Privacy as Reachability

Sébastien Gondron[1], Sebastian Mödersheim[2] and Luca Viganò[3]
[1,2]*DTU Compute*, Denmark, Kgs. Lyngby
[3]*Department of Informatics*, King's College London, London, UK
[1]spcg,[2]samo@dtu.dk, [3]luca.vigano@kcl.ac.uk

August 26, 2021

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., the number of votes and the result). We give a conservative extension by probabilistic variables. We prove useful properties and discuss examples, e.g., problems of linkability and vote copying, and the core of the privacy-preserving proximity tracing system DP-3T.

***Keywords*—** Formal Methods. Protocol security. Transition system. Probabilities. Linkability. Voting. Vote copying. DP-3T.

1 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 [2, 5, 9, 13]. While such approaches have uncovered vulnerabilities in a number of protocols, they rely on asking whether the intruder can distinguish two variants of a process; e.g., 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 for. 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 intuitive ideas of the privacy goals and the mathematical notions used to formalize and reason about them, (α, β)*-privacy* has been proposed in [27]. 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, e.g., a vote.

There are, however, two further open problems, which we tackle in this paper.

Problem 1. 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 (and so are the executed steps). Many works tame this difficulty by making the processes just differ in some message-subterms, so everything except these subterms is equal. One can obtain a verification question that is close to a reachability problem, which drastically reduces 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. (α, β)-privacy is until now only a static approach that does not reason about the development of a system, like the influence that the actions of an intruder can have on a system, and thus does not solve Problem 1 ... yet.

The *first main 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).

We define *dynamic (α, β)-privacy* that holds if (α, β)-privacy holds in every 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.

This does not solve all the challenges of automation: (i) (α, β)-privacy is in general undecidable, but for most reasonable protocols (including the algebraic model of cryptography) it is decidable, as it boils down to a static equivalence of frames; (ii) the set of reachable states is infinite. Symbolic and abstract interpretation methods still need to be developed.

We argue though that this approach is 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 topical case study we consider the core of the privacy-preserving proximity tracing system DP-3T [33]. We discover counter-examples for dynamic (α, β)-privacy, i.e., the intruder can make more deductions about the honest agents than released in α. Step by step, including more details in α, we obtain a characterization of all information that the system actually discloses, and then prove dynamic (α, β)-privacy. This can be helpful to understand the actual privacy impact of a system, and is also an answer to Problem 1.

Problem 2. Many approaches (e.g., quantitative information flow, differential privacy, etc.) reason about privacy by considering quantitative aspects and probabilities. Trace equivalence approaches are instead purely qualitative and possibilistic, and so is (dynamic) (α, β)-privacy; this is appropriate for many scenarios, but we give examples where probability distributions play a crucial role (i.e., in a purely possibilistic setting, there is no attack, but with probabilities there is).

As *second main contribution*, we give a *conservative* extension of (dynamic) (α, β)-privacy by probabilistic variables. As for non-deterministic variables (used when probabilities are irrelevant or when the intruder does not know the distribution), probabilistic variables can be sampled from a finite domain with a probability distribution, which may depend on probabilistic variables that were chosen earlier.

As proof-of-concept, we consider some simple examples, and then we show that a well-known problem of vote copying (e.g., in Helios, where a dishonest voter can copy an honest voter's vote) can be analyzed with probabilistic (α, β)-privacy in a new light: one can observe the influence on the distribution by the dishonest votes, where possibilistic models would not allow the deduction. The intruder almost becomes an empirical scientist who needs to decide when the distortion of the probabilities is significant to deduce how a particular voter voted. Hence, our approach successfully tackles Problem 2.

We prove two theorems for (dynamic) probabilistic (α, β)-privacy. We define a notion of *extensibility*, which says that β does not exclude choices of probabilistic variables. Theorem 1 says that if we prove *possibilistic* (α, β)-privacy for an extensible pair (α, β), then *probabilistic* (α, β)-privacy holds as well. Theorem 2 proves stability under background knowledge: if the intruder has additional background information α_0 (e.g., knowledge about the distribution of votes or particular voters), then in any state with an extensible pair (α, β), probabilistic $(\alpha \wedge \alpha_0, \beta \wedge \alpha_0)$-privacy still holds; the intruder does not learn more than what he already knew and what we deliberately release.

As a *third contribution*, we formalize the relationship between our approach and trace equivalence (Theorems 3 and 4).

We provide preliminaries in §2. In §3, we lift (α, β)-privacy from static to dynamic. In §4, we give a conservative extension of (dynamic) (α, β)-privacy with probabilities. We formalize the DP-3T protocol with dynamic possibilistic (α, β)-privacy in §A, we show how to formalize voting privacy goals with dynamic (α, β)-privacy in §B, and we prove the relationship with trace equivalence in §5, and discuss that with information flow in §6. Finally, in §7, we discuss future work.

2 Preliminaries

We adapt some useful notions from [20, 27, 18].

2.1 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 [20]. 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 $\Sigma = \Sigma_f \uplus \Sigma_i \uplus \Sigma_r$ 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 $x_1 = t_1 \wedge \ldots \wedge x_n = t_n$.

2.2 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. In order to represent the intruder's knowledge, we use frames.

Definition 1 (Frame)**.** *A* frame *is written as* $\digamma = \{\!|m_1 \mapsto t_1, \ldots, m_l \mapsto t_l|\!\}$*, 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* $\digamma\{\!|t|\!\}$ *for replacing in the term* t *every occurrence of* m_i *with* t_i*, i.e.,* $\digamma$ *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 *concr* and *struct* that always have the same domain D in any formula. Let *concr* and *struct* be unary function symbols, and *gen* a unary relation symbol defined by the following axioms:

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

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

Then, the formula

$$\begin{aligned}\phi \equiv\ & \mathit{struct}[l_1] = t_1 \wedge \ldots \wedge \mathit{struct}[l_n] = t_n \wedge \\ & \mathit{concr}[l_1] = t'_1 \wedge \ldots \wedge \mathit{concr}[l_n] = t'_n\end{aligned}$$

specifies two frames *concr* and *struct* with domain $D = \{l_1, \ldots, l_n\}$ and the augmentation with the axioms above means that *concr* and *struct* are *statically equivalent*: for any pair of recipes r and s that the intruder can generate, *concr* agrees on r and s iff *struct* does.

2.3 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 (Model-theoretical (α, β)-privacy [27]). *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 (model-theoretically) *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 this paper, we call model-theoretical (α, β)-privacy also *static possibilistic (α, β)-privacy* and, in contrast to [27], we allow β to have more free variables than α. All α formulae we consider in this paper are *combinatoric*, which means 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. For example, let $h \in \Sigma \setminus \Sigma_{op}$ and $\mathit{struct} = \{m_1 \mapsto h(v_1), m_2 \mapsto h(v_2)\}$ and $\mathit{concr} = \{m_1 \mapsto h(0), m_2 \mapsto h(1)\}$. Every model of β has the property $v_1 \neq v_2$. Suppose $\alpha \equiv v_1, v_2 \in \{0, 1\}$, then (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. However, if $\alpha \equiv v_1, v_2 \in \{0, 1\} \wedge v_1 + v_2 = 1$, then all models of α are compatible with β and privacy is preserved.

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} and $\phi_{\sim}$ where D is the set of labels occurring in β.

3 Transition Systems for Alpha-Beta-privacy

We lift the definition of static (α, β)-privacy to a dynamic one with transition systems. In §3.1, we describe the syntax of a protocol specification, notably the syntax of processes. We give the operational semantics for transition systems in §3.2 and define the state with, amongst other things, the following formulae: the *payload formula* α, the *technical information formula* β and the *truth formula* γ. We also define δ, which is a sequence of *conditional updates* on the cells, and η, which is a *probability decision tree* for the random variables. In §3.3, we show how to derive a legitimate linkability attack on the OSK protocol.

3.1 Syntax

We consider a number a *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).

In the processes, we talk about privacy variables of two sorts: *non-deterministic* or *random*. Each of them has a domain $D = \{c_1, \dots, c_n\}$, where $c_1, \dots, c_n$ are constants, i.e., a variable will be instantiated to one of these values. A random variable can also use $D_{prob} = \{c_1 : p_1, \dots, c_n : p_n\}$, where the p_i are probabilities s.t. they form a distribution, i.e., $p_i \in [0, 1]$ and $\sum_{i=1}^{n} p_i = 1$. We might omit the probabilities in D_{prob} when the distribution is uniform, and if only some of the p_i are made precise, then the rest of them are uniformly distributed amongst the remaining probability weight. 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 preliminaries).*

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

$$
\begin{array}{rll}
\mathcal{P}_l & ::= & \mathsf{mode}\ x \in D.\mathcal{P}_l \\
& | & \mathsf{mode}\ x \leftarrow D_{prob}.\mathcal{P}_l \\
& | & \mathsf{rcv}(x).\mathcal{P}_l \\
& | & x := \mathsf{cell}(s).\mathcal{P}_l \\
& | & \mathsf{if}\ \phi\ \mathsf{then}\ \mathcal{P}_l\ \mathsf{else}\ \mathcal{P}_l \\
& | & \nu \overline{N}.\mathcal{P}_r
\end{array}
\qquad
\begin{array}{rll}
\mathcal{P}_r & ::= & \mathsf{snd}(t).\mathcal{P}_r \\
& | & \mathsf{cell}(s) := t.P_r \\
& | & \mathsf{mode}\ \phi.\mathcal{P}_r \\
& | & 0
\end{array}
$$

where x ranges over variables; mode *is either $\star$ or $\diamond$, D is the finite domain of a non-deterministic variable; D_{prob} is the finite domain of a random variable; s and t range over terms,* cell *over families of memory cells, and ϕ over Herbrand formulae; and $\overline{N}$ is a set of fresh variables, i.e., that do not occur elsewhere. In ϕ formulae, we also allow the symbol $\mathcal{I}$ around terms, e.g., $x \doteq \mathcal{I}(x)$. This will refer during execution to the true interpretation of the variables as we define below.*

"$\mathsf{mode}\ x \in D$" is the picking of a non-deterministic variable, "$\mathsf{mode}\ x \leftarrow D_{prob}$" is the sampling of a random variable, and both are only released in β if mode is $\diamond$, and additionally in α if mode is $\star$. "$\mathsf{rcv}(x)$" is a standard message input, where the variable x is replaced with an actual received message. "$x \coloneqq \mathsf{cell}(s)$" is a cell read where x is replaced by a value stored in the memory cell. The conditional "$\mathsf{if}\ \phi\ \mathsf{then}\ \mathcal{P}_l\ \mathsf{else}\ \mathcal{P}_l$" is standard. "$\nu\overline{N}.\mathcal{P}_r$" creates a sequence of fresh variables; it does not recur on $\mathcal{P}_l$ but on $\mathcal{P}_r$, meaning that one can have new variables only once, directly before entering the right process. "$\mathsf{snd}(t)$" is a standard message output. "$\mathsf{cell}(s) \coloneqq t$" is a cell write that stores a term in the cell. "$\mathsf{mode}\ \phi$" releases a formula in α of the current state if mode is $\star$ and in γ if mode is $\diamond$. Finally, "0" is the null process.

