$ *is a left process according to the syntax below, and*
- *an initial state (see Definition 5), containing, e.g., domain specific axioms in the formulae* α *and* β *(see §II-C).*

We define left processes *and* right processes *as follows:*

$$\begin{array}{ll} \mathcal{P}_l ::= \mathrm{mode}\ x \in D.\mathcal{P}_l & \mathcal{P}_r ::= \mathsf{snd}(t).\mathcal{P}_r \\ \quad \mid \mathsf{rcv}(x).\mathcal{P}_l & \quad \mid \mathsf{cell}(s) := t.P_r \\ \quad \mid x := \mathsf{cell}(s).\mathcal{P}_l & \quad \mid \mathrm{mode}\ \phi.\mathcal{P}_r \\ \quad \mid \mathsf{if}\ \phi\ \mathsf{then}\ \mathcal{P}_l\ \mathsf{else}\ \mathcal{P}_l & \quad \mid 0 \\ \quad \mid \nu\overline{N}.\mathcal{P}_r & \end{array}$$

where x *ranges over variables;* mode *is either* $\star$ *or* $\diamond$*,* D *is the finite domain of a non-deterministic variable;* s *and* t *range over terms,* cell *over families of memory cells,* ϕ *over Herbrand formulae; and* $\overline{N}$ *is a set of fresh variables, i.e., that do not occur elsewhere in a left process. We introduce a meta-notation: a Herbrand formula* ϕ *in the* mode *in* $\mathcal{P}_r$ *may contain* $\mathcal{I}(t)$ *if* t *is a term.*

The syntactic structure of left and right processes ensures that the steps in a transaction can only occur in a particular order. In the first (left) part, we have the "incoming" aspects, like receiving messages and reading from memory, and then in the second (right) part, we have the "outgoing" aspects, like sending messages and writing to memory. This corresponds to the typical workflow, e.g, of a server or device API. Note that all bindings of variables take place in the left part. The naming of left and right is inspired by (multi-)set rewriting rules where the left part corresponds to requirements for applying the rule and the right part enacts the changes. The situation is however different in our formalism, because the conditionals give rise to several different possible executions. Thus in our case, every transaction is always applicable, but one can run into, e.g., an else case with the 0 process, meaning that the process makes no changes to the state and produces no outgoing message.

Let us look first at the left-side actions."$\mathrm{mode}\ x \in D$" means picking non-deterministically a value from domain D for the privacy variable x. Here, mode is either $\diamond$ or $\star$. $\diamond$ means that the variable x is of a low-level technical nature, i.e., it is not considered a privacy breach if the intruder should find out the value of x (but we do not directly give this information to the intruder). Thus, the formula $x \in D$ will be added as a new conjunct to the formula β of the current state. $\star$ means that x is high-level information, i.e., the intruder should not find out anything about x (unless we deliberately release later some information about x). Thus, $x \in D$ will be added as a new conjunct to both α and β in the current state. All these changes of α and β will be made precise in the formal semantics below.

The other left-hand constructs are pretty standard: "$\mathsf{rcv}(x)$" is message input, where the variable x is replaced with an actual received message. "$x := \mathsf{cell}(s)$" means reading the memory cell $\mathsf{cell}(s)$ into variable x. The conditional "$\mathsf{if}\ \phi\ \mathsf{then}\ \mathcal{P}_l\ \mathsf{else}\ \mathcal{P}_l$" is as expected. "$\nu\overline{N}.\mathcal{P}_r$" creates a sequence of fresh and distinct variables.

On the right-hand, we have "$\mathsf{snd}(t)$" for message output, "$\mathsf{cell}(s) := t$" means writing the term t into the memory cell $\mathsf{cell}(s)$. The step "$\mathrm{mode}\ \phi$" is a specialty of (α, β)-privacy where again mode is either $\star$ or $\diamond$, and where the meta-notation $\mathcal{I}(t)$ allows us to refer to the concrete value of t in a formula ϕ (see Example 8). If the mode is $\star$, this means that we deliberately release the information ϕ, i.e., it is added as a conjunct to α. This is crucial in specifying the privacy goals, since we determine in this way positively what the intruder is allowed to know (and everything else would be a violation of privacy). For the $\mathrm{mode} = \diamond$, this means the formula ϕ is added to γ. Finally, "0" is the null process.

We may write "let $x = t$" for the substitution of all following occurrences of x by t. We use a for construct as a syntactic sugar, e.g. for x: Agent. We need to unroll this loop, i.e., repeat the body for each agent. This syntactic sugar allows us to keep our formalization parametrized over an arbitrary number of agents, while a concrete specification that results from unrolling this loop has the number of agents fixed. Another syntactic sugar concerns parsing of messages. For many (cryptographic) operators we may have a corresponding *destructor* and *verifier*. Let f/n be a destructor and v/n a corresponding verifier; then we may write "$\mathsf{try}\ t = f(t_1, \ldots, t_n)\ \mathsf{in}\ \mathcal{P}_1\ \mathsf{catch}\ \mathcal{P}_2$" in lieu of "$\mathsf{if}\ v(t_1, \ldots, t_n) \doteq \mathsf{true}\ \mathsf{then}\ \mathsf{let}\ t = f(t_1, \ldots, t_n).\mathcal{P}_1\ \mathsf{else}\ \mathcal{P}_2$".

Example 2. *Let us illustrate our try and catch syntactic sugar with the functions* $\mathsf{pair}/2$*,* $\mathsf{proj}_1/1$ *and* $\mathsf{vpair}/1$ *with the properties* $\mathsf{proj}_i(\mathsf{pair}(t_1, t_2)) \approx t_i$ *and* $\mathsf{vpair}(\mathsf{pair}(t_1, t_2)) \approx \mathsf{true}$*:*

$$\begin{array}{l} \mathsf{try}\ t = \mathsf{proj}_1(\mathsf{pair}(t_1, t_2))\ \mathsf{in}\ \mathsf{send}(t) \\ \mathsf{catch}\ \mathsf{send}(\mathsf{error}) \end{array}$$

is syntactic sugar for

$$\begin{array}{l} \mathsf{if}\ \mathsf{vpair}(\mathsf{pair}(t_1, t_2)) \doteq \mathsf{true}\ \mathsf{then} \\ \quad \mathsf{let}\ t = \mathsf{proj}_1(\mathsf{pair}(t_1, t_2)).\mathsf{send}(t) \\ \mathsf{else}\ \mathsf{send}(\mathsf{error}) \end{array}$$

In the try construct, t is substituted in $\mathcal{P}_1$ and, as for the else branch in the conditional construct, we may omit the catch branch when $\mathcal{P}_2$ is the null process. Let us now look at a first example of processes.

Example 3 (Basic Hash)**.** *As a first example, we consider the* Basic Hash protocol *[7]: a reader can access a database of authorized tags that carry a mutable state. We consider* n *tags in the domain* $\mathsf{Tags} = \{t_1, \ldots, t_n\}$*. Let* $\mathsf{sk}/1$ *be a private function. Each tag* T *has an immutable secret key* $\mathsf{sk}(T)$*. Let* $h/2$*,* $\mathsf{pair}/2$*,* $\mathsf{vpair}/1$ *and* $\mathsf{proj}_i/1$ *be public functions as before. The tag sends messages of the form of a pair of a fresh nonce and the hash of the same nonce and its secret key.*

Tag	*Reader*
$\star\ T \in \mathsf{Tags}.$	$\mathsf{rcv}(t).$
$\nu N.\mathsf{snd}(\mathsf{pair}(N, h(\mathsf{sk}(T), N))).0$	$\mathsf{try}\ R = \mathsf{extract}(t)\ \mathsf{in}$
	$\quad \mathsf{snd}(\mathsf{ok}).0$

When the reader receives a message from a tag T, it has first to figure out who T is by trying all known keys $\mathsf{sk}(T)$ of any token T, almost like a guessing attack (this ensures T is a valid tag from Tags*). In order not to have to describe this procedure as transactions (it is included in the intruder model if he knows any keys), we simply define two special private functions for the reader (*$\mathsf{extract}/1$ *and* $\mathsf{vextract}/1$*) that check if a message is valid and extract T from it such that* $\mathsf{extract}(\mathsf{pair}(N, h(\mathsf{sk}(T), N))) \approx \mathsf{sk}(T)$ *and* $\mathsf{vextract}(\mathsf{pair}(N, h(\mathsf{sk}(T), N))) \approx \mathsf{true}$.

Definition 4 (Requirements on Processes)**.** *We require that α formulae are over Σ_0 and contain only variables that were released in α. In "*$\text{mode } x \in D.\mathcal{P}_l$*", "*$\mathsf{rcv}(x).\mathcal{P}_l$*" and "*$x := \mathsf{cell}(s).\mathcal{P}_l$*", we require that x cannot be instantiated twice, i.e., $\mathcal{P}_l$ contains neither "*$\text{mode } x \in D'$*", nor "*$\mathsf{rcv}(x)$*", nor "*$x := \mathsf{cell}(s')$*". We also require that in different branches of conditionals, the same non-deterministic variables are chosen in the same order and from the same set of values, and the ordering with receive steps is also the same. This is formalized by the following function that is only defined when the requirements are met:*

$$\begin{aligned}
varseq(\text{mode } x \in D.\mathcal{P}_l) &= \text{mode } x \in D.varseq(\mathcal{P}_l)\\
varseq(\mathsf{if}\ \phi\ \mathsf{then}\ \mathcal{P}_1\ \mathsf{else}\ \mathcal{P}_2) &= varseq(\mathcal{P}_1)\\
\mathsf{if}\ varseq(\mathcal{P}_1) = varseq(\mathcal{P}_2)&\ \textit{and undefined otherwise}\\
varseq(\mathsf{rcv}(t).\mathcal{P}_l) &= \mathsf{rcv}(t).varseq(\mathcal{P}_l)\\
varseq(_.\mathcal{P}_r) &= varseq(\mathcal{P}_r)\\
varseq(0) &= 0
\end{aligned}$$

Finally, we require that every transaction in a protocol specification is a closed process*, i.e., it has no free variables and the* binding occurrence *of a variable is the first occurrence where in the context it is not free (so further occurrences do not open a new scope):*

$$\begin{aligned}
fv(\text{mode } x \in D.\mathcal{P}_l) &= fv(\mathcal{P}_l) \setminus \{x\}\\
fv(\mathsf{rcv}(x).\mathcal{P}_l) &= fv(\mathcal{P}_l) \setminus \{x\}\\
fv(x := \mathsf{cell}(s).\mathcal{P}_l) &= (fv(s) \cup fv(\mathcal{P}_l)) \setminus \{x\}\\
fv(\mathsf{if}\ \phi\ \mathsf{then}\ \mathcal{P}_1\ \mathsf{else}\ \mathcal{P}_2) &= fv(\phi) \cup fv(\mathcal{P}_1) \cup fv(\mathcal{P}_2)\\
fv(\nu\overline{N}.\mathcal{P}_r) &= fv(\mathcal{P}_r) \setminus \overline{N}
\end{aligned}$$

B. Operational Semantics

We describe the operational semantics that lifts the definition of static (α, β)-privacy to a dynamic one with transition systems: intuitively, we define *dynamic (α, β)-privacy*, to hold if (α, β)-privacy holds in every state of the transition system. We first define states as tuples $(\alpha, \beta, \gamma, \delta)$ where α and β are as before, γ is a formula representing the ground truth and δ records conditional updates, i.e., write operations along with the conditional context under which they appear in the execution. We will then define a transition relation on states induced by the transaction processes. In §A, we give a detailed example of the application of our semantics.

Definition 5 (State)**.** *A* state *is a tuple $(\alpha, \beta, \gamma, \delta)$, where:*

- *formula α over Σ_0 is the* released information*,*
- *formula β over Σ is the* technical information *available to the intruder, such that β is consistent and entails α (thus also α is consistent and $fv(\alpha) \subseteq fv(\beta)$[2]),*
- *formula γ over Σ_0 is the* truth*, which is true for exactly one model with respect to the free variables of α and Σ_0, and $\gamma \wedge \beta$ is consistent,*
- *δ is a sequence of* conditional updates *of the form $\mathsf{cell}(s) := t$ if ϕ, where s and t are terms and ϕ is a formula over Σ, and its free variables are a subset of the free variables of α.*

The formulae α and β play the same roles as in the previous section. To define our transition system, we introduce the formula γ that represents the "truth", i.e., the real execution of a protocol. For instance, for a voting protocol, α may contain $v_i \in \{0, 1\}$ (i.e., that vote v_i is one of these values), β may contain cryptographic messages that contain v_i, and γ may contain $v_i \doteq 1$, i.e., what the vote actually is (and this is not visible to the intruder). The consequences of γ is what really happened, e.g., the result that one can derive from the votes in γ is the true result of the election. The sequence δ represents in a symbolic way all updates that a protocol may have performed on the memory cells: when updates are under a condition, the intruder does not know whether they where executed, so each update operation in δ comes with a condition ϕ; these entries in general contain variables when the intruder does not know the concrete values.

During the execution of a transaction, the intruder will in general not know what exactly is happening, in particular in a conditional, he will generally not know which branch has been taken. To model this precisely, our semantics models how the intruder can "symbolically execute" the transaction step by step in his mind: in particular for an if ϕ then $\mathcal{P}_1$ else $\mathcal{P}_2$ the intruder only knows that either ϕ is true and the process is now executing $\mathcal{P}_1$ or $\neg\phi$ holds and the process is now executing $\mathcal{P}_2$. Now if, for instance, $\mathcal{P}_1$ would send out a message and $\mathcal{P}_2$ would not, then the intruder can rule out one of the possibility, depending on whether he observes a message or not. Similarly, if both $\mathcal{P}_1$ and $\mathcal{P}_2$ send a message, then the intruder might still be able to tell whether it is a $\mathcal{P}_1$-message or a $\mathcal{P}_2$-message, and thus still rule out one of the possibilities.

Our semantics now models this symbolic execution by the intruder, including the management of several possibilities that the intruder at the current point cannot rule out, which one might call the *ignorance of the intruder*. In a given state $(\alpha, \beta, \gamma, \delta)$ and given a transaction process, we will step by step execute the process, appropriately splitting into different possibilities, where we always have one possibility marked as being *what really happened*. From the final set of possibilities obtained at the end of the process, we then derive the new state $(\alpha', \beta', \gamma', \delta')$ that reflects how the execution of the transaction has changed the world and the intruder's knowledge about it.

Definition 6 (Possibility, configuration)**.** *A* possibility *$(P, \phi, struct)$ consists of a process P, a formula ϕ over Σ and a frame $struct$ representing the structural knowledge attached to this process P.*

[2] [20] only allowed $fv(\alpha) = fv(\beta)$, but our constructions don't require it.

TABLE I
SUMMARY OF NORMALIZATION AND EVALUATION RULES

Normalization rules		Evaluation rules	
Redundancy	(NR 1.)	Non-deterministic choice	(ER 1.)
Redundant entries in δ	(NR 2.)	Marked process receives	(ER 2.)
Cell Read	(NR 3.)	Marked process sends	(ER 3.)
Conditional	(NR 4.)	Marked process has terminated	(ER 4.)
Cell Write	(NR 5.)		
Release	(NR 6.)		

A configuration *is a pair $(\mathcal{S}, \mathcal{P})$, where $\mathcal{S}$ is a state and $\mathcal{P}$ is a non-empty finite set of possibilities s.t.:*

- *$fv(\mathcal{P})$ is a subset of the free variables of $\mathcal{S}$,*
- *exactly one element of $\mathcal{P}$ is marked as the* actual possibility*, which we depict by underlining that element,*
- *the formulae $\phi_1, \ldots, \phi_n$ of $\mathcal{P}$ are mutually exclusive (i.e., $\models \neg\phi_i \vee \neg\phi_j$ for $i \neq j$) and β implies their disjunction (i.e., $\beta \models \phi_1 \vee \ldots \vee \phi_n$), and*
- *$\beta \wedge \gamma \models \phi$ for the condition ϕ of the marked possibility.*

We call a configuration $(\mathcal{S}, \mathcal{P})$ ready *if $P = 0$ for all $(P, \phi, struct) \in \mathcal{P}$.*

For a ready configuration $(\mathcal{S}, \mathcal{P})$ in the protocol, we can start the execution of any transaction $\mathcal{P}_l$ from the protocol description with an initial configuration of $\mathcal{P}_l$ defined as:

Definition 7 (Initial configuration of a transaction)**.** *Consider a ready configuration $(\mathcal{S}, \mathcal{P})$, a transaction process $\mathcal{P}_l$, a substitution θ that substitutes the fresh variables $\overline{N}$ (from a $\nu\overline{N}.\mathcal{P}_r$ specification) with fresh and distinct constants from $\Sigma \setminus \Sigma_0$ that do not occur elsewhere in the description or in $(\mathcal{S}, \mathcal{P})$, and that replaces all other variables with fresh variables that do not occur elsewhere in the description or in $(\mathcal{S}, \mathcal{P})$. The* initial configuration of $\mathcal{P}_l$ w.r.t. $(\mathcal{S}, \mathcal{P})$ and θ *is* $(\mathcal{S}, \{(\theta(\mathcal{P}_l), \phi, struct) \mid (0, \phi, struct) \in \mathcal{P}\})$.

Example 4. *Consider a transition for the Basic hash protocol in Example 3. The initial state of the protocol is $\mathcal{S} = (\mathsf{true}, \mathsf{true}, \mathsf{true}, \emptyset)$ and the initial set of possibilities is $\mathcal{P} = \{(0, \mathsf{true}, \emptyset)\}$. The first transition to be taken is Tag. The initial configuration for the Tag process w.r.t. $(\mathcal{S}, \mathcal{P})$ is $(\mathcal{S}, \{(\star\ T_1 \in \mathsf{Tags.snd}(\mathsf{pair}(N_1, h(\mathsf{sk}(T_1), N_1))).0, \mathsf{true}, \emptyset)\})$.*

From this initial configuration, we can get to a new state (or several states) by the following normalization and evaluation rules, basically working off the steps of the process $\mathcal{P}_l$. We first define these rules and then give a larger example in §A. Table I provides a summary of the following normalization and evaluation rules.

1) Normalization Rules: Recall that in a configuration, we have always one possibility being marked, which we denote by underlining it; in the following rules however, if no possibility is underlined, then the rule applies for all possibilities, no matter if marked or not.

a) Redundancy (NR 1.): We can always remove redundant possibilities when the intruder knows that a condition is inconsistent with β (this can never happen to the marked possibility, as the truth is always consistent with β):

$$\{(P, \phi, struct)\} \cup \mathcal{P} \implies \mathcal{P} \text{ if } \beta \models \neg\phi$$

b) Redundant entries in δ (NR 2.): An entry $\mathsf{cell}(s) := t$ if ϕ can be removed from δ if $\beta \models \neg\phi$.

c) Cell Reads (NR 3.): Let $C[\cdot]$ be the initial state of cell, and let the cell operations in the current state $\mathcal{S}$ be $\mathsf{cell}(s_1) := t_1$ if ϕ_1, …, $\mathsf{cell}(s_n) := t_n$ if ϕ_n. Then, every possibility that starts with the reading of cell is normalized via:

$$\begin{aligned}&\{(x := \mathsf{cell}(s).P_l, \phi, struct)\} \cup \mathcal{P} \implies \\ &\{(\mathsf{if}\ s = s_n \wedge \phi_n\ \mathsf{then\ let}\ x := t_n.P_l\ \mathsf{else} \\ &\quad \mathsf{if}\ s = s_{n-1} \wedge \phi_{n-1}\ \mathsf{then\ let}\ x := t_{n-1}.P_l\ \mathsf{else} \\ &\quad \ldots \\ &\quad \mathsf{if}\ s = s_1 \wedge \phi_1\ \mathsf{then\ let}\ x := t_1.P_l\ \mathsf{else} \\ &\quad \mathsf{let}\ x := C[s].P_l, \phi, struct)\} \cup \mathcal{P}\end{aligned}$$

The same rule holds if the possibility is marked (and then the transformed possibility is the marked one).

Example 5. *Consider a cell family r with initial state $C[\cdot] = \mathsf{init}(\cdot)$, and a state where δ has exactly one entry for memory cell r: $r(X) := h(\mathsf{init}(X))$ if true. Consider now the following possibilities: $\{(\mathsf{Key} := r(Y).P, \mathsf{true}, struct)\} \cup \mathcal{P}$. The normalization yields:*

$$\begin{aligned}&\{(\mathsf{if}\ Y \doteq X\ \mathsf{then\ let\ Key} := h(\mathsf{init}(Y)).P\ \mathsf{else} \\ &\quad \mathsf{let\ Key} := \mathsf{init}(Y).P, \mathsf{true}, struct)\} \cup \mathcal{P}\end{aligned}$$

Thus cell reads are reduced to conditionals at run time and conditionals we consider next.

d) Conditional (NR 4.): A conditional process is normalized via:

$$\begin{aligned}&\{(\mathsf{if}\ \psi\ \mathsf{then}\ \mathcal{P}_1\ \mathsf{else}\ \mathcal{P}_2, \phi, struct)\} \cup \mathcal{P} \implies \\ &\{(\mathcal{P}_1, \phi \wedge \psi, struct), (\mathcal{P}_2, \phi \wedge \neg\psi, struct)\} \cup \mathcal{P}\end{aligned}$$

If the process "if ψ then $\mathcal{P}_1$ else $\mathcal{P}_2$" is marked, then, by construction, $\beta \wedge \gamma \models \phi$. Recall that the interpretation of symbols is fixed, and that due to well-formedness, the truth γ determines one value for all variables in ψ. Thus, either $\beta \wedge \gamma \models \phi \wedge \psi$ or $\beta \wedge \gamma \models \phi \wedge \neg\psi$. Accordingly, exactly one of the alternatives gets marked.

Example 6. *The possibilities reached in the previous example, we can thus further normalize:*

$$\begin{aligned}&\{(\mathsf{if}\ Y \doteq X\ \mathsf{then\ let\ Key} := h(\mathsf{init}(Y)).P\ \mathsf{else} \\ &\ \mathsf{let\ Key} := \mathsf{init}(Y).P, \mathsf{true}, struct)\} \cup \mathcal{P} \implies \\ &\quad \{(\mathsf{let\ Key} := h(\mathsf{init}(X)).P, X \doteq Y, struct), \\ &\quad (\mathsf{let\ Key} := \mathsf{init}(Y).P, X \not\doteq Y, struct)\} \cup \mathcal{P}\end{aligned}$$

e) Cell write (NR 5.): A cell write process is normalized via:

$$\{(\mathsf{cell}(s) := t.P_r, \phi, struct)\} \cup \mathcal{P} \implies \{(P_r, \phi, struct)\} \cup \mathcal{P}$$

where δ is augmented with the entry $\mathsf{cell}(s) := t$ if ϕ. The order of these entries in δ depends on which normal-

izations are performed first, e.g., if we have $\{(\mathsf{cell}(s_1) := t_1.0, \phi_1, struct_1), (\mathsf{cell}(s_2) := t_2.0, \phi_2, struct_2)\}$, the resulting δ is either $\mathsf{cell}(s_1) := t_1$ if ϕ_1, $\mathsf{cell}(s_2) := t_2$ if ϕ_2 or $\mathsf{cell}(s_2) := t_2$ if ϕ_2, $\mathsf{cell}(s_1) := t_1$ if ϕ_1.

However, both orderings are in some sense equivalent, because ϕ_1 and ϕ_2 are mutually exclusive, so at most one of them can happen in any given model $\mathcal{I}$ of β. A similar argument holds for any critical pair of applicable normalization rules, and thus an arbitrary application strategy of the normalization rules may be fixed for the uniqueness of the definition.

Example 7. *Continuing the previous example, suppose* $P = r(Y) := h(\mathsf{Key}).P'$*; then, we have the possibilities:*

$$\{(r(Y) := h(h(\mathsf{init}(X))).P', X \doteq Y, struct), \\ (r(Y) := h(\mathsf{init}(Y)).P', X \not\doteq Y, struct)\} \cup \mathcal{P}$$

Thus normalization yields

$$\{(P', X \doteq Y, struct), \\ (P', X \not\doteq Y, struct)\} \cup \mathcal{P}$$

and δ *is augmented by entries (in any order):*

$$r(Y) := h(h(\mathsf{init}(X))).P' \text{ if } X \doteq Y \\ r(Y) := h(\mathsf{init}(Y)).P' \text{ if } X \not\doteq Y$$

f) Release (NR 6.): Given a process that wants to release some information ϕ_0, if the possibility is marked then we add it to α, if mode is $\star$ or to γ if mode is $\diamond$, else we ignore it:

$$\{\underline{(\text{mode } \alpha_0.P_r, \phi, struct)}\} \cup \mathcal{P} \implies \{\underline{(P_r, \phi, struct)}\} \cup \mathcal{P}$$

Recall that in process specifications, the formula ϕ_0 may contain subterms of the form $\mathcal{I}(t)$, e.g., $x = \mathcal{I}(x)$. If ϕ_0 is added to α, it must only contain symbols in Σ_0, otherwise we consider it as a specification error (i.e., privacy for this specification is undefined). Recall that γ gives every privacy variable a unique value that occurs in the current state. We write $\mathcal{I}$ to denote the corresponding substitution induced by γ.