We may write "$\mathsf{let}\ x = t$" for the substitution of all following occurrences of x by t. Another syntactic sugar concerns parsing of messages. For many (cryptographic) operators we may have a corresponding *destructor* and *verifier*, e.g., we often use the public functions $\mathsf{pair}/2$, $\mathsf{proj}_i/1$ and $\mathsf{vpair}/1$ with the properties $\mathsf{proj}_i(\mathsf{pair}(t_1, t_2)) \approx t_i$, and $\mathsf{vpair}(\mathsf{pair}(t_1, t_2)) \approx \mathsf{true}$. More generally, let f/n be a destructor (like the proj_i) and v/n a corresponding verifier (like vpair); 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 "$\mathsf{if}\ v(t_1, \ldots, t_n) \doteq \mathsf{true}\ \mathsf{then}\ \mathsf{let}\ t = f(t_1, \ldots, t_n).\mathcal{P}_1\ \mathsf{else}\ \mathcal{P}_2$". 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.

Example 1 (Basic Hash)**.** *As a first example, we consider the* Basic Hash *protocol [6]: 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 $\mathsf{sk}/1$ and $h/2$ be private functions. Each tag T has an immutable secret key $\mathsf{sk}(T)$. Let $\mathsf{pair}/2$, $\mathsf{vpair}/1$ 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
────────────────────────────
⋆ $T \in \mathsf{Tags}$.
$\nu N.\mathsf{snd}(\mathsf{pair}(N, h(\mathsf{sk}(T), N))).0$
```

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. 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* $\mathsf{vextract}(\mathsf{pair}(N, h(\mathsf{sk}(T), N))) \approx \mathsf{true}$.

```
Reader
────────────────────────────
$\mathsf{rcv}(t)$.
try $R = \mathsf{extract}(t)$ in
  $\mathsf{snd}(\mathsf{ok}).0$
```

Definition 4 (Requirements on Processes)**.** *We require that α formulae are over Σ_0 and contain only variables that were released in α. In* "$\mathsf{mode}\ x \in D.\mathcal{P}_l$", "$\mathsf{mode}\ x \leftarrow$

$D_{prob}.\mathcal{P}_l$", "$\mathsf{rcv}(x).\mathcal{P}_l$" and "$x \coloneqq \mathsf{cell}(s).\mathcal{P}_l$", we require that x cannot be instantiated twice, i.e., $\mathcal{P}_l$ contains neither "$\mathsf{mode}\ x \in D'$", nor "$\mathsf{mode}\ x \leftarrow D'_{prob}$", nor "$\mathsf{rcv}(x)$", nor "$x \coloneqq \mathsf{cell}(s')$". We also require that in different branches of conditionals, the same random and 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:

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

Note that in the case for D_{prob}, the right-hand side uses D, which is supposed to mean drop the probabilities from the domain: two branches are allowed to assign different probabilities to the elements of the domain, but it has to be the same domain. We also require that when a random variable y is sampled inside a conditional "$\mathsf{if}\ \phi\ \mathsf{then}\ \mathcal{P}_l\ \mathsf{else}\ \mathcal{P}_l$", ϕ can only contain conjunctions of the form $x \doteq c_i$, where x is a random variable that has previously been sampled, and if y is an α variable, then also x must be.

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):*

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

, and the free variables of a right process are all variables occurring in it.

3.2 Operational Semantics

We describe the operational semantics that lifts the definition of static (α, β)-privacy to a dynamic one with transition systems. We define *possibilistic dynamic (α, β)-privacy*, which intuitively holds if (α, β)-privacy holds in every state of the transition system. Let us start by defining the states of our transition system, where, for simplicity, we already include the tree η that we will need in Section 4, in which we discuss probabilities explicitly.

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

[2][27] only allowed that $fv(\alpha) = fv(\beta)$, but our constructions do not require it.

- *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, and*
- *η is a* probability decision tree*, which, for a sequence of random variables $x_1, \ldots, x_n$, has $n+1$ levels where every inner node on the i-th level is labeled x_i, and the leaves on level $n+1$ are unlabeled. To each variable x_i we associate a domain $D_i = \{c_{l_1}, \ldots, c_{l_i}\}$. Every x_i node in η has l_i children, and every branch from a node to its children is labeled with one of the c_{l_i} and a probability, so that the probabilities under a variable sum up to 1. In the following, we will use η in such a way that the first k levels are the random variables of α, and the remaining levels the random variables of β.*[3] *If there are no probabilistic variables, we may omit the tree η.*

The formulae α and β play the same roles than 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 $v_i = 1$, 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 come with a condition ϕ; these entries in general contain variables when the intruder does not know the concrete values. We describe η and the probabilistic mechanisms in Section 4.

During the execution of a transaction, the intruder will in general not know what exactly is happening, in particular in a conditional, it will not be generally clear which branch has been taken. Therefore, we define now the notion of possibilities that represent all possible choices, what that means for the condition and the structure of messages. One of the possibilities is marked to indicate which one is actually true.

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

[3] Note that the tree has exponential size in the number of variables, so for implementation in automated tools, a more efficient representation should be chosen, but for the conceptual level, this is irrelevant.

In a state, we can start the execution of any transaction from the protocol description as follows:

Definition 7 (Initial configuration)**.** *Consider a configuration* $\underline{(\mathcal{S}, \mathcal{P})}$*, 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 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* $(\mathcal{S}', \{(\theta(\mathcal{P}_l), \phi, \mathit{struct}) \mid (0, \phi, \mathit{struct}) \in \mathcal{P}\})$*.*

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 Section 3.3.

3.2.1 Normalization Rules

We have six normalization rules for a configuration: redundancy, cell reads, conditional, cell write, redundant entries in δ, release in α. 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.

Redundancy 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):

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

Cell Reads 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 $\phi_1, \ldots, \mathsf{cell}(s_n) := t_n$ if ϕ_n. Then, every configuration that starts with the reading of a memory cell is normalized via:

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

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

Conditional A conditional process is normalized via:

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

If the process "$\mathsf{if}\ \psi\ \mathsf{then}\ \mathcal{P}_1\ \mathsf{else}\ \mathcal{P}_2$" is marked, then, by construction, $\beta \wedge \gamma \models \phi$, thus either $\beta \wedge \gamma \models \phi \wedge \psi$ or $\beta \wedge \gamma \models \phi \wedge \neg\psi$. Accordingly, exactly one of the alternatives is marked.

Cell write A cell write process is normalized via:

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

where δ is augmented with the entry $\mathsf{cell}(s) \coloneqq t$ if ϕ. The order of these entries in δ depends on which normalizations are performed first, e.g., if we have $\{(\mathsf{cell}(s_1) \coloneqq t_1.0, \phi_1, \mathit{struct}_1), (\mathsf{cell}(s_2) \coloneqq t_2.0, \phi_2, \mathit{struct}_2)\}$, then normalization yields $\{(0, \phi_1, \mathit{struct}_1), (0, \phi_2, \mathit{struct}_2)\}$. Depending on whether we have started normalizing the first or the second possibility, the resulting δ is either $\delta \equiv \mathsf{cell}(s_1) \coloneqq t_1$ if ϕ_1, $\mathsf{cell}(s_2) \coloneqq t_2$ if ϕ_2 or $\delta \equiv \mathsf{cell}(s_2) \coloneqq t_2$ if ϕ_2, $\mathsf{cell}(s_1) \coloneqq t_1$ 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.

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

Release 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 $\diamond$, otherwise we ignore it:

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

Recall that in process specifications, the formula ϕ_0 may contain subterms of the form $\mathcal{I}(t)$, e.g., $x = \mathcal{I}(x)$. When adding to α or to γ, this subterm must be replaced by the actual value $\mathcal{I}(t)$ where $\mathcal{I}$ is the unique model of γ, i.e., the truth.

3.2.2 Evaluation Rules

We call a set of configurations *normalized* if normalization rules have been applied as far as possible. The first step of a normalized set of configurations is either a random sampling or 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 or random sampling), and they can produce non-normalized configurations. In this case, before another of the evaluation rules can be taken, the configurations have to be normalized again.

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

$$\begin{aligned}((\alpha, \beta, \gamma, \delta, \eta), &\{((\mathsf{mode}\ x \in D.\mathcal{P}_1, \phi_1, \mathit{struct}_1), \ldots, \\ &\quad (\mathsf{mode}\ x \in D.\mathcal{P}_n, \phi_n, \mathit{struct}_n))\}) \implies \\ ((\alpha', \beta', \gamma', \delta, \eta), &\{(\mathcal{P}_1, \phi_1, \mathit{struct}_1), \ldots, (\mathcal{P}_n, \phi_n, \mathit{struct}_n)\})\end{aligned}$$

where $\alpha' = \alpha \wedge x \in D$ if mode is $\star$ and $\alpha' = \alpha$ if mode is $\diamond$, $\beta' = \beta \wedge x \in D$ and $\gamma' = \gamma \wedge x \doteq c$ for whatever mode.

Random Sampling For the same reason as before, if the first step in the marked process is a random sampling, then all processes must start with a random choice of the same variable x. They may be sampled at different probabilities $D_{prob,1}, \dots, D_{prob,n}$, but the underlying set of elements D is identical. Then, for every $c \in D$ we have a configuration transition as follows:

$$\begin{gathered}((\alpha,\beta,\gamma,\delta,\eta),\{((\mathsf{mode}\ x \leftarrow D_{prob,1}.\mathcal{P}_1, \phi_1, \mathit{struct}_1), \dots, \\ (\mathsf{mode}\ x \leftarrow D_{prob,n}.\mathcal{P}_n, \phi_n, \mathit{struct}_n))\}) \implies \\ ((\alpha',\beta',\gamma',\delta,\eta'),\{(\mathcal{P}_1, \phi_1, \mathit{struct}_1), \dots, (\mathcal{P}_n, \phi_n, \mathit{struct}_n)\})\end{gathered}$$

where $\alpha' = \alpha \wedge x \in D$ if $\star$ is specified and $\alpha' = \alpha$ otherwise, $\beta' = \beta \wedge x \in D$ and $\gamma' = \gamma \wedge x \doteq c$ for whatever mode.

The tree η' is obtained from η as follows. First, let us describe the case that x has been sampled when mode was set to $\diamond$. Then, we replace the leaves of η with a new level of nodes labeled x, each of which have $|D|$ leaves as children. The probabilities on the new branches are determined as follows: for every x-labeled node, the path from the root determines an interpretation of all the random variables. We check which of the conditions of $\phi_1, \dots, \phi_n$ agree with this interpretation. If there is more than one, say ϕ_i and ϕ_j, then $D_{prob,i} = D_{prob,j}$ (due to our requirement that the conditions for a probability distribution for a random variable can only depend on the value of earlier random variables). Thus we can label the children of a given node accordingly.Note that it may happen that no ϕ_i agrees with the node; this is when the respective interpretation has already been excluded by α or β; in this case, the probability distribution is immaterial in the following, we just set it to uniform distribution.

If x has been sampled when mode was set to $\star$, i.e., is an α variable, the construction of η' is slightly different: we introduce the new level below the last α-variable in η, where the children of an x-node in η' are n copies of the subtree of the corresponding node in η. This is possible since the choice of an α variable by construction can only depend on other α variables.

Example 2 (Probability tree). *Consider the simple process that samples in that order a variable x with* mode *set to $\star$, a variable y with* mode *set to $\diamond$ and a variable z with* mode *set to $\star$:*

Example of a probability tree

$\star\ x \leftarrow \{1\colon \frac{1}{3}, 2\colon \frac{1}{3}, 3\colon \frac{1}{3}\}.$
$\diamond\ y \leftarrow \{1\colon \frac{1}{4}, 2\colon \frac{3}{4}\}.$
$\star\ z \leftarrow \{1\colon \frac{1}{2}, 2\colon \frac{1}{2}\}$

With our evaluation rules, this produces the probability decision tree in Figure 1 (note that the nodes with variables z appears before the nodes with variable y because z has been sampled with mode *set to $\star$):*

To see an another example of an η tree, see Figure 4.

Marked process receives 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., $\beta \models \mathit{gen}(r)$.

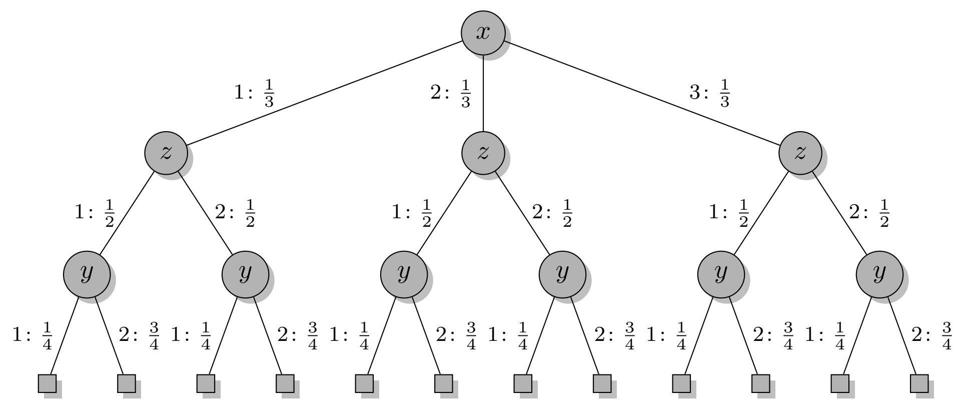

Figure 1: Example of a probability tree

For every such r, we have a configuration transition:

$$\{\underline{(\mathsf{rcv}(x).P_1, \phi_1, \mathit{struct}_1)}, \dots, (\mathsf{rcv}(x).P_k, \phi_k, \mathit{struct}_k)\} \to$$
$$\{\underline{(P_1[x \mapsto \mathit{struct}_1[r]], \phi_1, \mathit{struct}_1)}, \dots, (P_k[x \mapsto \mathit{struct}_k[r]], \phi_k, \mathit{struct}_k)\}$$

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.

Marked process sends 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

$$\{\underline{(\mathsf{snd}(m_1).P_1, \phi_1, \mathit{struct}_1)}, \dots, (\mathsf{snd}(m_k).P_k, \phi_k, \mathit{struct}_k)\} \cup \mathcal{P} \to$$
$$\{\underline{(P_1, \phi_1, \mathit{struct}_1 \cup \{\!|l \mapsto m_1|\!\})}, \dots, (P_k, \phi_k, \mathit{struct}_k \cup \{\!|l \mapsto m_k|\!\})\}$$

where l is a fresh label and $\mathcal{P}$ is a set of configurations that are finished, and we augment β by:

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

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 the m_i and this is the case iff ϕ_i holds.

Marked process has terminated 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:

$$\{\underline{(0,\phi_1,\mathit{struct}_1)},\ldots,(0,\phi_k,\mathit{struct}_k)\}\cup\mathcal{P}\to$$
$$\{\underline{(0,\phi_1,\mathit{struct}_1)},\ldots,(0,\phi_k,\mathit{struct}_k)\}$$

where $\mathcal{P}$ is a set of configurations that start with a send, and we augment β by $\phi_1 \vee \ldots \vee \phi_k$. In any case, we have thus finished the normalization and evaluation rules, and thus have reached a state.

After defining transition systems, let us define *dynamic* (α,β)*-privacy.* Note that this definition is possibilistic, so we refer to states without regards to η:

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 Definition 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.*

3.3 Linkability attack on OSK Protocol

As a second example, we consider the OSK protocol [28]. Consider a finite set of tags $\mathsf{Tags} = \{t_1,\ldots,t_n\}$, and let $h/1$ and $g/1$ be two public functions (modeling one-way functions). Consider also two families of memory cells: one for the tags, $r(\cdot)$, one for the reader, $\mathsf{state}(\cdot)$, 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 thus that the intruder cannot find out anything besides the fact that this action is performed by *some* tag $T \in \mathsf{Tags}$, i.e., that he cannot link two or more sessions to the same tag.

Tag

$\star\ T \in \mathsf{Tags}.$
$\mathsf{Key} := r(T).$
$r(T) := h(\mathsf{Key}).$
$\mathsf{snd}(g(\mathsf{Key})).0$

The reader should receive a message of the form $g(h^j(\mathsf{init}(T)))$, and would accept this message, if its own database contains the value $h^i(\mathsf{init}(T))$ for some $i \leq j$ (to prevent replay). Like in Example 1, the server has to perform a kind of guessing attack to figure out T and $j-i$. To model this simply we introduce private functions $\mathsf{getT}/1$, $\mathsf{vgetT}/1$, $\mathsf{extract}/2$, $\mathsf{vextract}/2$ with the algebraic properties in Figure 2. The getT function 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 the reader, who when successful, updates its own state and sends an ok

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

Figure 2: Algebraic properties for the OSK example.

message:

```
$Reader$
rcv($x$).
try $T$ = getT($x$) in
   $s$ := state($T$).
   try $s'$ = extract($x, s$) in
      state($T$) := $h(s')$.
      snd(ok).0
```

We illustrate the semantics by showing how to reach a state of the OSK protocol that violates (α, β)-privacy. The initial state is $\mathcal{S}_0 = \{\alpha_0 \equiv \mathsf{true}, \beta_0 \equiv \mathsf{true}, \gamma_0 \equiv \mathsf{true}, \delta_0 \equiv \mathsf{true}\}$; we omit η_0 since this example is purely possibilistic. We start with a transition of process *Tag*, and we thus get to the following possibilities (with a variable-renamed copy of *Tag*): $\{(\star\, T_1 \in \mathsf{Tags}.\ \mathsf{Key}_1 := r(T_1).\ r(T_1) := h(\mathsf{Key}_1).\ \mathsf{snd}(g(\mathsf{Key}_1)).\ 0, \mathsf{true}, \{\})\}$. The first step is choosing a value from Tags for T_1, i.e., we have $|\mathsf{Tags}|$ successor states. Let us focus on the choice t_1, and thus γ_0 is augmented by $T_1 \doteq t_1$, and α and β are augmented by $T_1 \in \mathsf{Tags}$. Then we apply the rule for cell reads. Since δ_0 is still empty, we just replace Key_1 by $\mathsf{init}(T_1)$ in the rest of the process. We can now apply the rule for cell write. This means δ_0 is augmented by $r(T_1) := h(\mathsf{init}(T_1))$ if true. is sending a message and we thus augment β by $\mathit{concr}[l_1] = g(\mathsf{init}(t_1)) \wedge \mathit{struct}[l_1] = g(\mathsf{init}(T_1))$. There is just one possibility in our configuration and it has terminated, so the transaction is completed, getting to the state shown in the first line of Figure 3 (we refer to the α in that line as α_1 and so on).

For the second transition, we use *Tag* once more, the new possibilities being $\{(\star\, T_2 \in \mathsf{Tags}.\ \mathsf{Key}_2 := r(T_2).\ r(T_2) := h(\mathsf{Key}_2).\ \mathsf{snd}(g(\mathsf{Key}_2)).\ 0, \mathsf{true}, \mathit{struct})\}$. Let us consider the transition where we pick for the choice of T_2 the same tag t_1. This time, the cell read introduces a case split:

$$\mathsf{if}\ T_2 \doteq T_1\ \mathsf{then\ let\ Key}_2 = h(\mathsf{init}(T_1)) \ldots\ \mathsf{else\ \ let\ Key}_2 = \mathsf{init}(T_2) \ldots .$$

The normalization of if splits this into two possibilities: $\{\underline{(P_a, T_1 \doteq T_2, \mathit{struct}_1)}, (P_b, T_1 \not\doteq T_2, \mathit{struct}_1)\}$ where P_a and P_b are instantiations of the process $r(T_2) := \mathsf{Key}_2.\ \mathsf{snd}(g(\mathsf{Key}_2))$ by $\mathsf{Key}_2 = h(\mathsf{init}(T_1))$ and $\mathsf{Key}_2 = \mathsf{init}(T_2)$, respectively, and where

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

Figure 3: Execution of the OSK Protocol

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 normalization of the cell writes 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 and the structural information is now different, leading to the β in line 2 of Figure 3. The corresponding structural knowledge of each possibility is updated with the respective version, let us call them struct_a and struct_b in the following. Since they both have terminated, we have reached the end of the second transaction.

The new possibilities are $\{\underline{(\mathit{Reader}(3), T_1 \doteq T_2, \mathit{struct}_a)}, (\mathit{Reader}(3), T_1 \not\doteq T_2, \mathit{struct}_b)\}$ after a *Reader* transition, where $\mathit{Reader}(3)$ is a renaming of the reader process variables with index 3. We evaluate the receive step and here we have a choice of every recipe that the intruder can generate: we use l_2, i.e., the message from the second token transaction. Note that $\mathit{struct}_a[l_2] = g(h(\mathsf{init}(T_1)))$ and $\mathit{struct}_b[l_2] = g(\mathsf{init}(T_2))$, 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 get thus let $T_3 = T_1$ and let $T_3 = T_2$, respectively. The state lookup gives the respective initial value, since we have not yet written anything to the state cells. Thus also trying $\mathsf{extract}(T, s)$ will succeed and either produce $s_3 := h(\mathsf{init}(T_1))$ or $s_3 := \mathsf{init}(T_2)$. We thus amend δ 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 $T_1 \not\doteq T_2$. In both processes, we are now at a sending step. 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 line 3 of Figure 3 (this formula can be simplified, of course). Again, both processes are empty, so we have finished the third transaction.

Finally, we have $\{\underline{(\mathit{Reader}(4), T_2 \doteq T_1, \mathit{struct}'_a)}, (\mathit{Reader}(4), T_2 \not\doteq T_1, \mathit{struct}'_b)\}$ after performing another *Reader* process, with $\mathit{Reader}(4)$ again being a renaming of variables with index 4 and struct'_a and struct'_b are the augmented *struct*s frames with the last ok-message. We consider the intruder choosing l_1 as a recipe for the received message, i.e., $\mathit{struct}'_a[l_1] = \mathit{struct}'_b[l_1] = g(h(\mathsf{init}(T_1)))$ for variable x_4. The next operation tries $\mathsf{getT}(x_4)$, which gives T_1 in any case. Looking up the $\mathsf{state}(T_1)$ gives $s_4 := h(h(\mathsf{init}(T_1)))$ in the first possibility (due to $T_1 \doteq T_2$), and the initial value $s_4 := \mathsf{init}(T_1)$ in the second. Thus, the next try succeeds only for the second possibility, and we have: $\{\underline{(0, T_2 \doteq T_1, \mathit{struct}'_a)}, (\mathsf{snd}(\mathsf{ok}).0, T_2 \not\doteq T_1, \mathit{struct}'_b)\}$. Now, the evaluation rule for the marked possibility being finished tells us: the second possibility cannot be the case because it would send a message, and the intruder can see that this does not happen, so we can 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 $T_1 \not\doteq T_2$.

4 Probabilistic privacy

In the previous section, we introduced random privacy variables, but we did not fully explain their purpose and functioning yet. For some problems that satisfy dynamic possibilistic (α, β)-privacy, we might want to refine the analysis. Extending all the models of α to a model of β does not mean that we do not leak information on the likelihood of a specific model of α through the technical information in β.

4.1 Probabilistic Alpha-Beta-Privacy

We thus propose a conservative extension first of static and then of dynamic (α, β)-privacy in order to add probabilities. We still consider a protocol specification, but, while the problems that we considered before had only non-deterministic privacy variables, we now consider problems that also have some privacy variables that can be sampled according to a probability distribution. These random variables are organized in a probability decision tree as defined via η in the previous section. We define the probability of a model of α, and intuitively, if the models of β yield a different probability for the model of α that they extend, then β violates the privacy of α in a probabilistic sense. However, the difference in probabilities might not be *significant*, so it is left to the modeler's appreciation to define an acceptable threshold.

Before giving formal definitions, we illustrate the interest of a probabilistic definition of (α, β)-privacy by means of a simple example, the well-known *Monty Hall problem* that originates from the game show "Let's Make a Deal" [29].

Example 3 (Monty Hall problem)**.** *There are three doors, and only one of these doors hides a valuable prize; the two other doors hide joke prizes*[4]*. We model this situation with a random variable x. The prize has an equal probability to be found behind each door. We write $x \leftarrow \{1, 2, 3\}$ to say that x is sampled from these values with a uniform distribution. The trader (this is "Let's Make a Deal" lingo) has to choose a door and communicate (send) their choice to the host, Monty Hall. Monty then opens one of the two doors that were not chosen by the trader and that does not hide the prize. However, from the point of view of the trader, Monty chooses a door randomly between the two remaining doors. Monty then gives the trader the choice whether they want to switch door or not, before ultimately the two remaining doors are opened and the location of the prize is revealed. Let us formalize this with a dynamic probabilistic (α, β)-privacy and define the Monty process.*