Example 8. *Consider a state where* $\gamma \equiv X \doteq 0$*. Consider now the possibility* $\{(\star\ X \doteq \mathcal{I}(X).0, \phi, struct)\}$*. The normalization yields* $\{(0, \phi, struct)\}$ *and the* α *formula is now augmented with the conjunct* $X \doteq 0$ *since* $\mathcal{I}(X) = 0$*.*

2) Evaluation Rules: We call a set of possibilities *normalized* if normalization rules have been applied as far as possible. The first step of a normalized set of possibilities is either a non-deterministic choice, a send or a receive step, or they finished—since all other constructs are acted upon by the normalization rules. The following evaluation rules can produce multiple successor configurations (due to non-deterministic choice), and they can produce non-normalized possibilities. Before another of the evaluation rules can be taken, the possibilities have to be normalized again.

a) Non-deterministic choice (ER 1.): If the first step in the marked process is a non-deterministic choice, then, all processes must start with a non-deterministic choice of the same variable x from the same domain D, since we required that *varseq* is defined as in Def. 4 and the set of configurations is normalized. In this case, the evaluation is defined as a non-deterministic configuration transition for every $c \in D$:

$$((\alpha, \beta, \gamma, \delta), \{((\text{mode } x \in D.\mathcal{P}_1, \phi_1, struct_1), \ldots, \\ (\text{mode } x \in D.\mathcal{P}_n, \phi_n, struct_n))\}) \to \\ ((\alpha', \beta', \gamma', \delta), \{(\mathcal{P}_1, \phi_1, struct_1), \ldots, (\mathcal{P}_n, \phi_n, struct_n)\})$$

where: $\alpha' \equiv \alpha \wedge x \in D$ if mode is $\star$; $\alpha' \equiv \alpha$ if mode is $\diamond$; $\beta' \equiv \beta \wedge x \in D$; $\gamma' \equiv \gamma \wedge x \doteq c$.

Example 9. *We consider the following configuration:* $((\alpha, \beta, \gamma, \delta), \{(\star\ X \in \{0,1\}, \phi, struct)\})$*. There are two successor configurations that represent the different possible instantiations of the variable* X*:*

$$((\alpha, \beta, \gamma, \delta), \{(\star\ X \in \{0,1\}.0, \phi, struct)\}) \to \\ ((\alpha', \beta', \gamma', \delta), \{(0, \phi, struct)\})$$

where $\alpha' \equiv \alpha \wedge X \in \{0,1\}$*,* $\beta' \equiv \beta \wedge X \in \{0,1\}$ *and* $\gamma' \equiv \gamma \wedge X \doteq 0$*; The other state is identical except* $\gamma' \equiv \gamma \wedge X \doteq 1$*.*

b) Marked process receives (ER 2.): Also in this case, if one process starts with a receive, all the others start with a receive as well. Also here, we have several possible state transitions, since the intruder can freely choose a message to send to the processes. Let r be any recipe that the intruder can generate according to β, i.e., $\beta \models gen(r)$. For every such r, we have a configuration transition:

$$\{\underline{(\mathsf{rcv}(x).P_1, \phi_1, struct_1)}, \ldots, (\mathsf{rcv}(x).P_k, \phi_k, struct_k)\} \to \\ \{\underline{(P_1[x \mapsto struct_1\{\!|r|\!\}], \phi_1, struct_1)}, \ldots, \\ \underline{(P_k[x \mapsto struct_k\{\!|r|\!\}], \phi_k, struct_k)}\}$$

Note that our construction requires that in any $\mathsf{rcv}(x).P_k$, x is a variable that did not occur previously in the same process, i.e., we forbid $\mathsf{rcv}(x).\mathsf{rcv}(x).P_k$, as explained in Definition 4.

Example 10. *Consider the set of possibilities* $\{(\mathsf{rcv}(Z).P_a, \phi, struct_a), (\mathsf{rcv}(Z).P_b, \phi, struct_b)\}$*. Suppose the intruder chooses to send as input the recipe* $h(l)$ *for some label* l *in* $struct_a$ *(by construction* $struct_a$ *and* $struct_b$ *must have the same domain). Each process receives the message that results from that recipe in the respective possibility:*

$$\{(P_a[Z \mapsto struct_a\{\!|h(l)|\!\}], \phi, struct_a), \\ (P_b[Z \mapsto struct_b\{\!|h(l)|\!\}], \phi, struct_b)\}$$

c) Marked process sends (ER 3.): If the marked process sends a message next, this is observable, and all processes that do not send are ruled out. Thus, we have the rule

$$\{(\mathsf{snd}(m_1).P_1, \phi_1, struct_1), \ldots, \\ (\mathsf{snd}(m_k).P_k, \phi_k, struct_k)\} \cup \mathcal{P} \to \\ \{\underline{(P_1, \phi_1, struct_1 \cup \{\!|l \mapsto m_1|\!\})}, \ldots, \\ \underline{(P_k, \phi_k, struct_k \cup \{\!|l \mapsto m_k|\!\})}\}$$

where l is a fresh label and $\mathcal{P}$ is a set of possibilities that are finished (i.e., all the processes are the 0 process), and we augment β by:

$$\phi_1 \vee \ldots \vee \phi_k \wedge concr[l] = \gamma(m_1) \wedge \\ \exists i \in \{1, \ldots, k\}.\ \textstyle\bigvee_{j=1}^{k} i = j \wedge struct[l] = m_j \wedge \phi_j$$

This is because the intruder can now rule out all possibilities in $\mathcal{P}$ and their conditions (so one of the ϕ_i in the remaining processes must be true). Moreover, the intruder knows a priori only that the message they receive, concretely $\gamma(m_1)$, is one of the m_i and this is the case iff ϕ_i holds.

Example 11. *Consider the following process and its evaluation, and suppose that γ contains $X \doteq 1$:*

$$\{(\mathsf{send}(h(\mathsf{init}(X))).0, X \doteq Y, struct_a),\\ \underline{(\mathsf{send}(\mathsf{init}(Y)).0, X \not\doteq Y, struct_b)}\} \to\\ \{\underline{(0, X \doteq Y, struct_a \cup \{\!|l \mapsto h(\mathsf{init}(X))|\!\})},\\ \underline{(0, X \not\doteq Y, struct_b \cup \{\!|l \mapsto \mathsf{init}(Y)|\!\})}\}$$

β is augmented by $concr[l] = \mathsf{init}(1) \wedge \exists i \in \{1,2\}.\big((i = 1 \wedge struct[l] = h(\mathsf{init}(X))) \vee (i = 2 \wedge struct[l] = \mathsf{init}(Y))\big)$, representing that the concrete message the intruder receives follows the underlined possibility (instantiating X with its true value) and that the intruder a priori does not know which of the two possibilities the true structure of the message is.

d) Marked process has terminated (ER 4.): If the marked process has terminated, then the others have either also terminated or start with a send step (since other cases are already done). If all processes are terminated, we are done, otherwise the intruder can rule out the processes that are not yet done:

$$\{\underline{(0, \phi_1, struct_1)}, \ldots, (0, \phi_k, struct_k)\} \cup \mathcal{P} \to\\ \{\underline{(0, \phi_1, struct_1)}, \ldots, (0, \phi_k, struct_k)\}$$

where $\mathcal{P}$ is a set of possibilities that start with a send, and we augment β by $\phi_1 \vee \ldots \vee \phi_k$. In any case, no further normalization and evaluation rules are applicable, and thus have reached a state.

After defining transition systems, let us define *dynamic (α, β)-privacy*.

Definition 8 (Dynamic (α, β)-privacy)**.** *Given a configuration $(\mathcal{S}, \mathcal{P})$, a transaction process $\mathcal{P}_l$, and a substitution θ as in Def 7, the* successor states *are defined as all states reachable from the initial configuration of $\mathcal{P}_l$ using the normalization and evaluation rules. The set of* reachable states *of a protocol description is the least reflexive transitive closure of this successor relation w.r.t. a given initial state of the specification (the possibilities being initialized with $(0, \mathsf{true}, \emptyset)$).*

We say that a transition system satisfies dynamic (α, β)-privacy *iff static (α, β)-privacy holds for every reachable state.*

IV. DP-3T

As a concrete and topical example, we consider the decentralized, privacy-preserving proximity tracing system DP-3T [14], which has been developed to help slow the spread of the SARS-CoV-2 virus by identifying people who have been in contact with an infected person. The DP-3T system aims to minimize privacy and security risks for individuals and communities, and to guarantee the highest level of data protection.

A. Modeling

For every agent and for every day, we have a day key, and the day is further separated into periods (e.g., of 15 minutes), and for each period, each agent generates a new ephemeral identity. In order to avoid any complications with infinite numbers of models, we consider finite (but arbitrarily large) sets of agents, day keys, and ephemeral IDs. Moreover, we use these sets as *sorts*, so that we can define interpreted functions between these sorts without inducing infinitely many models for these functions. We use the following sorts:

- Agent is the sort of all participating agents,
- $Day = \{0, \ldots, \mathsf{D}-1\}$ identifies days,
- $Period = \{0, \ldots, \mathsf{P}-1\}$ identifies a particular period of a day, i.e., a day is partitioned into P periods (e.g., of 15 minutes),
- SK is the sort of daily identities that contains at least $(D \times |\mathsf{Agent}| + 1)$ elements, and
- $EphID$ is the sort of ephemeral identities (changing, e.g., every 15 minutes). This set contains at least $(|\mathsf{Agent}| \times D \times P)$ elements.

Let all elements of these sorts but SK be part of Σ_0, so that α formulae can talk about agents, days, and ephemeral identities. On these sorts, we define the following functions and relations:

- $\mathsf{sk}_0[\cdot]\colon \mathsf{Agent} \to SK$ maps every agent to their first-day key. We assume that this key is distinct for every agent, i.e., $\mathsf{sk}_0[a] \neq \mathsf{sk}_0[b]$ for any $a \neq b$.
- $h[\cdot]\colon SK \to SK$ is a hash function that maps every daily identity to the next day. We assume that for every $a\colon \mathsf{Agent}$, we have a seed value $\mathsf{sk}_0[a] \in SK$ such that $h^i[\mathsf{sk}_0[a]] \neq h^j[\mathsf{sk}_0[b]]$ for any $a, b \in \mathsf{Agent}$, $i, j \in Day$ with $(a, i) \neq (b, j)$: every daily identity of an agent is unique.
- $prg[\cdot,\cdot]\colon SK \times Period \to EphID$ models a pseudo-random number generator to generate the ephemeral identities. We assume prg is injective on the domain $SK \times Period$, so that there is also no collision between the ephemeral identities of any agents (with respect to any timepoints).
- $pwnr[\cdot]\colon EphID \to \mathsf{Agent}$ relates an ephemeral ID to its actual owner in our model, i.e., for $e = prg[h^i[\mathsf{sk}(a)], j]$, we have $pwnr[e] = a$.
- $dayof[\cdot]\colon EphID \to Day$ tells the day an ephemeral ID is issued.
- $sick \subseteq EphID \times Day$ is a relation where $sick(e, d)$ means that the agent identified by e has declared sick on day d. In contrast, $dayof[e]$ is the day when e was used.

We fix the interpretation of these functions and relations so that the described constraints are satisfied: we pick for each agent and each day a unique element from SK, and interpret $\mathsf{sk}_0[a]$ as the key of a for day 0, and the $h[\cdot]$ maps that key to the day 1 key of a. Observe there is at least one more element in SK, which is where all remaining $h[\cdot]$ map, so we do not have any collisions except outside the area that we are using. Given the size of $EphID$ we can fix an injective interpretation for prg, and can then set the interpretation of $pwnr$, $dayof$ and $sick$ as expected.

The functions h and prg are cryptographic functions, and sk_0 is a cryptographic setup. We regard them as techni-

cal/implementation related, so they are only part of $\Sigma \setminus \Sigma_0$ and cannot be used in α. We have made several assumptions about absence of collisions in these functions: these assumptions are part of β in the initial state. The functions *pwnr* and *dayof* and the relation *sick* are part of the high-level modeling, and thus part of Σ_0.