```
Monty
$\star\ x \leftarrow \{1, 2, 3\}$.
$\mathsf{rcv}(\mathsf{choice})$.
$\star\ \mathsf{open} \leftarrow \{1, 2, 3\} \setminus \{x, \mathsf{open}\}$.
$\star\ x \not\doteq \mathsf{open}$.
$\mathsf{snd}(\mathsf{open}).0$
```

$\mathsf{open} \leftarrow \{1\colon \frac{1}{3}, 2\colon \frac{1}{3}, 3\colon \frac{1}{3}\} \setminus \{x, \mathsf{open}\}$ *is syntactic sugar for:*

```
if $x \doteq 1$ then
    if choice $\doteq 2$ then open $\leftarrow \{1\colon 0, 2\colon 0, 3\colon 1\}$.
    else if choice $\doteq 3$ then open $\leftarrow \{1\colon 0, 2\colon 1, 3\colon 0\}$.
    else open $\leftarrow \{1\colon 0, 2\colon \frac{1}{5}, 3\colon \frac{1}{2}\}$.
else if $x \doteq 2 \ldots$
```

Before the execution of the process, $\alpha_0 \equiv \mathsf{true}$. After, it is revealed as part of α that the prize was put behind a door chosen with a uniform probability. It is also revealed which door has been opened, thus which door does not hide the prize. Let us now consider one concrete reachable state after the execution of the Monty process, namely

[4]The joke prizes were called *zonk* in the game

one where the intruder in the role of trader chose, or sent, 1 *for the choice, the prize is behind the third door and Monty opened the second door:* $\gamma \equiv x \doteq 3 \ \wedge \ \mathsf{choice} \doteq 1 \ \wedge \ \mathsf{open} \doteq 2$*. We then have:*

$$\alpha \equiv x \in \{1,2,3\} \ \wedge \ x \not\doteq 2, \ \textit{i.e.,} \ \alpha \equiv x \in \{1,3\}$$

Intuitively, the trader thinks they have an equal chance to find the prize either behind the door that they initially choose or behind the other door. However, since they know the way Monty chooses the door to open, β *and* η *are as follows, where* l *is a fresh label generated during the evaluation of the transition:*

$$\beta \equiv \alpha \wedge \mathit{concr}[l] = 2 \wedge \mathit{struct}[l] = \mathsf{open}$$

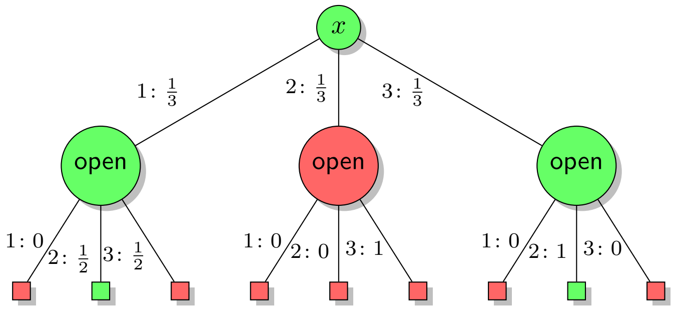

Figure 4: Probability tree for the Monty Hall problem

We colored in red the nodes for which no model exists in the considered state, e.g., the branch for which $x \doteq 2$ *is not supported by any valid model. The table below shows the models of* α *and their extension to* β *for this concrete reachable state where AP stands for the* absolute probability*, i.e., the multiplication of the probability of events for that model, and NP for the* normalized probability*, i.e., so that the column sums up to* 1*. Remember that this does not yet follow from our transition systems definition, but rather motivates the following definition of the probability of the models of* α *and* β*. We can see now the well-known glitch in the Monty Hall problem: according to* α*, the probability of* $x \doteq 1$ *and that of* $x \doteq 3$ *are both* $\frac{1}{2}$*, but according to* β*,* $x \doteq 1$ *has probability* $\frac{1}{3}$ *whereas* $x \doteq 3$ *has* $\frac{2}{3}$*, i.e.,* β *has been leaking information about* x *that was not contained in* α*. This reflects the advice that in this situation the trader should switch to door* 3*.*

Since we propose below a conservative extension, we want to be able to talk about models that differ only by the choice of non-deterministic variables. To this aim, we define an equivalence relation for the interpretations over the free random variables of α (respectively, of β): given two models of α (of β) $\mathcal{I}_1$ and $\mathcal{I}_2$, $\mathcal{I}_1 =_r \mathcal{I}_2$ iff $\mathcal{I}_1(x) = \mathcal{I}_2(x)$ for all $x \in \mathit{fv}_r(\alpha)$ (for all $x \in \mathit{fv}_r(\beta)$) where $\mathit{fv}_r(\cdot)$ refers to free random variables of a formula. We define an equivalence class $[\mathcal{I}]$ ($[\mathcal{I}']$) induced by this relation for all Σ_0-interpretations $\mathcal{I}$ of α (for all Σ-interpretation $\mathcal{I}'$ of β).

Since there are finitely many free random variables of α (of β), there are finitely many equivalence classes of probabilistic models of α, i.e., there exist $\mathcal{I}_1, \ldots, \mathcal{I}_k$, s.t. $k \in \mathbb{N}^+$, such that $[\mathcal{I}_1] \cup \ldots \cup [\mathcal{I}_k]$ is a partition of the models of α (respectively, there exists

α-Model	α-AP	α-NP	β-Model	β-AP	β-NP
			$\mathsf{open} = 1$	impossible	
$x = 1$	$\frac{1}{3}$	$\frac{1}{2}$	$\mathsf{open} = 2$	$\frac{1}{6}$	$\frac{1}{3}$
			$\mathsf{open} = 3$	impossible	
			$\mathsf{open} = 1$	impossible	
$x = 3$	$\frac{1}{3}$	$\frac{1}{2}$	$\mathsf{open} = 2$	$\frac{1}{3}$	$\frac{2}{3}$
			$\mathsf{open} = 3$	impossible	

Table 1: Models of α and their extensions to β for the Monty Problem

$\mathcal{I}_1, \ldots, \mathcal{I}_{k'}$, s.t. $k' \in \mathbb{N}^+$, such that $[\mathcal{I}_1] \cup \ldots \cup [\mathcal{I}_{k'}]$ is a partition to their extensions to models of β).

Definition 9 (Absolute and Normalized Probabilities)**.** *Given two formulae α and β, and a probability decision tree η, an* interpretation class *$[\mathcal{I}]$ corresponds to a unique path in η starting at the root node, and we define its* absolute probability *$\mathcal{P}_{abs,\eta}([\mathcal{I}])$ as the product of the probabilities along the path. Note that if $[\mathcal{I}]$ is an α interpretation class, the path traverses just the upper part of η that corresponds to the free variables of α while if $[\mathcal{I}]$ is a β interpretation the path reaches a leaf.*

Let $[\mathcal{I}_1], \ldots, [\mathcal{I}_k]$ be the model classes of α (of β); we define the normalized probability *of each interpretation class as:*

$$\mathcal{P}_\eta([\mathcal{I}_i]) = \frac{\mathcal{P}_{abs,\eta}([\mathcal{I}_i])}{\sum_{j=1}^{k} \mathcal{P}_{abs,\eta}([\mathcal{I}_j])} .$$

Note that $\mathcal{P}_\eta([\mathcal{I}])$ is defined so it does not depend on the choice of the representative $\mathcal{I}$ of the equivalence class $[\mathcal{I}]$.

We allow β to have more free variables than α. In particular, it allows β to have more free random variables than α. The intuitive idea to formulate probabilistic (α, β)-privacy as a conservative extension is to require that the sum of the probabilities of the equivalence classes of β, when restricted to the free variables of α and the payload alphabet Σ_0, is equal to the probabilities of the equivalence classes of α:

Definition 10 (Probabilistic (α, β)-privacy)**.** *Let $\Sigma_0 \subsetneq \Sigma$ and consider a formula α over Σ_0 and a formula β over Σ, s.t. $\beta \models \alpha$, $fv(\alpha) \subseteq fv(\beta)$, and both α and β are consistent, and η is the probability decision tree. We say that* (α, β)-privacy holds probabilistically *iff (α, β)-privacy holds and*

$$\mathcal{P}_\eta([\mathcal{I}_0]) = \sum_{i=1}^{k} \mathcal{P}_\eta([\mathcal{I}_i])$$

for every model $\mathcal{I}_0$ of α, and for the models $[\mathcal{I}_1], \ldots, [\mathcal{I}_k]$ of β (partitioned by equivalence class) s.t. $\mathcal{I}_i|_{\Sigma_0, fv(\alpha)} = \mathcal{I}_0$ for every $i \in \{1, \ldots, k\}$. We say that (α, β)-privacy holds probabilistically and dynamically *iff (α, β)-privacy holds probabilistically in every reachable state.*

Intuitively, this means that the probability of every model of β that agrees on the payload part, when considered together, equals the probability of the original model of α. With this definition, we can see that the Monty Hall problem from Example 3 satisfies possibilistic (α, β)-privacy in the state we considered but breaks probabilistic (α, β)-privacy. Now, let us see how we could correct the protocol:

Example 4 (Alternative Monty Hall)**.** *Suppose the door that Monty opens after the choice of the trader is taken randomly between the doors that were not chosen by the trader, thus including the one hiding the prize (even though this might shorten the game). We propose the process:*

```
Alternative Monty
$\star\ x \leftarrow \{1, 2, 3\}$.
rcv(choice).
$\star$ open $\leftarrow \{1, 2, 3\} \setminus \{$choice$\}$.
if open $\doteq x$ then
    $\star\ x \doteq$ open.
else
    $\star\ x \not\doteq$ open.
snd(open).0
```

We consider the same reachable state as in Example 3. α, β and γ are similar. The probability tree has a similar form but we update the probability on the branches accordingly. Let us look again at the models of α and their extension to β. This time again, the probability of $x \doteq 1$ and $x \doteq 3$ is both $\frac{1}{2}$ according to α and so it is according to β. β is not leaking information about x that was not contained in α anymore. This reflects that the trader cannot adopt a better strategy.

The difference between the original Monty Hall problem in Example 3 and the alternative Monty Hall problem in Example 4 is that the choice of the door that Monty opens is independent of where the prize is located in the second case. In other words, the free random variable of β, namely the choice of the opened door, could have been 2 or 3, whatever the free random variable of α, namely the location of the prize; that is, the choice of the location of the prize does not influence the probability distribution of the choice of the opening of the door. We can actually identify a condition under which if (α, β)-privacy holds possibilistically, then (α, β)-privacy also holds probabilistically:

Definition 11 (Extensibility)**.** *Let $\Sigma_0 \subsetneq \Sigma$ and consider a formula α over Σ_0 and a formula β over Σ, s.t. $\beta \models \alpha$, $fv(\alpha) \subseteq fv(\beta)$ and both α and β are consistent. We say that a pair (α, β) is* extensible *if it is possible to extend every model of α by a number of models of β that cover the whole domain of the free random variables occurring exclusively in β, i.e., for all $\mathcal{I}_0 \models \alpha$. for all $\sigma\colon y_1, \ldots, y_k \mapsto dom(y_1) \times \cdots \times dom(y_k)$. there exists $\mathcal{I} \models \beta \wedge \sigma$ such that $\mathcal{I}|_{\Sigma_0, fv(\alpha)} = \mathcal{I}_0$.*

The definition of extensibility does not refer to probabilities or to a probability tree η. When we have extensibility, it means that the $[\mathcal{I}_1], \ldots, [\mathcal{I}_k]$ are exactly all the leaves under $\mathcal{I}_0$, so the probability of that subtree is 1, as can be seen by induction. As a consequence, the absolute probability of $[\mathcal{I}_0]$ is the same as the sum of the absolute probabilities of the $[\mathcal{I}_i]$.

We can prove that if (α, β)-privacy holds possibilistically for (α, β) that is extensible, then (α, β)-privacy also holds probabilistically. This is only a sufficient condition: probabilistic (α, β)-privacy may hold, even if (α, β) is not extensible.

Theorem 1. *Let $\Sigma_0 \subsetneq \Sigma$ and consider a formula α over Σ_0 and a formula β over Σ, s.t. $\beta \models \alpha$, $fv(\alpha) \subseteq fv(\beta)$ and both α and β are consistent. Let η be the probability decision tree. If (α, β)-privacy holds possibilistically, i.e., for every $\mathcal{I}_0 \models \alpha$, there exists $\mathcal{I} \models \beta$ such that $\mathcal{I}|_{\Sigma_0, fv(\alpha)} = \mathcal{I}_0$, and (α, β) is extensible, then (α, β)-privacy holds probabilistically. We extend this theorem to dynamic (α, β)-privacy as expected.*

4.2 The intruder as an empirical scientist

Probabilistic (α, β)-privacy is a "sharp sword": a relatively small shift in probabilities due to the information in β is already a violation of (α, β)-privacy, while it may be too insignificant to be beneficial for the intruder. One wants to avoid even minimal shifts, because in many areas of security we have seen how an intruder can actually "amplify" tiny imperfections in a system, so that they become significant. Nonetheless, it can be insightful to analyze also a system that does not satisfy probabilistic (α, β)-privacy for a weaker goal. In particular, we want to regard now the intruder as an empirical scientist who conducts an experiment, where they manipulate some "independent variables", and try to observe a significant effect.

The Helios voting protocol originally allowed an intruder, as a dishonest voter, to copy votes of others [10]. For a small number of voters, this can lead to a violation of possibilistic (α, β)-privacy, e.g., a and b are honest, c dishonest, and we consider a binary vote between 0 and 1; then the intruder can copy a's vote; the result is greater than two iff a voted 1. Thus, possibilistic (α, β)-privacy does not hold. It would, however, if the cardinality would not allow for such a deduction, but if we look at the problem probabilistically, i.e., we know a *distribution* of the votes, then this yields a shift in probabilities.

We could see this as an experiment, where the intruder is not really interested in breaking the privacy of the entire vote, but rather targeting the privacy of a particular voter a—even sacrificing their chance to make their preferred option win by copying a's vote. If this is the actual goal, then actually it makes sense to focus on this single vote also in the privacy goal, i.e., to say $\alpha \equiv x_1 \in \{0, 1\}$, and β containing all the other votes and the result. This means that all the other voters and the result are for the intruder right now not interesting but just how a has voted. The copying of α's vote is thus the experiment the intruder takes if we regard a's vote as an *independent variable* of a scientific experiment.

Let us make such an experiment for 6 honest and 2 dishonest voters, meaning the intruder can copy the vote of a two times. The result is 5 votes for candidate 1. Let $v_1, \dots, v_6$ be random variables and v_7, v_8 are the votes controlled by the intruder.

$$\begin{array}{rcl} \alpha & \equiv & v_1 \in \{0,1\} \\ \beta & \equiv & \alpha \ \wedge \ v_2, \cdots, v_6 \in \{0,1\} \ \wedge \ v_7, v_8 \in \{0,1\} \\ & & \wedge \ \sum_{i=1}^{8} v_i = 5 \ \wedge \ v_7 \doteq v_1 \ \wedge \ v_8 \doteq v_1 \, . \end{array}$$

By analyzing the previous elections, the intruder knows the random variables are chosen following the probability distribution $\{0\colon \frac{2}{3}, 1\colon \frac{1}{3}\}$. It is clear that α has only two models: $\mathcal{I}_1$ s.t. $\mathcal{I}_1(v_1) = 0$ and $\mathcal{P}_\eta([\mathcal{I}_1]) = \frac{2}{3}$, $\mathcal{I}_2$ s.t. $\mathcal{I}_2(v_1) = 1$ and $\mathcal{P}_\eta([\mathcal{I}_2]) = \frac{1}{3}$. Both these models can be extended to models of β. Possibilistically, (α, β)-privacy holds. However, if we normalize the sum of the probabilities of the interpretation class for the β models, we obtain for the first one $\mathcal{P}_1 \approx 2.44$ and for the second $\mathcal{P}_2 \approx 97.56$. In this case, there is nearly no doubt for the intruder that voter a voted for 0.

In general, the intruder may set a threshold for being *convinced* for a particular model, e.g., when all other models together have a probability of at most 0.05. One can thus also calculate, given the number of honest and dishonest voters, as well the probability distribution, whether the intruder has even a chance to find a significant result. For instance, in an election with 100 votes where the intruder controls just five votes, a's privacy is well protected, unless there is a candidate with a very low popularity, say 0.01, and a votes for that candidate. We explain in Appendix B how to model voting protocols with dynamic (α, β)-privacy.

4.3 Background Knowledge

The authors of [27] explained what happens when the intruder can use background knowledge outside our formal method. Consider a small village where everybody votes the same way. A new person settles in, and in the next election, a vote for the opposition party is cast. Even a perfect privacy-preserving voting system cannot prevent the intruder to infer that it is quasi-certain this vote comes from the newcomer. The authors proved that the possibilistic (α, β)-privacy is *stable* under an arbitrary consistent intruder background knowledge. Unfortunately, this property does not hold for possibilistic (α, β)-privacy in general. Consider:

```
$\star\ x \leftarrow \{0\colon \frac{1}{2}, 1\colon \frac{1}{2}\}$.
$\star\ y \in \{0, 1\}$.
if $x \doteq 1$ then
    $\star\ z \leftarrow \{0\colon \frac{1}{2}, 1\colon \frac{1}{2}\}$.
    snd($y \oplus z$)
else
    $\star\ z \leftarrow \{0\colon 0, 1\colon 1\}$.
    snd($z$)
```

While this is an artificial example, it has some similar patterns to Σ-protocols, i.e., giving out a "secret" y "blinded" by z in one case, and just z in another case. Suppose this process executes and the intruder observes that it sends the value 1. Then, we have the models depicted in the following table where (we have arranged the items, so that we can summarize them to model classes) we have shaded red all those models that get excluded by the fact that we have observed sending 1 (because this means that either $x = 1$, and then y and z must be different, or $x = 0$, then z must be 1).

So far, (α, β)-privacy holds probabilistically: all model classes of β where $x \doteq 1$ have together probability $\frac{1}{2}$ as does the corresponding α-class, and the same for $x \doteq 0$. If we now have the background knowledge that $y \doteq 1$, then this excludes some further models that are shaded in blue in the from x. While this still preserves *possibilistic* (α, β) privacy, it violates *probabilistic* (α, β) privacy: according to β, $x \doteq 1$ is half as likely as $x \doteq 0$.

Indeed, this is a very constructed example (where for $x \doteq 0$, z is not really random any more), and actually in many practically relevant examples, background knowledge indeed also preserves probabilistic (α, β)-privacy for extensible pair (α, β):

Theorem 2 (Stability Under Background Knowledge)**.** *Let α and β be given so that (α, β)-privacy holds possibilistically, and (α, β) is extensible. Let α_0 be a Σ_0-formula. Then $(\alpha \wedge \alpha_0, \beta \wedge \alpha_0)$-privacy holds probabilistically. We extend this result for dynamic (α, β)-privacy as expected.*

α-Prob	x	y	z	β-Prob
$\frac{1}{2}$	0	0	0	0
	0	1	0	
	0	0	1	$\frac{1}{2}$
	0	1	1	
$\frac{1}{2}$	1	0	0	$\frac{1}{4}$
	1	1	0	
	1	0	1	$\frac{1}{4}$
	1	1	1	

Table 2: Example: for non-extensible (α, β), probabilistic privacy is in general not stable under background knowledge.

Proof. If (α, β) is extensible, then also $(\alpha \wedge \alpha_0, \beta \wedge \alpha_0)$ is extensible. Since (α, β)-privacy holds possibilistically, by stability under background knowledge, $(\alpha \wedge \alpha_0, \beta \wedge \alpha_0)$-privacy holds possibilistically, and since it is still extensible, it also holds probabilistically by Theorem 1. $\square$

5 Comparison with Trace Equivalence Approaches

The gold standard for privacy in security protocols are the notions of *observational equivalence* and *trace equivalence* (see, e.g., [11] 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 [7], especially when systems have a long-term mutable state [4], but still privacy notions are typically formulated as such an equivalence between two alternative worlds, rather than reachability problem. Interesting in this context is the notion of *diff-equivalence* [5] that is implemented in the most popular verification tools ProVerif and Tamarin: here the processes are parametrized over a binary choice in terms and this gets close to a reachability problem, because the processes are practically in lockstep. This is, however, so restrictive that only a very limited class of privacy properties can be considered.

(α, β)*-privacy* was introduced in [27] as a more declarative way to formalize and reason about privacy than the indistinguishability of two alternatives. There is an underlying notion of equivalence in (α, β)-privacy though: the static equivalence of the frames *concr* and *struct*. These describe not two alternative worlds but rather different levels of knowledge of the intruder: the concrete messages and the structural knowledge. However, (α, β)-privacy is a static notion, describing a fixed state of the world, and does not reason about the interaction of the intruder with his environment. We showed in this paper how to lift (α, β)-privacy to full-fledged transition systems, and have thus re-cast privacy as a reachability problem without the limitations that come, e.g., in diff-equivalence.

In a nutshell, all that can be expressed with trace equivalence, and more, can be expressed with (α, β)-privacy. We now give a formal comparison[5].

[5]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 restric-

5.1 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 in the first place; 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:[6] 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):