We use the following memory cells with their initial values:

- $\mathsf{sk}_l(A\colon \mathsf{Agent}) := \mathsf{sk}_0[A]$ is whatever is the opposite of a look-ahead: it represents the day ID of agent A of l days ago, where l is the period how far back we want to report the sickness after a positive test (e.g. five days),
- $\mathsf{sk}(A\colon \mathsf{Agent}) := h^l[\mathsf{sk}_0[A]]$. The current day ID of A is l hashes ahead of sk_0. Thus, within the first l days of the app, we have some "virtual" past days where we can report sickness—this is to keep the model simple,
- $today() := l$ is the current day counter (it is the same for all agents),
- $period() := 0$, where 0 identifies the first period of a day,
- $ephid(A\colon \mathsf{Agent}) := prg[\mathsf{sk}(A), period()]$ is the current ephemeral ID, and
- $isSick(A\colon \mathsf{Agent}) := \mathsf{false}$ is a flag to indicate that the agent has reported sick and should no longer use the app and should quarantine.

We consider the agent transactions in Fig. 1. The transaction *New Day or Period* advances a global clock, and when a day is finished, automatically triggers the generation of new day keys for each agent. This ignores any privacy problems that could arise from de-synchronized clocks and the like. The *Agent Advertise* transaction models that an agent can at any time communicate its current ephemeral identity e and that the intruder never learns more than the owner of e is some agent $x \in \mathsf{Agent}$. Our model ignores the details of how two agents' phones actually exchange IDs, which can cause also several problems [23]. The *Agent Sick* Transaction models that an agent declares sick and publishes the day keys in their sickness period (for simplicity, we publish only the oldest, the others can be generated by everybody themselves). We specify that the intruder should now only learn that all ephemeral IDs belong to an agent that has just declared sick. The model actually omits the details of how this sick report is communicated to a central server (who must also somehow check with health authorities whether the agent is indeed sick), which again is not trivial to get right [23]. Our model focuses on the core privacy question that arises, even if all exchange protocols work perfectly.

B. Privacy violated

Suppose that we have two advertisements by the same agent a in the first two periods of the first day (numbered l), i.e., let $\mathsf{sk}_l = h^l[\mathsf{sk}_0[a]]$ be the day key, and $e_0 = prg[\mathsf{sk}_l, 0]$ and $e_1 = prg[\mathsf{sk}_l, 1]$ be the released ephemeral IDs. On the same day, a releases a sick note $\mathsf{sk}_0[a]$ that gives rise to further ephemeral IDs $e_2, \ldots, e_n$. Then, α in the reached state is:

$$\begin{aligned}\alpha \equiv\ & x_1 \in \mathsf{Agent} \wedge pwnr[e_0] \doteq x_1 \wedge dayof[e_0] \doteq l \wedge \\ & x_2 \in \mathsf{Agent} \wedge pwnr[e_1] \doteq x_2 \wedge dayof[e_1] \doteq l \wedge \\ & x_3 \in \mathsf{Agent} \wedge sick(e_0, l) \wedge \ldots \wedge sick(e_n, l)\end{aligned}$$

where $e_0, \ldots, e_n$ are all ephemeral keys of a released in the sick report. The following can be derived from β, for some labels m_1, m_2 and m_3 where the sent messages are stored:

$$\begin{aligned} concr[m_1] &= e_0 & struct[m_1] &= prg[h^l[\mathsf{sk}_0[x_1]], 0] \\ concr[m_2] &= e_1 & struct[m_2] &= prg[h^l[\mathsf{sk}_0[x_2]], 1] \\ concr[m_3] &= \mathsf{sk}_l & struct[m_3] &= h^l[\mathsf{sk}_0[x_3]] \end{aligned}$$

Intruder deductions:

$$\begin{aligned} &concr[prg[m_3, 0]] = prg[h^l[\mathsf{sk}[a]], 0] = e_0 = concr[m_1] \\ &concr[prg[m_3, 1]] = prg[h^l[\mathsf{sk}[a]], 1] = e_1 = concr[m_2] \end{aligned}$$

Using $\phi_\sim$:

$$\begin{aligned} &struct[prg[m_3, 0]] = struct[m_1] \\ &struct[prg[m_3, 1]] = struct[m_2] \\ &prg[h^1[\mathsf{sk}_1[x_3]], 0] = prg[h^l[\mathsf{sk}_0[x_1]], 0] \\ &prg[h^l[\mathsf{sk}_0[x_3]], 1] = prg[h^l[\mathsf{sk}_0[x_2]], 1] \end{aligned}$$

By the properties of prg, h and $\mathsf{sk}_0 : x_3 = x_2 \wedge x_3 = x_1$ and thus $x_1 = x_2$.

This last statement is however not compatible with all models of α, so dynamic (α, β)-privacy is indeed violated. Note that we do not find out that $x_1 \doteq a$, but we have linkability of pseudonyms of sick persons.

C. The Actual Privacy Guarantee

The protocol releases more information than we have specified so far in α. This corresponds to the privacy problem that the intruder gets to know that all the ephemeral identities of a day are related to the same agent. This could be relevant if, e.g., the intruder surveys in several places for ephemeral identities and can then build partial profiles of users who declared sick.

We at least need to add the following information: in the sick release by the information there is one particular agent who is the owner of all released sick-predicates, i.e., in the Agent Sick transaction we have the α release. This provides the link between all ephemeral IDs released by an agent, because the owner is the same agent x (who of course still remains anonymous, hence the variable).

As a consequence, this additional information (i.e., that $x_1 \doteq x_2 \doteq x_3$) no longer counts as an attack, because we explicitly declare that we want to release this information, which we can formalize by adding it to α. This highlights how in (α, β)-privacy one can—as a conscious choice of the modeler—move to a weaker privacy goal (by allowing the intruder to obtain more information), when the protocol is not as strong as initially expected.

However, this extended α *still* does not cover all the information we release. For instance, if the two agents $x_1 \doteq a$ and $x_2 \doteq b$ have released ephemeral IDs e_a and e_b, respectively, and a has declared sick, then we can still observe that $x_1 \neq\!\!\!\!\!\doteq x_2$ because e_b does not belong to any of the keys that have been released with a sick note. Similarly, we can distinguish agents that have declared sick; for instance, if both a and b have declared sick, then we can also derive $x_1 \neq\!\!\!\!\!\doteq x_2$, because we have distinct day keys and moreover, when two day keys belong to the same agent, then they are related by the hash function, i.e., $\mathsf{sk}_1 = h^k[\mathsf{sk}_2]$ or vice-versa.

New Day or Period

if $(period() < \mathsf{P} - 1)$ then
  $period() \coloneqq period() + 1$
else $period() \coloneqq 0$
  if $(today() < \mathsf{D} - 1)$ then
    $today() \coloneqq today() + 1$
    for $x\colon \mathsf{Agent}$
      $\mathsf{sk}(x) \coloneqq h(\mathsf{sk}(x))$
      $\mathsf{sk}_l(x) \coloneqq h(\mathsf{sk}_l(x))$

Agent Advertise

$\star\ x \in \mathsf{Agent}$
if $\neg isSick(x)$ then
  let $z = prg[\mathsf{sk}(x), period()]$
  $\star\ pwnr[\mathcal{I}(z)] = x \wedge dayof[\mathcal{I}(z)] = today()$
  $\mathsf{snd}(z)$

Agent Sick

$\star\ x \in \mathsf{Agent}$
if $\neg isSick(x)$ then
  $isSick(x) \coloneqq \mathsf{true}$
  let $y = \mathsf{sk}_l(x)$
  for $i \in Period\ \wedge\ j \in \{0, \ldots, l\}$
    $\star\ sick(\mathcal{I}(prg[h^j[y], i]), \mathcal{I}(today()))$
  $\mathsf{snd}(y)$

Fig. 1. A model of DP-3T (with insufficient α)

$$\begin{aligned}
&\forall E, F \in EphID, C, D \in Day\colon\ sick(E, C) \wedge sick(F, D) \wedge \\
&\quad pwnr[E] \doteq pwnr[F] \implies C \doteq D \\
&\wedge\ \forall E, F \in EphID, D \in Day\colon\ sick(E, D) \wedge \\
&\quad pwnr[E] \doteq pwnr[F] \implies dayof[F] \leq D \\
&\wedge\ \forall E, F \in EphID, D \in Day\colon\ pwnr[E] \doteq pwnr[F] \wedge \\
&\quad dayof[E] \doteq dayof[F] \wedge sick(E, D) \implies sick(F, D) \\
&\wedge\ \forall E_0, E_1, \ldots, E_{\mathsf{P}} \in EphID\colon\ \textstyle\bigwedge_{i,j \in \{0,\ldots,\mathsf{P}\}, i \neq j} E_i \neq E_j \\
&\quad pwnr[E_1] \doteq \ldots \doteq pwnr[E_{\mathsf{P}}] \wedge \\
&\quad dayof[E_1] \doteq \ldots \doteq dayof[E_{\mathsf{P}}] \\
&\quad \implies pwnr[E_0] \neq pwnr[E_1] \vee dayof[E_0] \neq dayof[E_1]
\end{aligned}$$

Fig. 2. Axioms for DP-3T

So, actually, what we really give out here is much more, and it is not easy to keep track of it without basically copying into α most of what is going on in β, and thus basically making the implementation be also the specification. However, as most logicians will agree, there is almost always a declarative way to describe things. In this case, we can actually formalize a few relevant properties of the implementation as axioms on the Σ_0 level, without talking about the day keys SK or how they are generated and how the ephemeral IDs are generated. These axioms are given in Figure 2, and we obtained them from failed attempts of proving dynamic (α, β)-privacy, adding missing aspects until we could prove it. Here is what these axioms respectively express:

- an agent declares sick only once,
- after declaring sick, the agent does not use the app anymore. In fact, they could, if we had a reset operation that installs a new initial key, but we refrained from further complicating the model,
- when an agent reports sick for a particular day, this entails all ephemeral identities for that day, and
- finally, let $\mathsf{P} = |Period|$ denote the number of periods in a day; then there cannot be more P ephemeral IDs that belong to the same agent on the same day.

We shall thus, from now on, consider the axioms in Figure 2 as part of α in our initial state. Now, it may not be entirely intuitive anymore what this actually implies. So, let us look at the general form that α has after a number of transitions, and how to compute the models (satisfying interpretations) of α. In general, in any reachable state the formula α consists of conjuncts of the following forms:

- from Agent Advertise: $x \in \mathsf{Agent} \wedge pwnr[e] \doteq x \wedge dayof[e] = d$, where $e \in EphID$, $d \in Day$, and x is a variable that occurs nowhere else in α, and
- from Agent Sick: $\bigwedge_{e \in E} pwnr[e] \doteq x \wedge sick(e, d_r)$, where E is a set of ephemeral IDs that are released on reporting day d_r. Among all agent sick reports, the set E is pairwise disjoint. Moreover, the variable x occurs nowhere else in α. Finally, the size of E is $|Period| \times l$, i.e., for every of the l days and for every time period of a day, we identify exactly one ephemeral ID as sick.

Lemma 1. *Every model $\mathcal{I}$ of α can be computed by the following* non-deterministic *algorithm:*

1) *Consider every conjunct that arose from Agent Sick and consider the variable x of that conjunct.*
   a) *For every such x,* choose *a unique $a \in \mathsf{Agent}$ and set $\mathcal{I}(x) = a$. (Unique here means: two different Agent-sick conjuncts with variables x and x' must be interpreted as different agents $\mathcal{I}(x) \neq \mathcal{I}(x')$).*
   b) *For every e that occurs in this conjunct, we have $\mathcal{I}(pwnr[e]) = a$.*
2) *Consider every conjunct that arose from Agent Advertise and let x be the variable occurring in there and e be the ephemeral ID in there.*
   a) *If $\mathcal{I}(pwnr[e]) = a$ has been determined, then $\mathcal{I}(x) = a$.*
   b) *If $\mathcal{I}(pwnr[e])$ has not yet been determined, then let d be the day that is has been declared. Let Agent_s be the set of agents that have declared sick on day d or before, i.e., $\mathcal{I}(x')$ for every x' s.t. α contains $sick(e, d') \wedge pwnr[e] = x' \wedge dayof[e] = d_0$ and $d_0 \leq d$. Further, let Agent_e denote all the agents a for which $\mathcal{I}(pwnr[e]) = a$, and $\mathcal{I}(dayof[e]) = d$ for P different ephemeral IDs e. Then,* choose *$a \in \mathsf{Agent} \setminus \mathsf{Agent}_s \setminus \mathsf{Agent}_e$ arbitrarily and set $\mathcal{I}(x) = a$ and $\mathcal{I}(pwnr[e]) = a$.*
3) *All remaining aspects of $\mathcal{I}$ are actually irrelevant (i.e., $\mathcal{I}(pwnr[e])$ for e that did not occur in the formula).*

In a nutshell: α does not reveal any agent names, but allows one to distinguish all sick agents from each other and from the non-sick, and it allows one to link all ephemeral IDs of every sick agent from the first day of sickness on.

Proof. Soundness (i.e., the algorithm produces only models of α): the algorithm respects obviously every conjunct of α produced during transactions, and for the axioms the distinct choice of sick-reported agents is actually sufficient.

Completeness (i.e., every model of α is produced by the algorithm): we have first to show that α enforces $\mathcal{I}(x_i) \neq \mathcal{I}(x_j)$ for every pair of variables x_i and x_j that occur in distinct sickness reports. Suppose this were not true, i.e., we have a model $\mathcal{I}$ of α such that $\mathcal{I}(x_i) = \mathcal{I}(x_j)$ for the variables x_i and x_j from distinct agent sickness reports. From the construction, we know each sick report contains exactly $\mathsf{P} \cdot l$ ephemeral IDs (l days reporting, and P periods per day), and the ephemeral IDs from distinct sick reports are disjoint. Moreover, each sick report has a reporting day, say d_i and d_j. Let thus e_i and e_j be ephemeral IDs from the two sick reports, then $\mathcal{I} \models pwnr[e_i] \doteq x_i \doteq x_j \doteq pwnr[e_j]$ and therefore the axioms entail $d_i = d_j$ (same day of reporting). Thus, α contains for each sick report P ephemeral IDs for l days up to reporting day $d_i = d_j$. That is however impossible by the axiom that not more than P different ephemeral IDs can have the same day and the same owner (while we have $2 \cdot \mathsf{P}$ according to assumption). Thus, $\mathcal{I}(x_i) \doteq \mathcal{I}(x_j)$ is absurd.

That all distinct sickness reports must be interpreted as being done by different agents shows the completeness of the choice in step 1a. Steps 1b and 2a are directly enforced by α. For step 2b, we have an ephemeral ID e for an agent x, such that e is not contained in any sick-report. By $dayof[e] = d$ we can check all sick reports that have been done on day d or before, and which agents we have reported there according to a given model $\mathcal{I}$, which the algorithm calls the set Agent_s. Suppose $\mathcal{I}(x) \in \mathsf{Agent}_s$, i.e., there is a sick report for an agent x' and $\mathcal{I}(x') = \mathcal{I}(x)$ that has at least one ephemeral id e' that is included in the sick report for day $d' \leq d$. If $d = d'$, this contradicts the axiom that an agent releases all their ephemeral IDs for a given sick day, because we were considering an e that was not reported sick. If $d' < d$, this contradicts the axiom that the agent stops using the app after the sick report, i.e., $dayof[e]$ must be before the sick report. Finally, we have to show that also $\mathcal{I}(x) \in \mathsf{Agent}_e$ is not possible, because Agent_e contains all agents for which we have interpreted already P different ephemeral IDs for this day. This directly follows from the axiom that there are at most P different ephemeral IDs for the same agent on the same day. This shows that the choice in step 2b of an agent outside Agent_s and Agent_e is complete.

Hence, the algorithm allows all choices that are not excluded by α itself, and is thus complete. □

This characterization of the models of α of any reachable state allows us to prove dynamic (α, β)-privacy as follows.

Theorem 1. *DP-3T with the extended α specification (including the axioms in Figure 2) given in this section satisfies dynamic (α, β)-privacy.*

Proof. We have to show that in every reachable state, any model $\mathcal{I}_0$ of α can be extended to a model $\mathcal{I}$ of β. Note that β must have a model $\mathcal{I}_r$ that corresponds to what really happened (and it is also a model of α). The idea is that we incrementally construct $\mathcal{I}$ close to $\mathcal{I}_r$.

First, we choose a key from SK for every agent a and every day d that occur in β; let us call it $\mathsf{sk}_{a,d}$. The principle here is: if, according to $\mathcal{I}$, agent a declares sick at some point, then β will contain the publication of the corresponding day keys of some agent x, where $\mathcal{I}(x) = a$. So, we have to set $\mathsf{sk}_{a,d}$ for those days d and a accordingly. All remaining keys can be set to arbitrary distinct values from SK, disjoint from those occurring in β. $\mathsf{sk}_{a,d} = \mathsf{sk}_{b,c}$ implies $a = b$ and $c = d$ by construction now, so set $\mathcal{I}(\mathsf{sk}_0[a]) = \mathsf{sk}_{a,0}$, and $\mathcal{I}(h[\mathsf{sk}_{a,d}]) = \mathsf{sk}_{a,d+1}$ for any agent a and day d occurring in β.

For prg, we can already pick some values in a convenient way: for those sk that are part of a sick report (i.e., not arbitrarily chosen from SK in the previous step), we can choose the ephemeral IDs derived from them to be identical to those in $\mathcal{I}_r$, i.e., set $\mathcal{I}(prg[\mathsf{sk}, i]) = \mathcal{I}_r(prg[\mathsf{sk}, i])$ for every period $i \in Period$ and every day key sk that is covered by a sickness report. The remaining ephemeral IDs (that did not occur in sickness reports) will be chosen "on the fly" now. It is yet to be proved that this is consistent with the rest of β.

For the initial state, we have thus an "intruder interpretation", i.e., what the initial value of the memory cells $\mathsf{sk}_l(a)$ and $\mathsf{sk}(a)$ of every agent a is, namely $\mathcal{I}(\mathsf{sk}_0[a])$ and $\mathcal{I}(h^l[\mathsf{sk}_0[a]])$, respectively (while the real initial values are $\mathcal{I}_r(\mathsf{sk}_0[a])$ and $\mathcal{I}_r(h^l[\mathsf{sk}_0[a]])$). The intruder cannot see all the concrete values sk that occur here: the intruder can only see those values that have been explicitly released and apply the hash function further to them. Let us speak in the following of the *virtual state* of the memory cells, i.e., what value they would have (after a given sequence of transaction) if $\mathcal{I}$ were the reality.

The next day and the next period transactions just change the state; the virtual state is changed in a way that is completely determined by what we have determined in $\mathcal{I}$ so far.

For an agent advertisement transaction, let x be the variable for the agent in the transaction and $\mathcal{I}(x) = a$ the concrete agent according to $\mathcal{I}$ and e the ephemeral ID advertised. Let further sk, i, and d be the current values of $\mathsf{sk}(a)$, $period()$ and $today()$ in the virtual state. We distinguish two cases: first, if sk is a day key published in a sick report later, then we have already determined $\mathcal{I}(prg[\mathsf{sk}, i]) = \mathcal{I}_r(prg[\mathsf{sk}, i])$ previously, and $\mathcal{I}_r(prg[\mathsf{sk}, i]) = e$ because this is indeed the advertisement of the agent $\mathcal{I}_r(x)$ (which may have a name different from $\mathcal{I}(x)$) at this day and time period and sk is indeed the current day key this agent. Otherwise, if sk is not reported sick later, then $\mathcal{I}(prg[\mathsf{sk}, i])$ is not yet determined, unless we run the same advertisement a second time for the same agent on the same day and time period, and so it is already set to e, and we can set it to e. This is possible since in every other reached virtual state, sk and i are necessarily different, so $prg[\mathsf{sk}, i]$ has not yet been assigned a different interpretation yet. The formula β now contains (for an appropriate label m): $concr[m] = e \wedge struct[m] = prg[h^d[\mathsf{sk}_0[x]], i]$. This is because d and i in the virtual state are equal to the value in reality. Under $\mathcal{I}$, the struct term thus also equals e. We show below also for the other transitions that on every introduced label m it holds that $\mathcal{I} \models concr[m] = struct[m]$, and thus $concr$ and $struct$ will be trivially in static equivalence under $\mathcal{I}$.

For a sick report, let x be the variable for the agent in the transition and $\mathcal{I}(x) = a$ the concrete agent according to $\mathcal{I}$, and

let sk_l, i, and d be the current values of $\mathsf{sk}_l(a)$, $period()$, and $today()$ in the current virtual state. The formula β now contains $concr[m] = \mathsf{sk}_l$ and $struct[m] = h^{d-l}[\mathsf{sk}_0[x]]$. Observe also here that we have $\mathcal{I} \models concr[m] = struct[m]$ because $\mathsf{sk}_l(x)$ is x's key from l days ago. □

V. Comparison with Trace Equivalence

The gold standard for privacy in security protocols are the notions of *observational equivalence* and *trace equivalence* (see, e.g., [12] for a survey). Roughly, a pair of processes is trace equivalent if all transitions of one process can be simulated by the other. This entails substantial difficulties for automated verification [8], especially when systems have a long-term mutable state [3], but still privacy notions are typically formulated as such an equivalence between two alternative worlds, rather than a reachability problem. Interesting in this context is the notion of *diff-equivalence* [6] that is implemented in the most popular verification tools ProVerif [5] and Tamarin [22]: here the processes are parameterized over a binary choice in terms, and one proves the equivalence between the two processes that result from taking either all the "left" or all the "right" choices. The main requirement is now that during execution all if conditions are either both true or both false for the two variants. Thus, the two processes are basically in lockstep and we have also practically a reachability problem. While this is helpful for automation, it restricts the set of protocols that can be verified (without false positives); for instance, [4] discusses why the Basic Hash protocol and OSK could not be reasonably modeled in ProVerif and Tamarin directly, and [12] gives the similar BAC protocol as an example that cannot be handled with diff-equivalence.

In contrast, (α, β)-privacy gives us a reachability problem without such a restriction. In particular, the different possibilities $struct_i$ that we are maintaining in each state represent the different ways past conditions could have turned out and that the intruder cannot rule out. In fact, all mentioned examples can be directly expressed as reachability problems in (α, β)-privacy. In terms of expressive power, (α, β)-privacy thus seems close to the unrestricted trace equivalence and, while there are some substantial differences, we give some formal arguments for that in the following.

In addition to (α, β)-privacy's advantages of a declarative modeling, the simplicity of a reachability problem is also beneficial to automation. A first step towards that is found in [15], which solves the message analysis problem of static (α, β)-privacy defined in [20] that has only one $struct$. We are currently extending this method for the case of several $struct_i$ as is needed for the dynamic (α, β)-privacy defined in this paper. To handle the interaction with the intruder that arises from the rcv command, we are also working on a constraint-based approach to obtain a decision procedure for a bounded number of sessions. Note that related tools such as DEEPSEC [9] are also limited to a bounded number of sessions but are implementing a decision procedure for full trace equivalence. This leaves open the question whether all really expressive notions of privacy require a limit to a bounded number of sessions, or whether (despite undecidability) there can be algorithms for handling the unbounded case reasonably well in practice. We believe (α, β)-privacy may be a way, since it provides a reachability problem without requiring any restrictions such as those inherent in diff-equivalence.

A. Visibility of Transactions

It is inherent in the semantics of (α, β)-privacy that the intruder knows which transaction is currently being executed; but the intruder does *not* know which of the if-then-else branches is taken, unless this can be inferred from the communication behavior of the transaction. In contrast, most trace-based approaches are formulated in a variant of the Applied-π calculus and do not have a notion of transaction; the intruder view is thus limited to the communication behavior.

If desired, it is easy to express the same limited intruder view in (α, β)-privacy transactions;[3] given a specification of transactions $T_1, \ldots, T_n$, one can transform them into a single transaction T as follows (where z is a variable that does not occur in any of the T_i):

$$
\begin{array}{l}
\diamond\ z \in \{1, \ldots, n\}. \\
\mathsf{if}\ (z \doteq 1)\ \mathsf{then}\ T_1. \\
\mathsf{else\ if}\ (z \doteq 2)\ \mathsf{then}\ T_2. \\
\ldots \\
\mathsf{else\ if}\ (z \doteq n)\ \mathsf{then}\ T_n
\end{array}
$$

This transaction allows all the same behaviors as the T_is together, except that the intruder does not see a priori which of the T_is is taken. Depending on the output messages of the T_is, the intruder may anyway find out which T_i it is (or just narrow it down to a few candidates), but that in itself is not a violation of privacy since the non-deterministic choice of z was not released in α.

In our opinion, it is better to let the intruder know the transaction by default, and have the modeler explicitly specify otherwise (with the above construction), when the protocol privacy indeed relies on this. This makes it less likely that such a reliance is overlooked upon implementation. For the rest of this discussion, we will speak of transactions $T_1, \ldots, T_n$, but allowing for the case that $n = 1$ with the above construction.