```
$\diamond\ z \in \{1, \ldots, n\}$.
if $(z \doteq 1)$ then $T_1$.
else if $(z \doteq 2)$ then $T_2$.
...
else if $(z \doteq n)$ then $T_n$
```

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 (and thus learning the value of z does not exclude any models of 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.

5.2 Restrictions

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 $\star\ x \in D$;[7] 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

tions, that are somewhat artificial, but that at least give an idea of how the two approaches relate.

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

[7]We also ignore the probabilistic aspect in this section as it is not part of the common trace equivalence notions.

- for a transaction that receives a message, which of all available messages is received.

Thus, for a given choice of transactions to perform and recipes of the intruder to send for the inputs, the α- and β-variables are the only non-determinism.

(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 the intruder may not directly infer anything about the conditionals from the number of messages sent or received by a transaction; of course, the intruder may still be able to conclude from the observed output which path was taken by a transaction. Hence, this is not a significant restriction in practice. In a trace of k steps, the intruder has to give k inputs and receives k outputs. Thus, in each reached state after k steps, every *struct* frame and the *concr* frame have the same domain $\{l_1, \ldots, l_k\}$, where each l_i labels the output of the i-th transaction. Similarly, the input from the intruder to each transaction is thus simply a recipe r_i which uses only labels $\{l_0, \ldots, l_{i-1}\}$, i.e., all outputs received so far.

Definition 12. *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 $\{l_1, \ldots, l_{i-1}\}$, and*
- *$(\mathcal{S}, \mathcal{P})$ 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 Section 3.2.)*

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 specification of (α, β)-privacy transactions. 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 $(\mathcal{P}_i, \phi_i, \mathit{struct}_i)$ 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 $(\mathcal{P}_i, \phi_i, \mathit{struct}_i)$) at a given point. Now, there is however no possibility $(\mathcal{P}_i, \phi_i, \mathit{struct}_i)$ 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 can now relate (α, β)-privacy in the binary case with trace equivalence (we first prove Theorem 4 as it will come in handy to prove Theorem 3):

Theorem 3. *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 $\mathit{Spec}_0 \approx \mathit{Spec}_1$.*

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

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

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

We can finally prove Theorem 3:

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 trivially no violation of (α, β)-privacy or trace distinction possible. Thus, for the rest of this theorem, we consider only a symbolic trace tr that includes the special transaction T_{bin}. Observe 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 4, 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 now, for the sake of contradiction, that (α, β)-privacy is violated in $Spec$. Then, by Theorem 4, 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. $\square$

6 Comparison with information flow

For many applications, it is interesting to take into account probabilities. We see no obvious way to reason with them in equivalence-based specifications, but the fact that every state in dynamic possibilistic (α, β)-privacy represents a single reality allows us to make a declarative extension that integrates non-determinism and probabilistic aspects. In fact, the two theorems on extensible specification allow for adding probabilities and background knowledge to an otherwise possibilistic specification without further proofs.

Our work is also related to privacy approaches based on *non-interference* and *information flow* [17, 26, 32]. In fact, one may wonder if dynamic probabilistic (α, β)-privacy is not simply information flow in disguise. That is, one may think that α is basically akin to high variables, and β is akin to low variables. While there are similarities and parallels, there are also key conceptual differences. Most importantly, the β information is "low" in quite a different sense: it is regarded as technical information, but without stipulating whether it is privileged. For instance, β may contain cryptographic keys and nonces, and some nonces (and even some keys) may be obtained by the intruder without violating dynamic probabilistic (α, β)-privacy; in fact, knowing them does not in itself constitute a violation. However, sometimes even just knowing that two low-level messages are identical can be sufficient to deduce a violation, because it rules out some model of α.

Our explicit focus on probabilities bears similarities with approaches in *quantitative information flow* [3, 8, 14, 19, 22, 24, 30], as well as *differential privacy* [16, 34], *k-anonymity* [31], *l-diversity* [25], and *t-closeness* [23], which aim at quantifying privacy to capture leaks or information disclosure in a system. Differential privacy, in particular, is concerned with statistical queries to databases, i.e., how do you reveal accurate statistics about a set of individuals while preserving their privacy? In this approach, private and public information are not completely disjoint, e.g., one might want to reveal the average age of the participants in a survey, but if a new participant is added after the reveal of the average, then one can learn their age.

Differential privacy (and related approaches) and approaches for protocol security and privacy (like ours) are typically seen as two different disciplines that look at privacy in distinct, particular ways. We regard our work also as an attempt to start bridging this gap. While our approach is rooted in protocol verification, we have tried to import here notions that have been successful on the other side of the gap.

Still, it is important to note an interesting relationship between our work in this paper and information flow. This comparison underlines the fact that in general, many information flow approaches are a form of static analysis that over-approximates and classifies many potential problems as violations, whereas (α, β)-privacy is usually more precise. The comparison with information flow is a bit more difficult, since it is farther away from (α, β)-privacy than trace equivalence, and thus we give a less formal comparison.

An intuition is that the semantics of transactions in (α, β)-privacy reflects what would be called explicit and implicit flows in information flow approaches: the messages sent out by a transaction are visible to the intruder. This may give rise to an explicit flow, for instance, if the sent message contains directly the value of an α-variable. It may give rise to an implicit flow, if the sent message depends on a condition and the intruder may thus be able to determine the value of the condition. In fact, an important part of the semantics is to formalize what the intruder knows about the shape of the message (i.e., *struct*) depending on the different outcomes of conditions taken so far.

For example, consider the following simple program:

```
if (x>10){
  y:=y*y;
}
else{
  y:=y*x;
}
```

Suppose x is of level H (high) and y of level L (low). Then, we obviously have two violations of information flow. We could model this also in (α, β)-privacy, but since we are modifying variables, we would have to use here memory cells Xcell and Ycell. We can initialize these memory cells with an arbitrary value from, say, the set of integers. The privacy condition is that the intruder cannot find out anything about the initial value of x. However, he should be able to see the value of y before and after the transaction. The initialization can be formalized as follows (where init is a boolean memory cell, initialized to false):

```
if (init $\doteq$ false) then
    init := true.
    $\star\ X \in \{0, \dots, \mathsf{MAXINT}\}$.
    $\diamond\ Y \in \{0, \dots, \mathsf{MAXINT}\}$.
    Xcell := $X$.
    Ycell := $Y$.
    snd(Ycell)
```

and the actual program is:

```
if (init $\doteq$ true) then
   if (Xcell $> 10$) then
      Ycell := Ycell $*$ Ycell.
   else
      Ycell := Ycell $*$ Xcell.
   snd(Ycell)
```

A possible trace is now that the intruder observes an initialization transaction with the concrete value 1 (for y) and then the same value again in a program transaction. Then, he can exclude the second branch (because y would have to be at least 11 for that), and thus knows $X > 10$, violating the (α, β)-privacy privacy goal.

Here we can however observe five major differences between (α, β)-privacy and information flow. The first major difference is that (α, β)-privacy will not trigger if there is no actual information flow that can be observed. In the above example, for instance, if y gets initialized with 0, then the result is not telling anything about x, and hence in that state, (α, β)-privacy holds. Thus, (α, β)-privacy does not share the false positives that can arise from the over-approximation of static information flow analysis. While it is nice to avoid false positives, the simplicity and efficiency of static analysis naturally raise the question whether this can also be an effective technique to analyze (α, β)-privacy problems and whether we can over-approximate in a similar way—a problem we must leave open at this point.

A second major difference is that (α, β)-privacy allows for sending also classified values on a public channel when they are encrypted such that the intruder cannot decrypt them. While one can of course define an appropriate declassification to allow also for this in information flow, this is not suitable for the analysis of privacy in protocols, and one would need here (α, β)-privacy or trace equivalence approaches: this is because one needs to analyze in general whether the employed encryption regime indeed excludes attacks, so that the intruder cannot obtain any information, for instance by comparison of encrypted messages (or their reuse).

For a third major difference, recall that (α, β)-privacy by default does not hide which transaction is taken, but this information can be hidden from the intruder (cf. the comparison with trace equivalence in Section 5). It is important to note that the *number* of transactions taken *cannot* be hidden from the intruder in (α, β)-privacy. For instance, this program

```
while (x>0){
  x:=x-1;
  y:=y*x;
}
```

would be fine in classical information flow if x and y are both H (or both L), as all implicit and explicit flows are allowed. However, the number of transactions that can be taken now reveals information about x. One may consider this as a (rather blunt) timing leak. In fact, we cannot directly implement this program as one transaction in (α, β)-privacy. Instead we would have to also model a program counter—as a β-

variable that is *not* given to the intruder:

```
if $(\mathsf{PC} \doteq \mathsf{start} \wedge \mathsf{Xcell} > 0)$ then
    $\mathsf{Xcell} := \mathsf{Xcell} - 1.$
    $\mathsf{Ycell} := \mathsf{Ycell} * \mathsf{Xcell}.$
else if $(\mathsf{PC} \doteq \mathsf{start} \wedge \mathsf{Xcell} \doteq 0)$ then
    $\mathsf{PC} \doteq \mathsf{end}$
```

Now after only two such transactions, the intruder can infer that the initial value of x is greater then 0 and thus violate privacy.

A fourth difference between the approaches concerns declassifications. While in information flow one would declassify a particular value, (α, β)-privacy in general allows one to rather declassify a statement, and declassification is closed under logical deduction. For instance, when we release the result of an election, we do not just declassify the number of votes received by each candidate (that was computed based on the classified votes), but we also release the statement that it is the sum of all accepted votes. Thus, if we have three dishonest voters voting for candidate A, and candidate A received only three votes, then the intruder can infer—and is allowed to infer—that no honest voter voted for candidate A. Similarly, in a unanimous vote, the intruder also learns the value of every single vote, and that is permitted by the goal.

As a final difference, consider the security lattices that can be used for the different levels in information flow. While we, in (α, β)-privacy, basically consider only two levels (i.e., whether the intruder may know or not), this can be used anyway for a more complex lattice by several separate analyses, one for each security level that the intruder may realistically obtain. There is, however, one aspect that (α, β)-privacy cannot handle at all: how information flow can use lattices for integrity analysis, e.g., that information cannot flow from an untrusted variable into a trusted one. Here, (α, β)-privacy can only offer the formulation of the usual authentication goals like injective agreement that are standard in protocol verification.

7 Future Work

Having introduced transition systems, it is natural to consider the automation of our approach. Since it is based on FOL, our formalization is expressive but not decidable, not even semi-decidable, which presents challenges for automation. Still, some fragments of (α, β)-privacy are decidable [27] and are, in some sense, equivalent to the classical static equivalence of frames, so there is hope for automation for fragments of dynamic probabilistic (α, β)-privacy too. We also plan to extend our approach to formalize other quantitative aspects of privacy in addition to probabilities, such as costs and timing leaks as in [15, 21].

References

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

[2] Martín Abadi, Bruno Blanchet, and Cédric Fournet. "The Applied Pi Calculus: Mobile Values, New Names, and Secure Communication". In: *J. ACM* 65.1 (2018), 1:1–1:41.

[3] Mário S. Alvim et al. *The Science of Quantitative Information Flow.* 2020.

[4] Myrto Arapinis et al. "Stateful applied pi calculus: Observational equivalence and labelled bisimilarity". In: *J. Log. Algebraic Methods Program.* 89 (2017), pp. 95–149.

[5] Bruno Blanchet, Martín Abadi, and Cédric Fournet. "Automated verification of selected equivalences for security protocols". In: *J. Log. Algebraic Methods Program.* 75.1 (2008), pp. 3–51.

[6] Mayla Brusò, Konstantinos Chatzikokolakis, and Jerry den Hartog. "Formal Verification of Privacy for RFID Systems". In: *CSF.* 2010, pp. 75–88.

[7] Vincent Cheval, Hubert Comon-Lundh, and Stéphanie Delaune. "A procedure for deciding symbolic equivalence between sets of constraint systems". In: *Inf. Comput.* 255 (2017), pp. 94–125.

[8] David Clark, Sebastian Hunt, and Pasquale Malacaria. "Quantitative Analysis of the Leakage of Confidential Data". In: *ENTCS* 59.3 (2001), pp. 238–251.

[9] Véronique Cortier, Michaël Rusinowitch, and Eugen Zalinescu. "Relating two standard notions of secrecy". In: *Log. Methods Comput. Sci.* 3.3 (2007).

[10] Véronique Cortier and Ben Smyth. "Attacking and Fixing Helios: An Analysis of Ballot Secrecy". In: *CSF.* 2011, pp. 297–311.

[11] Stéphanie Delaune and Lucca Hirschi. "A survey of symbolic methods for establishing equivalence-based properties in cryptographic protocols". In: *JLAMP* 87 (2017), pp. 127–144.

[12] Stéphanie Delaune, Steve Kremer, and Mark Ryan. "Coercion-Resistance and Receipt-Freeness in Electronic Voting". In: *CSFW.* 2006.

[13] Stéphanie Delaune, Mark Ryan, and Ben Smyth. "Automatic Verification of Privacy Properties in the Applied pi Calculus". In: *Trust Management II - Proceedings of IFIPTM 2008: Joint iTrust and PST Conferences on Privacy, Trust Management and Security, June 18-20, 2008, Trondheim, Norway.* Vol. 263. IFIP Advances in Information and Communication Technology. 2008, pp. 263–278.

[14] A. Di Pierro, C. Hankin, and H. Wiklicky. "Approximate Non-Interference". In: *J. Comput. Secur.* 12.1 (2004), pp. 37–81.

[15] A. Di Pierro et al. "Tempus Fugit: How to Plug It". In: *JLAP* 72.2 (2007), pp. 173–190.

[16] Cynthia Dwork. "Differential Privacy: A Survey of Results". In: *TAMC.* LNCS 4978. 2008.

[17] Joseph A. Goguen and José Meseguer. "Security Policies and Security Models". In: *IEEE Symposium on Security and Privacy.* 1982, pp. 11–20.

[18] Sébastien Gondron and Sebastian Mödersheim. "Formalizing and Proving Privacy Properties of Voting Protocols Using Alpha-Beta Privacy". In: *ESORICS.* LNCS 11735. 2019, pp. 535–555.

[19] Damas P. Gruska. “Probabilistic Information Flow Security”. In: *Fundam. Inform.* 85 (2008), pp. 173–187.

[20] T. Hinrichs and M. Genesereth. *Herbrand Logic.* Tech. rep. Stanford University, 2006.

[21] Paul C. Kocher. “Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems”. In: *Advances in Cryptology.* 1996, pp. 104–113.

[22] Ruggero Lanotte, Andrea Maggiolo-Schettini, and Angelo Troina. “Time and Probability-Based Information Flow Analysis”. In: *IEEE Trans. Software Eng.* 36.5 (2010), pp. 719–734.

[23] Ninghui Li, Tiancheng Li, and Suresh Venkatasubramanian. “t-Closeness: Privacy Beyond k-Anonymity and l-Diversity”. In: *ICDE.* 2007, pp. 106–115.

[24] Gavin Lowe. “Quantifying Information Flow”. In: *CSF.* 2002, pp. 18–31.

[25] Ashwin Machanavajjhala et al. “L-diversity: Privacy beyond k-anonymity”. In: *ACM Trans. Knowl. Discov. Data* 1 (2007).

[26] Heiko Mantel, David Sands, and Henning Sudbrock. “Assumptions and Guarantees for Compositional Noninterference”. In: *CSF.* 2011, pp. 218–232.

[27] Sebastian Mödersheim and Luca Viganò. “Alpha-Beta Privacy”. In: *ACM Trans. Priv. Secur.* 22.1 (2019), 7:1–7:35.

[28] M. Ohkubo, K. Suzuki, and S. Kinoshita. “Cryptographic approach to "privacy-friendly" tags”. In: *RFID Privacy Workshop* (2003).

[29] Steve Selvin. “On the Monty Hall problem (letter to the editor)”. In: *The American Statistician* 29.3 (1975).

[30] Geoffrey Smith. “On the Foundations of Quantitative Information Flow”. In: *FoSSaCS.* LNCS 5504. 2009, pp. 288–302.

[31] Latanya Sweeney. “k-Anonymity: A Model for Protecting Privacy”. In: *Int. J. Uncertain. Fuzziness Knowl. Based Syst.* 10.5 (2002), pp. 557–570.

[32] Henk C. A. van Tilborg and Sushil Jajodia, eds. *Encyclopedia of Cryptography and Security, 2nd Ed.* 2011.

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

[34] Jiannan Yang, Yongzhi Cao, and Hanpin Wang. “Differential privacy in probabilistic systems”. In: *Inf. Comput.* 254 (2017), pp. 84–104.

A DP-3T

As a concrete, and topical example, let us consider the decentralized, privacy-preserving proximity tracing system DP-3T [1], 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.1 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,
- $\mathit{Day} = \{0, \dots, \mathsf{D} - 1\}$ identifies days,
- $\mathit{Period} = \{0, \dots, \mathsf{P} - 1\}$ 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, and
- EphID is the sort of ephemeral identities (changing, e.g., every 15 minutes),

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:

- $\mathsf{sk}_0[\cdot]\colon \mathsf{Agent} \to \mathit{SK}$ 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 $a \neq b$,
- $h[\cdot]\colon \mathit{SK} \to \mathit{SK}$ 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 $\mathsf{sk}_0[a] \in \mathit{SK}$ such that $h^i[\mathsf{sk}_0[a]] \neq h^j[\mathsf{sk}_0[b]]$ for any $a, b \in \mathsf{Agent}$, $i, j \in \mathit{Day}$ with $(a, i) \neq (b, j)$: every daily identity of an agent is unique[8],
- $\mathit{prg}[\cdot,\cdot]\colon \mathit{SK} \times \mathit{Period} \to \mathit{EphID}$ models a pseudo-random number generator to generate the ephemeral identities. We assume prg is injective on the domain $\mathit{SK} \times \mathit{Period}$, so that there is also no collision between the ephemeral identities of any agents (with respect to any timepoints).
- $\mathit{pwnr}[\cdot]\colon \mathit{EphID} \to \mathsf{Agent}$ relates, in our model, an ephemeral ID to its actual owner, i.e., for $e = \mathit{prg}[h^i[\mathsf{sk}(a)], j]$, we have $\mathit{pwnr}[e] = a$,
- $\mathit{dayof}[\cdot]\colon \mathit{EphID} \to \mathit{Day}$ tells the day an ephemeral ID is issued, and
- $\mathit{sick} \subseteq \mathit{EphID} \times \mathit{Day}$ is a relation where $\mathit{sick}(e, d)$ means that the agent identified by e has declared sick on day d. In contrast, $\mathit{dayof}[e]$ is the day when e was used.

The functions h and prg are cryptographic functions, and sk_0 is a cryptographic setup. We regard them as technical/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 function pwnr 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:

[8] SK is a finite set, so h must have collisions, and we merely exclude that these collisions are relevant to the protocol. We abstract from cryptography and thus from the negligible probability of collisions between agents.

- $\mathsf{sk}_l(A\colon \mathsf{Agent}) \coloneqq \mathsf{sk}_0[A]$ 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),
- $\mathsf{sk}(A\colon \mathsf{Agent}) \coloneqq h^l[\mathsf{sk}_0[A]]$. 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,
- $\mathit{today}() \coloneqq l$ is the current day counter (it is the same for all agents),
- $\mathit{period}() \coloneqq 0$, where 0 identifies the first period of a day,
- $\mathit{ephid}(A\colon \mathsf{Agent}) \coloneqq \mathit{prg}[\mathsf{sk}(A), \mathit{period}()]$ is the current ephemeral ID, and
- $\mathit{isSick}(A\colon \mathsf{Agent}) \coloneqq \mathsf{false}$ 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 Figure 5. 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 $x \in \mathsf{Agent}$. Here, our model ignores the details of how two agents' phones actually exchange IDs, which can cause also several problems [33]. Finally, 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 [33]. Our model thus focuses on the core privacy question that arises, even if all exchange protocols work perfectly.

A.2 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 $e_0 = \mathit{prg}[\mathsf{sk}_l, 0]$ and $e_1 = \mathit{prg}[\mathsf{sk}_l, 1]$ 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:

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

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:

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

Intruder deductions:

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

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

Figure 5: A model of DP-3T (with insufficient α).

Using $\phi_\sim$:
$struct[prg[m_3, 0]] = struct[m_1]$
$struct[prg[m_3, 1]] = struct[m_2]$
$prg[h^1[\mathsf{sk}_1[x_3]], 0] = prg[h^l[\mathsf{sk}_0[x_1]], 0]$
$prg[h^l[\mathsf{sk}_0[x_3]], 1] = prg[h^l[\mathsf{sk}_0[x_2]], 1]$

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 possibilistic (α, β)-privacy is indeed violated. Note that we do not find out that $x_1 = a$, but we have linkability of pseudonyms of sick persons.

A.3 The Actual Privacy Guarantee

Actually, 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 practically

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

Figure 6: Axioms for DP3T

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, "admitting" in α this additional information, what we could find out in the concrete scenario before, namely that $x_1 = x_2 = x_3$, no longer count as an attacks, as we explicitly declare that we want to release this information.

However, this extended α *still* does not cover all the information we release. For instance, if the two agents $x_1 = a$ and $x_2 = b$ have released ephemeral IDs e_a and e_b, respectively, and a has declared sick, then we can still observe that $x_1 \neq x_2$ 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 $x_1 \neq x_2$, 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.

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 6, and we obtained them from failed attempts of proving dynamic possibilistic (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 $\mathsf{P} = |\mathit{Period}|$ 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 6 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: $x \in \mathsf{Agent} \wedge \mathit{pwnr}[e] = x \wedge \mathit{dayof}[e] = d$, where $e \in \mathit{EphID}$, $d \in \mathit{Day}$, and x is a variable that occurs nowhere else in α, and
- from Agent Sick: $\bigwedge_{e \in E} \mathit{pwnr}[e] = x \wedge \mathit{sick}(e, d_r)$, where E is a set of ephemeral IDs that are released on reporting day d_r. Amongst 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 $|\mathit{Period}| \times l$, 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-determinstic *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 $\mathcal{I}(\mathit{pwnr}[e]) = a$.*
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 $\mathcal{I}(\mathit{pwnr}[e]) = a$ has been determined already, then $\mathcal{I}(x) = a$.*
   (b) *If $\mathcal{I}(\mathit{pwnr}[e])$ 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' such that α contains $\mathit{sick}(e, d') \wedge \mathit{pwnr}[e] = x' \wedge \mathit{dayof}[e] = d_0$ and $d_0 \leq d$. Further, let Agent_e denote all the agents a for which $\mathcal{I}(\mathit{pwnr}[e]) = a$, and $\mathcal{I}(\mathit{dayof}[e]) = d$ 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 $\mathcal{I}(\mathit{pwnr}[e]) = a$.*
3. *All remaining aspects of $\mathcal{I}$ are actually irrelevant (i.e., $\mathcal{I}(\mathit{pwnr}[e])$ 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 $\mathcal{I} \models \mathit{pwnr}[e_i] \doteq x_i \doteq x_j \doteq \mathit{pwnr}[e_j]$ 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 $\mathit{dayof}[e] = d$ 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., $\mathit{dayof}[e]$ 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. $\square$

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

Theorem 5. *DP-3T with the extended α specification given in this section satisfies dynamic possibilistic (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 $\mathcal{I}(\mathit{prg}[\mathsf{sk}, i]) = \mathcal{I}_r(\mathit{prg}[\mathsf{sk}, i])$ for every period $i \in \mathit{Period}$ 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 $\mathsf{sk}(a)$, $\mathit{period}()$ and $\mathit{today}()$ 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 $\mathcal{I}(\mathit{prg}[\mathsf{sk}, i]) = \mathcal{I}_r(\mathit{prg}[\mathsf{sk}, i])$ previously, and $\mathcal{I}_r(\mathit{prg}[\mathsf{sk}, i]) = e$ 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 $\mathcal{I}(\mathit{prg}[\mathsf{sk}, i])$ 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 $\mathit{prg}[\mathsf{sk}, i]$ has not yet been assigned a different interpretation yet. The formula β now contains (for an appropriate label m): $\mathit{concr}[m] = e \wedge \mathit{struct}[m] = \mathit{prg}[h^d[\mathsf{sk}_0[x]], i]$. 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 $\mathcal{I} \models \mathit{concr}[m] = \mathit{struct}[m]$, 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 $\mathsf{sk}_l(a)$, $\mathit{period}()$, and $\mathit{today}()$ in the current virtual state. The formula β now contains $\mathit{concr}[m] = \mathsf{sk}_l$ and $\mathit{struct}[m] = h^{d-l}[\mathsf{sk}_0[x]]$. Observe also here that we have $\mathcal{I} \models \mathit{concr}[m] = \mathit{struct}[m]$ because $\mathsf{sk}_l(x)$ is x's key from l days ago. $\square$

B Voting Protocols

The authors of [18] have modeled privacy goals for voting protocols with static possibilistic (α, β)-privacy. To illustrate the expressive power of our approach, we show how to adapt and formalize these privacy goals, namely voting privacy and receipt-freeness, with our dynamic extension of (α, β)-privacy. We show the formalization of these goals with a simple voting protocol. Indeed, while voting protocols can use more complicated cryptographic primitives, such as homomorphic encryption or blind signature, the principles behind the formalization of the goals remain similar. The state we discuss at the end of this section is rather complex, but the specification of the privacy goals is actually simple and intuitive with (α, β)-privacy, and we obtain a reachability problem out of this.

In our example, we consider a finite set of n voters $\mathsf{Voters} = \{x_1, \ldots, x_n\}$, two candidates candA and candB, and a trusted third-party a that acts both as the administrator of the election and the counter. Let $\mathsf{scrypt}/2$, $\mathsf{dscrypt}/2$ and $\mathsf{sk}/2$ be public functions, modeling symmetric encryption with a secret key. We consider the algebraic equation $\mathsf{dscrypt}(\mathsf{sk}(a, b), \mathsf{scrypt}(\mathsf{sk}(a, b), m)) \approx m$. Let $\mathsf{ballot}/2$ and $\mathsf{open}/1$ be two public functions modeling the ballot as a message format with the algebraic function $\mathsf{open}(\mathsf{ballot}(t_1, t_2)) \approx t_1, t_2$. We assume that each voter x_i shares an encryption key with the administrator, $\mathsf{sk}(x_i, a)$. We also assume that the intruder knows the key of dishonest voters, i.e., if x_i is dishonest, the intruder knows $\mathsf{sk}(x_i, a)$. We consider four families of memory cells: one for the status of the election $\mathsf{status}(\cdot)$, one for the status of a voter $\mathsf{voted}(\cdot)$, one for recording that the administrator accepts the vote of a voter $\mathsf{cast}(\cdot)$, and one for the result of a candidate $\mathsf{result}(\cdot)$. The initial values are voting, no, no and 0, respectively. Each voter X owns their own cell $\mathsf{voted}(X)$ and the administrator owns the three other entire families, i.e., the election public information. The administrator can update the status of the election to over when the tallying phase starts. A voter X updates their status $\mathsf{voted}(X)$ to yes when they vote, and the administrator updates $\mathsf{cast}(X)$ to yes when he accepts the vote of a voter X. Finally, the administrator updates the result of a candidate, say candA, with the memory cell $\mathsf{result}(\mathsf{candA})$, each time a valid vote for this candidate is counted. Moreover, let $v/1$ and $c/1$ be two interpreted functions that model respectively the voting function and the check for counted votes. Finally, we allow for dishonest voters, and we define the predicate $\mathsf{dishonest}$. For every dishonest $X \in \mathsf{Voters}$, $\mathsf{dishonest}(X)$ holds, and, conversely, for every honest $X \in \mathsf{Voters}$, $\neg\mathsf{dishonest}(X)$ holds.

The election is divided in two phases. The voting phase is modeled by the *Cast* and *Admin* processes in Figure 7. During a *Cast* process transaction, a honest voter X can choose their vote V. If the voter did not vote already, a new random value R is generated. Their status $\mathsf{voted}(X)$ is updated to yes. We publish in γ the true value of their vote, i.e., $v[\mathcal{I}(X)] \doteq \mathcal{I}(V)$: this is used later in the tallying phase to formalize the core of the privacy goal. Finally, the voter sends their vote to the administrator that they encrypt with their shared secret key. Note that this transaction can only be taken for honest voters. The intruder can send any messages that he wants, but ultimately, for the dishonest voters that he controls, he has to produce a vote that the administrator later accepts if he wants the vote to be counted.

During a *Admin* process transaction, every time the administrator receives a ballot from a voter X, they first check that the sender is a legit voter, and they also check that the legit voter did not cast a vote already by checking $\mathsf{cast}(X)$. If these requirements are met, the administrator updates $\mathsf{cast}(X)$ to yes. Besides, if the voter is dishonest, i.e., if $\mathsf{dishonest}(X)$ holds, the vote is also disclosed in γ and in α. This does not mean that the intruder necessarily knows the value of the vote, but that it should not count as an attack if he learns it. This can be seen as a sort of declassification: the relation between a dishonest voter and their vote is not a secret. This is important in systems like an early version of Helios, where an intruder can cast a copy of another voter's vote as his own vote. It would be obscuring the attack from our analysis, if we simply gave to the intruder the information what his vote is; leaving this information classified would lead to a false positive in general (because typically the intruder knows what he voted). (α, β)-privacy thus allows us to stay clear of both problems by just declassifying the relation between dishonest voter and their vote.

At the beginning of a *Counter* process transaction, the administrator who is also acting as the counter can switch the status of the election to over. The administrator

Cast