B. Relations between messages sent and received

Another subtle difference between the modeling in (α, β)-privacy and in trace equivalence approaches concerns what relationship the intruder can see between messages sent and received by a single entity/process, which is very relevant for linkability goals as we have considered in previous examples. In trace equivalence approaches, the intruder cannot a priori see any relation between incoming and outgoing messages. Consider, for instance, the following two processes running in parallel: $P_1 = \mathsf{rcv}(X).\nu N.\mathsf{snd}(h(X, N))$ and $P_2 = \mathsf{new}\ N.\mathsf{snd}(N)$.

Suppose the intruder sends a message m_1 and then observes a message m_2. Then m_2 may either be a reply to m_1 from P_1, or the message from P_2. Of course, if P_1 is modeling

[3] It is similarly possible to equip a process calculus specification with additional messages that tell the intruder a particular point has been reached.

an entity that directly gives a reply to an input, in particular without any mechanism to break the timely relation between input and output (like batching of answers or dummy traffic as in mix-networks), then it is, in our opinion, just reasonable that the intruder can tell which process has sent it. We thus chose to introduce in (α, β)-privacy the concept of transaction processes that form an "atomic unit". Thus, the intruder can relate all messages sent and received by one transaction.

If one wants to hide this relationship from the intruder in (α, β)-privacy, one can break a transaction into smaller ones. For instance, P_1 can be split into $P_{1,a} = \mathsf{rcv}(X).\mathsf{cell} := X$ and $P_{1,b} = X := \mathsf{cell}(s).\nu N.\mathsf{send}(h(X, N))$, two transactions between which any number of other transactions can happen. Thus, the intruder a priori cannot relate inputs and outputs. A modeler should only do this if one is certain that the relationship is not visible to the intruder. Note that also in trace equivalence approaches one often models the observable relationship between messages (e.g., by generating a new public channel and sending all relatable messages over that channel).

C. Equivalence

We consider now any specification of a protocol that can be expressed as transactions meeting two restrictions as explained below. We show that for such a specification all privacy properties that can be expressed with trace equivalence can also be expressed with (α, β)-privacy. The reader should bear in mind that trace equivalence and (α, β)-privacy are two quite different "games", so bridging between them often leads to constructions, and requires restrictions, that are somewhat artificial, but that at least give an idea of how the two approaches relate.

We consider two restrictions (R1) and (R2) that do not seem utterly necessary, but greatly simplify the exposition. (R1): for this discussion, we consider (α, β)-privacy without interpreted functions except $concr$ and $struct$ and without relation symbols except gen. Hence, there are only the following "sources" of non-determinism:

- variables that are introduced as $\star\ x \in D$; let us call such an x an *α-variable* (because it is part of α),
- variables that are introduced as $\diamond\ y \in D$; let us call such a y a *β-variable* (because it is *not* part of α),
- the non-determinism of the transition relation itself, i.e., in a sequence of steps, which transaction is performed next, and
- for a transaction that receives a message, which of all available messages is received.

(R1) is helpful for the following discussion: we forbid the complications that arise from interpreted functions and relation symbols (cf. discussion after Def 4). While many protocols like our Basic Hash and OSK examples satisfy this condition, the DP-3T example does not. It seems that in many cases one could find an alternative formalization that uses memory cells instead of interpreted functions, but we have found no precise characterization of the limits of such encodings.

(R2): we restrict transactions to having exactly one input and one output (on every path through its if-the-else conditions). This simplifies the problem as for a trace of k transactions we have now exactly k inputs and k outputs. Note that none of our major examples satisfies (R2) but they can all be transformed into equivalent specifications (in the sense that they enjoy the same privacy properties) by sending a dummy message for each case where no output is sent (observation of the dummy output is then equivalent to observing no output in the original specification), and similarly can be done for other examples, so (R2) does not mean a real restriction in practice.

Definition 9. *Given a transaction specification with the restrictions (R1) and (R2), we define a* trace tr *as a tuple* $((a_1, r_1), \ldots, (a_k, r_k), (\mathcal{S}, \mathcal{P}))$*, where*

- *each a_i identifies one of the transactions,*
- *each r_i is an intruder recipe over labels $\{l_1, \ldots, l_{i-1}\}$,*
- *$(\mathcal{S}, \mathcal{P})$ is any configuration reached by the given sequence of transactions when the inputs are bound to the r_i and the outputs labeled l_i. (This is according to our definition of transaction semantics in §III-B.)*

We refer to the $\alpha(\mathcal{S})$, $\beta(\mathcal{S})$, and $\gamma(\mathcal{S})$ of a trace as expected; we may also refer to the $concr(\mathcal{S})$ of a trace, i.e., the (unique) ground messages bound to the labels l_i according to $\beta(\mathcal{S})$.

We call a sequence $(a_1, r_1), \ldots, (a_k, r_k)$ a symbolic trace *that represents all those traces that have this sequence of (a_i, r_i) transactions and inputs. The set of represented traces is finite, corresponding to the possible interpretations of the non-deterministic α and β variables.*

We say that (α, β)-privacy holds *in a trace $((a_1, r_1), \ldots, (a_k, r_k), (\mathcal{S}, \mathcal{P}))$ if it holds in state $\mathcal{S}$, and that it* holds *in a symbolic trace tr if it holds in all traces represented by tr.*

We call two traces $tr = ((a_1, r_1), \ldots, (a_k, r_k), (\mathcal{S}, \mathcal{P}))$ and $tr' = ((a_1, r_1), \ldots, (a_k, r_k), (\mathcal{S}', \mathcal{P}'))$ equivalent*, and write $tr \approx tr'$, if $concr(\mathcal{S}) \sim concr(\mathcal{S}')$(and, as indicated by pattern matching, the a_i, r_i, and k are the same).*

Let $traces(Spec)$ be the set of traces produced by a protocol specification $Spec$. We call two specifications $Spec$ and $Spec'$ trace equivalent*, and write $Spec \approx Spec'$, if for every trace $tr \in traces(Spec)$, there is a $tr' \in traces(Spec')$ with $tr \approx tr'$, and vice versa.*

A binary privacy question *is a specification of (α, β)-privacy transactions that do not contain any α-variables and make no α-release, together with a special transaction $T_{bin} = \mathsf{if}\ (\mathsf{init} \doteq \bot)\ \mathsf{then}\ \star\ x \in \{0, 1\}.\ \mathsf{init} := x$, where* init *is a distinguished memory cell initialized to $\bot$ and the other transactions may only read, but not modify, the value of* init*.*

The traces represented by a symbolic trace are actually easy to compute thanks to the restrictions (R1) and (R2): we follow the normal semantics, but for every step "$\star\ x \in D_x$" and for every step "$\diamond\ y \in D_y$", we keep the choice symbolic, and compute a set of corresponding α and γ that we attach to the respective possibility $(\mathcal{P}_i, \phi_i, struct_i)$ in the configurations. The δ is the same for all, and the β can be reconstructed from γ and the configuration. This is taking advantage of the fact that we already have a representation for all the possibilities (the $(\mathcal{P}_i, \phi_i, struct_i)$) at a given point. Now, there is however no possibility $(\mathcal{P}_i, \phi_i, struct_i)$ marked, but that marking is actually only needed in case the different possibilities have differences

in the number of sent and received messages, which we do not consider here due to the restrictions (R1) and (R2).

Note that every trace has at least one interpretation since every if-then-else has at least one branch that can execute, i.e., every transaction is applicable in every trace (it may just fail to actually do something).

This definition expresses the fact that trace equivalence is about the ability to distinguish between two systems that each reflect a particular choice of the privacy information. Relating this to the terms of (α,β)-privacy means thus that α is simply the secrecy of a bit x. We start by giving a formal definition of static equivalence of frames in (α,β)-privacy. To that end, we defined the axioms ϕ_{gen}, ϕ_{hom}, ϕ_{dom} and $\phi_{\sim}$ for any two frames F_1 and F_2 that shares the same domain D:

$$\begin{aligned}
\phi_{gen}(D) \equiv & \forall r. gen(r) \Leftrightarrow \big(r \in D \vee \textstyle\bigvee_{f^n \in \Sigma_{op}} \exists r_1,\ldots,r_n. \\
& \quad r = f(r_1,\ldots,r_n) \wedge gen(r_1) \wedge \ldots \wedge gen(r_n)\big) \\
\phi_{hom}(F) \equiv & \textstyle\bigwedge_{f^n \in \Sigma_{op}} \forall r_1,\ldots,r_n. \\
& \quad gen(r_1) \wedge \ldots \wedge gen(r_n) \implies \\
& \quad F[f(r_1,\ldots,r_n)] = f(F_1[r_1],\ldots,F_1[r_n]) \\
\phi_{dom} \equiv & F_1[l_1] = t_1 \wedge \ldots \wedge F_1[l_n] = t_n \\
\phi_{\sim}(F_1,F_2) \equiv & \forall r,s.\ gen(r) \wedge gen(s) \implies \\
& \quad F_1[r] = F_1[s] \Leftrightarrow F_2[r] = F_2[s]
\end{aligned}$$

Using these axioms, we can now define the symbol $\sim$ for any two frames:

Definition 10 (Static Equivalence of Frames)**.** *Two frames F_1 and F_2 with the same domain $\{m_1,\ldots,m_l\}$ of memory locations are* statically equivalent *(we write $F_1 \sim F_2$) iff $\phi_{hom}(F_1) \wedge \phi_{dom}(F_1) \wedge \phi_{hom}(F_2) \wedge \phi_{dom}(F_2) \wedge \phi_{\sim}(F_1,F_2)$ holds.*

We can now relate (α,β)-privacy in the binary case with trace equivalence (we first prove Theorem 3 as it will come in handy to prove Theorem 2):

Theorem 2. *Consider a binary privacy question Spec that meets (R1) and (R2). For each $b \in \{0,1\}$, let $Spec_b$ be the specialization of Spec where T_{bin} sets the choice of x to $\{b\}$. Then (α,β)-privacy holds in Spec iff $Spec_0 \approx Spec_1$.*

Here, one can see two fundamental differences between (α,β)-privacy and the trace equivalence approach: in trace equivalence, we do not have to introduce a distinction between high-level and low-level (but we simply have a single bit a secret); on the other hand, we cannot express more than a binary choice between two systems in one go: of course one can specify several binary questions, but each is an independent binary question. In contrast, in (α,β)-privacy we can have a choice between any finite number of models and we can let this develop during transitions, also dependent on the actions of the intruder. For this reason, we also formulate a different equivalence notion that is based on traces, but that, instead of distinguishing two systems, is based on the models of a formula α in a single system:

Theorem 3. *(α,β)-privacy holds in a symbolic trace $tr = (a_1,r_1),\ldots,(a_k,r_k)$ iff for every trace $(tr,(\mathcal{S},\mathcal{P}))$ and every Σ_0-interpretation $\mathcal{I}_0 \models \alpha(\mathcal{S})$, there exists a trace $(tr,(\mathcal{S}',\mathcal{P}'))$ such that $\mathcal{I}_0 \models \gamma(\mathcal{S}')$ and $concr(\mathcal{S}) \sim concr(\mathcal{S}')$.*

Proof. Let $tr = (a_1,r_1),\ldots,(a_k,r_k)$ and first suppose (α,β)-privacy is violated in tr, i.e., for some trace $(tr,(\mathcal{S},\mathcal{P}))$, (α,β)-privacy is violated in $\mathcal{S}$. This means that there is one model $\mathcal{I}_0$ of $\alpha(\mathcal{S})$ that cannot be extended to a model of β, i.e., for every $(\mathcal{P}_i, struct_i, \phi_i) \in \mathcal{P}$, either $\mathcal{I}_0 \not\models \phi_i$ or the $\mathcal{I}_0(struct_i) \not\sim concr(\mathcal{S})$. Thus, the intruder can exclude in state $\mathcal{S}$ every trace $(tr,(\mathcal{S}',\mathcal{P}'))$ where $\mathcal{I}_0 \models \gamma(\mathcal{S}')$. Since only the α- and β-variables are to interpret, this means that in every trace $(tr,(\mathcal{S}',\mathcal{P}'))$ where $\mathcal{I}_0 \models \gamma(\mathcal{S}')$, we have $concr(\mathcal{S}) \not\sim concr(\mathcal{S}')$.

Vice-versa, suppose there is a trace $(tr,(\mathcal{S},\mathcal{P}))$ and a model $\mathcal{I}_0$ of $\alpha(\mathcal{S})$ such that for every trace $(tr,(\mathcal{S}',\mathcal{P}'))$ where $\mathcal{I}_0 \models \gamma(\mathcal{S}')$, $concr(\mathcal{S}) \not\sim concr(\mathcal{S}')$. Then, similarly, for every $(\mathcal{P}_i, struct_i, \phi_i) \in \mathcal{P}$, either $\mathcal{I}_0 \not\models \phi_i$ or $\mathcal{I}_0(struct_i) \not\sim concr(\mathcal{S})$. Thus, $(tr,(\mathcal{S},\mathcal{P}))$ violates (α,β)-privacy. □

We can finally prove Theorem 2:

Proof. Note that $Spec$, $Spec_0$ and $Spec_1$ have the same set of symbolic traces. If a symbolic trace tr does not contain the special transaction T_{bin}, then all the concrete traces it represents in $Spec_0$, $Spec_1$ and $Spec$ are also the same, so up to taking the special transaction, there is no violation of (α,β)-privacy or trace distinction possible. Thus, for the rest of this proof, we consider only a symbolic trace tr that includes the special transaction T_{bin}. Note that in $Spec$, all concrete traces $(tr,\mathcal{S},\mathcal{P})$ represented by tr have thus $\alpha(\mathcal{S}) \equiv x \in \{0,1\}$.

Suppose now (α,β)-privacy holds in $Spec$ and suppose $(tr,\mathcal{S},\mathcal{P})$ is a trace that tr represents in $Spec_0$. Then, $\gamma(\mathcal{S})(x) \equiv 0$. This trace is also possible in $Spec$, and since the privacy holds, by Theorem 3, there exists a trace $(tr,\mathcal{S}',\mathcal{P}')$ in $Spec$ that supports the other model of α, namely $\gamma(\mathcal{S}')(x) \equiv 1$, and such that $concr(\mathcal{S}) \sim concr(\mathcal{S}')$. By construction, $(tr,\mathcal{S}',\mathcal{P}')$ is a trace of $Spec_1$. Thus, for every trace in $Spec_0$ exists an equivalent one $Spec_1$. By a similar proof, every trace in $Spec_1$ has an equivalent in $Spec_0$. Hence, $Spec_0$ and $Spec_1$ are trace equivalent.

Suppose, for the sake of contradiction, that (α,β)-privacy is violated in $Spec$. Then, by Theorem 3, there exists a trace $(tr,\mathcal{S},\mathcal{P})$ in $Spec$, say with $\gamma(\mathcal{S})(x) \equiv 0$ (the proof for the case $\gamma(\mathcal{S})(x) \equiv 1$ is analogous), and there is no trace $(tr,\mathcal{S}',\mathcal{P}')$ of $Spec$ such that both $\gamma(\mathcal{S})(x) \equiv 1$ and $concr(\mathcal{S}) \sim concr(\mathcal{S}')$. Obviously, $(tr,\mathcal{S},\mathcal{P})$ is a trace of $Spec_0$, but for all $(tr,\mathcal{S}',\mathcal{P}')$ of $Spec_1$, $concr(\mathcal{S}) \not\sim concr(\mathcal{S}')$ (since they have $\gamma(\mathcal{S})(x) \equiv 1$). Thus, $Spec_0$ and $Spec_1$ are not trace equivalent. □

Consider again the Basic Hash of Example 3. In approaches based on trace equivalence, one commonly specifies an equivalence between a system where the *same* tag performs any number of sessions with the reader versus a system where any number of *different* tags each perform one session with the reader. We can simulate this idea as a binary privacy question with the transaction T_{bin} as above, the same reader transaction

as in Example 3 and the following modified transaction for tags where $\mathsf{id_{fix}} \in \mathsf{Tags}$ is a fixed tag:

Tag

$\diamond\ T \in \mathsf{Tags}.\nu N.$
$\mathsf{if}\ (\mathsf{init} \doteq \bot)\ \mathsf{then}\ \mathsf{snd}(\mathsf{waiting\_for\_init}).0$
$\mathsf{else\ if}\ (\mathsf{init} \doteq 1)\ \mathsf{then}\ \mathsf{snd}(\mathsf{pair}(N, h(\mathsf{sk}(T), N))).0$
$\mathsf{else}\ \mathsf{snd}(\mathsf{pair}(N, h(\mathsf{sk}(\mathsf{id_{fix}}), N))).0$

Here, T is a β-variable, i.e., it would not in itself count as a privacy violation if the intruder finds out the identity of a tag; rather the privacy goal is that the intruder does not find out the choice of x in the first execution of T_{bin} (which is saved then in init). If this x was 0, then it is always the tag $\mathsf{id_{fix}}$ who performs the transaction, otherwise it is non-deterministically chosen from Tags.[4] It follows from Th. 3 that (α, β)-privacy of this system is equivalent to the trace equivalence between the system that non-deterministically chooses the tags and the system that always uses $\mathsf{id_{fix}}$. We emphasize that this specification is only for the comparison to trace equivalence, while the preferred way to specify unlinkability in (α, β)-privacy is as in Ex. 3 and OSK in § A: in each transaction of a tag T, the intruder learns only that $\star\ T \in \mathsf{Tags}$, but nothing more, in particular not whether two transactions are performed by the same tag. We see this as particularly declarative, namely not focusing on what the intruder should not find out, but rather what he may find out, and unlinkability thus means he does not find out anything except that T is a tag.

VI. Conclusions

(α, β)-privacy was proposed in [20] to fill the gap between the intuitive ideas and the mathematical notions used to formalize and reason about them. Here, we lifted (α, β)-privacy from a static approach to a dynamic one. Dynamic (α, β)-privacy considers one possible reality rather than two as it is common in approaches based on trace-equivalence. This means that (α, β)-privacy is now a privacy approach based on reachable states. Reachability makes the reasoning substantially easier for manual proofs, as in the DP-3T case study, and it paves the road towards automation. In particular, Theorem. 3 shows that the privacy problem can be reduced to static equivalence problems for each reachable state. This is the same as in trace-equivalence approaches where one also has a static equivalence problem for comparing two traces, but one additionally has to show that for every trace in one system, one can obtain an equivalent trace in the other. Static equivalence is decidable for many algebraic theories relevant in protocol verification [2]. However, the set of reachable states is in general infinite and transactions can obviously simulate Turing machines, thus (α, β)-privacy is still undecidable (as is "standard" protocol verification).

A first approach for automatically verifying (α, β)-privacy is given by Fernet and Mödersheim [15] which solves the message analysis problem defined in [20] for standard cryptographic operators. This is similar to methods for deciding static equivalence, but adapted to frames with privacy variables (without grounding them). As mentioned above, this is limited to (α, β)-privacy states with just one $struct$, whereas the present paper requires one to consider several $struct_i$ in order to handle the different possibilities arising from the evaluation of the conditions. Moreover, the present paper also explicitly models the interaction with the intruder, who in every state has an infinite choice of messages that he could send. We are currently working on the extension of the procedure of [15] to handle both problems for a bounded number of sessions. The infinite choice of the intruder can be represented finitely by a symbolic, constraint-based representation, not unlike existing tools on trace equivalence for bounded sessions like DEEPSEC [9].

For the unbounded case, Cortier et al. [11] observed that an obstacle in abstracting away sessions is the fact that some actions can only happen once, but in the abstraction can happen infinitely many times, which can produce false positives. They devised a type system that can in many cases help one to avoid the problem and allow for unbounded verification of trace equivalence. We plan to investigate whether similar typing ideas could also lead to a practically feasible analysis tool for (α, β)-privacy with unbounded sessions.

The fact that every state in dynamic (α, β)-privacy represents a single reality has another striking advantage. For many applications, it is interesting to take into account quantitative approaches. We see no obvious way to reason with them in equivalence-based specifications, but it is possible in (α, β)-privacy to make a declarative extension that integrates, e.g., non-determinism and probabilistic aspects, and we are currently working at including this in an extended version of this paper. Such a probabilistic extension also motivates a future comparison to *information flow* [16, 19].

Finally, it is interesting to consider whether there are any similarities between dynamic (α, β)-privacy and cryptographic notions like UC and IITM [18]. Also there we have the distinction between two levels, namely an ideal and a real system, which bears some similarity to our high-level α and the low-level β. A difference is that the ideal and real systems in composability frameworks describe interactions, i.e., what interface a component exposes to the outside, while α and β describe facts (what happened) and how these facts are logically related, e.g., how conditions in the program are related to the structure of messages observed by the intruder. Yet, the idea of (α, β)-privacy is indeed inspired by cryptography, namely zero-knowledge proofs: The idea of a zero-knowledge proof is that the intruder (or a dishonest verifier) does not learn anything from the proof but the statement being proved. This statement was the inspiration for α, i.e., the high-level information that this intruder is allowed to learn, whereas the cryptographic messages actually observed inspired the low-level information β, and using a fully-fledged logic for expressing α and β allows us to easily model how the intruder can make arbitrary logical deductions, e.g., if somebody proves to be over 21 implies that they are also over 18, but not necessarily over 65.

[4] For simplicity, we are not forbidding here that in two sessions we may use the same tag; for privacy it is of course sufficient that there are traces for $x \doteq 1$ where all tags are different.

Acknowledgments: This work was supported by the Sapere-Aude project "Composec: Secure Composition of Distributed Systems" (grant 4184-00334B of the Danish Council for Independent Research), by the EU H2020 SU-ICT-03-2018 project no. 830929 "CyberSec4Europe", and by the UKRI Trustworthy Autonomous Systems Hub (EP/V00784X/1).

References

[1] Martín Abadi, Bruno Blanchet, and Cédric Fournet. "The Applied Pi Calculus: Mobile Values, New Names, and Secure Communication". In: *J. ACM* (2018).

[2] Martín Abadi and Véronique Cortier. "Deciding knowledge in security protocols under equational theories". In: *Theor. Comput. Sci.* (2006).

[3] Myrto Arapinis et al. "Stateful applied pi calculus: Observational equivalence and labelled bisimilarity". In: *JLAMP* (2017).

[4] David Baelde, Stéphanie Delaune, and Solène Moreau. "A Method for Proving Unlinkability of Stateful Protocols". In: *CSF*. 2020.

[5] Bruno Blanchet. "An Efficient Cryptographic Protocol Verifier Based on Prolog Rules". In: *CSF*. 2001.

[6] Bruno Blanchet, Martín Abadi, and Cédric Fournet. "Automated verification of selected equivalences for security protocols". In: *JLAMP* (2008).

[7] Mayla Brusò, Konstantinos Chatzikokolakis, and Jerry den Hartog. "Formal Verification of Privacy for RFID Systems". In: *CSF*. 2010.

[8] Vincent Cheval, Hubert Comon-Lundh, and Stéphanie Delaune. "A procedure for deciding symbolic equivalence between sets of constraint systems". In: *Inf. Comput.* (2017).

[9] Vincent Cheval, Steve Kremer, and Itsaka Rakotonirina. "DEEPSEC: Deciding Equivalence Properties in Security Protocols – Theory and Practice". In: *IEEE SP*. 2018.

[10] Véronique Cortier, Michaël Rusinowitch, and Eugen Zalinescu. "Relating two standard notions of secrecy". In: *Log. Methods Comput. Sci.* (2007).

[11] Véronique Cortier et al. "A Type System for Privacy Properties". In: *CCS*. 2017.

[12] Stéphanie Delaune and Lucca Hirschi. "A survey of symbolic methods for establishing equivalence-based properties in cryptographic protocols". In: *JLAMP* (2017).

[13] Stéphanie Delaune, Mark Ryan, and Ben Smyth. "Automatic Verification of Privacy Properties in the Applied pi Calculus". In: *IFIPTM*. 2008.

[14] *DP-3T – Decentralized Privacy-Preserving Proximity Tracing*. 2020. URL: https://github.com/DP-3T/documents/blob/master/DP3T%20White%20Paper.pdf.

[15] Laouen Fernet and Sebastian Mödersheim. "Deciding a Fragment of (α, β)-privacy". In: *STM*. 2021.

[16] Joseph A. Goguen and José Meseguer. "Security Policies and Security Models". In: *IEEE SP*. 1982.

[17] Timothy Hinrichs and Michael Genesereth. *Herbrand Logic*. Tech. rep. Stanford University, 2006.

[18] Ralf Küsters, Max Tuengerthal, and Daniel Rausch. "The IITM Model: A Simple and Expressive Model for Universal Composability". In: *J. Cryptol.* 33.4 (2020).

[19] Heiko Mantel, David Sands, and Henning Sudbrock. "Assumptions and Guarantees for Compositional Noninterference". In: *CSF*. 2011.

[20] Sebastian Mödersheim and Luca Viganò. "Alpha-Beta Privacy". In: *ACM Trans. Priv. Secur.* (2019).

[21] Miyako Ohkubo, Koutarou Suzuki, and Shingo Kinoshita. "Cryptographic approach to "privacy-friendly" tags". In: *RFID Privacy Workshop*. 2003.

[22] Benedikt Schmidt et al. "Automated Analysis of Diffie-Hellman Protocols and Advanced Security Properties". In: *CSF*. 2012.

[23] Serge Vaudenay. *Analysis of DP3T*. Cryptology ePrint Archive, Report 2020/399. 2020.

Appendix

A. Linkability attack on OSK Protocol

Consider the OSK protocol [21]. Let $\mathsf{Tags} = \{t_1, \ldots, t_n\}$ be a finite set of tags, and $h/1$ and $g/1$ be two public uninterpreted functions (modeling one-way functions). Consider two families of memory cells, $r(\cdot)$ for the tags and $\mathsf{state}(\cdot)$ for the reader, whose initial values are both $\mathsf{init}(\cdot)$. Each tag T owns $r(T)$ and the reader owns the entire family $\mathsf{state}(T)$, i.e., T's "database". The tag updates its state $r(T)$ by applying a hash to it at each session and sending out the current key under g. The privacy goal is that the intruder cannot find out anything besides the fact that this action is performed by *some* tag $T \in \mathsf{Tags}$.

The reader receives a message of the form $g(h^j(\mathsf{init}(T)))$, and accepts it if its own database contains the value $h^i(\mathsf{init}(T))$ for some $i \leq j$ (to prevent replay). As in Example 3, the server has to perform a kind of guessing attack to figure out T and $j - i$. To model this, we introduce private uninterpreted functions $\mathsf{getT}/1$, $\mathsf{vgetT}/1$, $\mathsf{extract}/2$, $\mathsf{vextract}/2$, $\mathsf{init}/1$ with the algebraic properties

$$\begin{aligned}
\mathsf{getT}(g(\mathsf{init}(T))) &\approx T \\
\mathsf{getT}(g(h(X))) &\approx \mathsf{getT}(g(X)) \\
\mathsf{vgetT}(g(\mathsf{init}(T))) &\approx \mathsf{true} \\
\mathsf{vgetT}(g(h(X))) &\approx \mathsf{vgetT}(g(X)) \\
\mathsf{extract}(g(\mathsf{init}(T)), \mathsf{init}(T)) &\approx \mathsf{init}(T) \\
\mathsf{extract}(g(h(X)), \mathsf{init}(T)) &\approx h(\mathsf{extract}(g(X), \mathsf{init}(T))) \\
\mathsf{extract}(g(h(X)), h(X')) &\approx h(\mathsf{extract}(g(X), X')) \\
\mathsf{vextract}(g(\mathsf{init}(T)), \mathsf{init}(T)) &\approx \mathsf{true} \\
\mathsf{vextract}(g(h(X)), \mathsf{init}(T)) &\approx \mathsf{vextract}(g(X), \mathsf{init}(T)) \\
\mathsf{vextract}(g(h(X)), h(X')) &\approx \mathsf{vextract}(g(X), X')
\end{aligned}$$

getT extracts the name (if it is a valid message, as checked with vgetT) and $\mathsf{extract}$ extracts the current key (if it is a higher hash than the given key, as checked with $\mathsf{vextract}$). For applying the verifiers, we use the syntactic sugar try again to formulate that the reader, when successful, updates its own state and sends an ok message.

Tag	*Reader*
$\star\ T \in \mathsf{Tags}.$	$\mathsf{rcv}(x).$
$\mathsf{Key} := r(T).$	$\mathsf{try}\ T = \mathsf{getT}(x)\ \mathsf{in}\ s := \mathsf{state}(T).$
$r(T) := h(\mathsf{Key}).$	$\quad \mathsf{try}\ s' = \mathsf{extract}(x, s)\ \mathsf{in}$
$\mathsf{snd}(g(\mathsf{Key}))$	$\quad\quad \mathsf{state}(T) := h(s').\mathsf{snd}(\mathsf{ok})$

	α	β	γ	δ
1	$T_1 \in \mathsf{Tags}$	$concr[l_1] = g(\mathsf{init}(t_1)) \wedge struct[l_1] = g(\mathsf{init}(T_1))$	$T_1 \doteq t_1$	$r(T_1) := h(\mathsf{init}(T_1))$ if true
2	$T_2 \in \mathsf{Tags}$	$concr[l_2] = g(h(\mathsf{init}(t_1))) \wedge \exists i \in \{1,2\}.$ $i = 1 \wedge struct[l_2] = g(h(\mathsf{init}(T_1))) \wedge T_1 \doteq T_2$ $\vee\ i = 2 \wedge struct[l_2] = g(\mathsf{init}(T_2)) \wedge T_1 \not\doteq T_2$	$T_2 \doteq t_1$	$r(T_2) := h(h(\mathsf{init}(T_1)))$ if $T_1 \doteq T_2$ $r(T_2) := h(\mathsf{init}(T_2))$ if $T_1 \not\doteq T_2$
3		$concr[l_3] = \mathsf{ok} \wedge \exists i \in \{1,2\}.$ $i = 1 \wedge struct[l_3] = \mathsf{ok} \wedge T_1 \doteq T_2$ $\vee\ i = 2 \wedge struct[l_3] = \mathsf{ok} \wedge T_1 \not\doteq T_2$		$\mathsf{state}(T_1) := h(h(\mathsf{init}(T_1)))$ if $T_1 \doteq T_2$ $\mathsf{state}(T_2) := h(\mathsf{init}(T_2))$ if $T_1 \not\doteq T_2$
4		$T_1 \doteq T_2$		$\mathsf{state}(T_1) := h(\mathsf{init}(T_1))$ if $T_1 \not\doteq T_2$

Fig. 3. Execution of the OSK Protocol

We show how to reach a state of the OSK protocol that violates (α, β)-privacy with a linkability attack [4] (two sessions were initiated by the same tag). In short, the goal, or the intended released information, is that two tags initiated a session. In the end, the payload formula is: $\alpha \equiv T_1 \in \mathsf{Tags} \wedge T_2 \in \mathsf{Tags}$. The intruder does not know more about these tags, especially whether they are the same. If the technical information allows him to conclude that they are the same ($\beta \models T_1 \doteq T_2$), then (α, β)-privacy is violated.

The initial state is $\mathcal{S}_0 = \{\mathsf{true}, \mathsf{true}, \mathsf{true}, \mathsf{true}\}$. Consider a *Tag* transition. In the initial configuration, the possibilities are $\{(\star\ T_1 \in \mathsf{Tags}.\ \mathsf{Key}_1 := r(T_1).\ r(T_1) := h(\mathsf{Key}_1).\ \mathsf{snd}(g(\mathsf{Key}_1)).\ 0, \mathsf{true}, \{\})\}$ (with a variable-renamed copy of *Tag*). First, a value from Tags is chosen for T_1, i.e., we have $|\mathsf{Tags}|$ successor states (ER 1.). Let us focus on one successor state with the choice t_1, and thus γ_0 is augmented by $T_1 \doteq t_1$, and α and β are augmented by $T_1 \in \mathsf{Tags}$. We apply the rule for cell reads (NR 3.). Since δ_0 is still empty, we replace Key_1 by the initial value, $\mathsf{init}(T_1)$, in the rest of the process. We can now apply the rule for cell write (NR 5.), so that δ_0 is augmented by $r(T_1) := h(\mathsf{init}(T_1))$ if true. The marked process sends a message and we augment β by $concr[l_1] = g(\mathsf{init}(t_1)) \wedge struct[l_1] = g(\mathsf{init}(T_1))$ (ER 3.). There is just one possibility and the process has terminated, so the transaction is completed, getting to the state in the first line of Fig. 3 (we refer to the α in that line as α_1 and so on).

Consider a second *Tag* transition. The possibilities in the initial configuration are $\{(Tag(2), \mathsf{true}, struct)\}$, where $Tag(2)$ is a renaming of the tag process variables with index 2. We again look only at one successor state where, for the choice of T_2, we pick the same tag t_1 (ER 1.). (NR 3.) now introduces a case split: if $T_2 \doteq T_1$ then let $\mathsf{Key}_2 = h(\mathsf{init}(T_1)) \ldots$ else let $\mathsf{Key}_2 = \mathsf{init}(T_2) \ldots$. The conditional rule (NR 4.) splits it into two possibilities: $\{\underline{(P_a, T_1 \doteq T_2, struct_1)}, (P_b, T_1 \not\doteq T_2, struct_1)\}$, where P_a and P_b are instantiations of $r(T_2) := \mathsf{Key}_2.\mathsf{snd}(g(\mathsf{Key}_2))$ by $\mathsf{Key}_2 = h(\mathsf{init}(T_1))$ and $\mathsf{Key}_2 = \mathsf{init}(T_2)$, respectively, and where $struct_1$ is the frame from the first transaction. The case where $T_2 \doteq T_1$ is marked since this is the reality. The cell write rule (NR 5.) augments δ_1 by two lines (in either order): $r(T_2) := h(h(\mathsf{init}(T_1)))$ if $T_2 \doteq T_1$ and $r(T_2) := h(\mathsf{init}(T_2))$ if $T_2 \neq T_1$. It remains to send the outgoing message (ER 3.): β in line 2 of Fig. 3 reflects that the structural information is different. The structural knowledge of each possibility is updated with the respective version, let us call them $struct_a$ and $struct_b$. Both have terminated, so we have reached the end of the second transaction.

After a *Reader* transition, the possibilities are $\{\underline{(Reader(3), T_1 \doteq T_2, struct_a)}, (Reader(3), T_1 \not\doteq T_2, struct_b)\}$. We evaluate the receive step (ER 2.) and we have a choice of every recipe that the intruder can generate: we use l_2, i.e., the message from the second tag transaction. Note that $struct_a\{\!|l_2|\!\} = g(h(\mathsf{init}(T_1)))$ and $struct_b\{\!|l_2|\!\} = g(\mathsf{init}(T_2))$, which is what we insert for the received message x_3 in the respective processes. When the processes (successfully) try $\mathsf{getT}(x_3)$, we obtain let $T_3 = T_1$ and let $T_3 = T_2$, respectively. The state lookup (NR 3.) gives the initial value, as we have not yet written anything to the state cells. Thus, trying $\mathsf{extract}(T, s)$ will succeed and produce either $s'_3 := h(\mathsf{init}(T_1))$ or $s'_3 := \mathsf{init}(T_2)$. We amend δ (NR 5.) by the two lines (in either order) $\mathsf{state}(T_1) := h(h(\mathsf{init}(T_1)))$ if $T_1 \doteq T_2$ and $\mathsf{state}(T_2) := h(\mathsf{init}(T_2))$ if $T_1 \not\doteq T_2$. Both processes are now at a sending step (ER 3.). Even if the message is the same in both processes, we still have to consider a case distinction since the conditions differ, as shown in Fig. 3. Again, both processes have terminated, so the third transaction is finished.

Finally, after another *Reader* process, we have $\{\underline{(Reader(4), T_2 \doteq T_1, struct'_a)}, (Reader(4), T_2 \not\doteq T_1, struct'_b)\}$, where $struct'_a$ and $struct'_b$ are the *struct*s frames augmented with the last ok-message. Suppose the intruder chooses l_1 as a recipe for the received message (ER 2.), i.e., $struct'_a\{\!|l_1|\!\} = struct'_b\{\!|l_1|\!\} = g(h(\mathsf{init}(T_1)))$ for variable x_4. The next operation tries $\mathsf{getT}(x_4)$, which gives T_1 in any case. Looking up the $\mathsf{state}(T_1)$, (NR 3.) gives $s_4 := h(h(\mathsf{init}(T_1)))$ in the first possibility (due to $T_1 \doteq T_2$), and $s_4 := \mathsf{init}(T_1)$ in the second. The next try succeeds only for the second possibility, and we have: $\{\underline{(0, T_2 \doteq T_1, struct'_a)}, (\mathsf{snd}(\mathsf{ok}).0, T_2 \not\doteq T_1, struct'_b)\}$. The marked process terminates, so the intruder can rule out the second possibility (ER 3.). We augment β by the condition of the only remaining possibility, i.e., $T_1 \doteq T_2$. That is indeed a violation of privacy since we can now exclude all those models of α, where $T_1 \not\doteq T_2$.