```
if status(·) $\doteq$ voting then
  $\star$ $X \in$ Voters.
  if ($\neg$dishonest($X$)) then
    $\star$ $V \in \{$candA, candB$\}$.
    $s :=$ voted($X$).
    if ($s \doteq$ no) then
      $\nu R.$ voted($X$) := yes.
      $\diamond$ $v[\mathcal{I}(X)] \doteq \mathcal{I}(V)$.
      snd(scrypt(sk($X, a$), ballot($R, V$)))
```

Admin

```
if status(·) $\doteq$ voting then
  rcv(scrypt(sk($X, a$), ballot($R, V$))).
  $s :=$ cast($X$).
  if ($X \in$ Voters $\wedge$ $s \doteq$ no) then
    cast($X$) := yes.
    if (dishonest($X$)) then
      $\diamond$ $v[\mathcal{I}(X)] \doteq \mathcal{I}(V)$.
      $\star$ $v[\mathcal{I}(X)] \doteq \mathcal{I}(V)$.
```

Counter

```
status(·) := over.
for$X$ : Voters
  $s :=$ cast($X$).
  if ($s \doteq$ yes $\wedge$ $v[\mathcal{I}(X)] \doteq$ candA) then
    result := result(candA).
    result(candA) := result + 1.
    $\diamond$ $c[\mathcal{I}(X)] \doteq$ true.
  if ($s \doteq$ yes $\wedge$ $v[\mathcal{I}(X)] \doteq$ candB) then
    result := result(candB).
    result(candB) := result + 1.
    $\diamond$ $c[\mathcal{I}(X)] \doteq$ true.
result$_A$ := result(candA).
result$_B$ := result(candB).
if (result$_A \not\doteq$ result_candA $\vee$ result$_B \not\doteq$ result_candB) then
  attack.
else
  $\star$ result_candA $\doteq \mathcal{I}$(result_candA) $\wedge$ result_candB $\doteq \mathcal{I}$(result_candB).
```

$$\text{result\_candA} = \Sigma_{X \in \text{Voters} \wedge v[X] \doteq \text{candA} \wedge c[X] \doteq \text{yes}} 1$$
$$\text{result\_candB} = \Sigma_{X \in \text{Voters} \wedge v[X] \doteq \text{candB} \wedge c[X] \doteq \text{yes}} 1$$

Figure 7: Voting Protocol

is going through all the voters. We use a for construct as a syntactic sugar. We need to unroll this loop, i.e., repeat the body for each voter. This notational sugar allows us to keep our formalization parametrized over an arbitrary number of voters, while a concrete specification that results from unrolling this loop has the number of voters fixed. This is because we do not wish to formalize an unbounded number of steps in a single step, which would have undesirable consequences on the semantics. For each voter, the administrator checks whether they cast a vote that has been accepted. If this is the case, a distinction is made following the value of the interpreted function $v[\mathcal{I}(X)]$ stored in γ. This does not represent that the administrator knows the value of the vote, but that we make a case distinction depending of what the truth γ is. Depending on $v[\mathcal{I}(X)]$ being candA or candB, the administrator updates the result memory cell for the corresponding candidate. Finally, the administrator sets the check for counted vote of the voter to true. Once this is done for all voters, the administrator checks that the result is correct. We encode this correctness property for candA for instance by $\mathsf{result}_A \not\doteq \mathsf{result\_candA}$. $\mathsf{result\_candA}$ is computed directly from the information published in γ, i.e., $\mathsf{result\_candA} = \sum_{X \in \mathsf{Voters} \land v[X] \doteq \mathsf{candA} \land c[X] \doteq \mathsf{true}} 1$, whereas result_A is the actual computation done by the administrator. Again, this does not mean that the administrator knows the value of each individual vote, but this means that we require that the result the counter computes corresponds to the truth. If this is not the case, there is an attack, i.e., it triggers a violation of (α, β)-privacy, since the basic correctness of the protocol is shown violated if this branched is reached. Otherwise, we can publish the result in α. We would like to emphasize the key role of the information published in γ through interpreted functions to express the privacy goals of a voting protocol (similarly to what was done in the static approach in [18]).

To give another example on how dynamic possibilistic (α, β)-privacy works, we show how to reach a state that we call "final", i.e., a state where the result has been printed and no new transactions can be taken (see Figure 8). For this example, we consider that there are three voters, i.e., $\mathsf{Voters} = \{x_1, x_2, x_3\}$. x_1 and x_2 are honest voters, and x_3 is dishonest, i.e., $\mathsf{dishonest}(x_3)$ holds. For the sake of simplicity, we consider that first the three voters cast their vote, and then only that the administrator registers their votes.

This means that a *Cast* process transaction is taken two times for the honest voters. For the two instantiations (lines $1, 2$ in Figure 8), public information about the voter is released in α, the intruder can observe the exchange of messages with the administrator, the true value of the votes is released in γ, and the status of each of the voters is updated to yes. Note that the intruder can send whatever messages that he can compose, especially he is able to send a valid vote to the administrator since he knows $\mathsf{sk}(x_3, a)$.

Then, a process *Admin* transaction is also taken three times. This is reflected for the three instantiations (line $3, 4, 5$ in Figure 8) by the update of $\mathsf{cast}(X)$ to yes. Note that for the dishonest voter x_3, the true result of the vote is released in both γ and α, since this is a vote that was produced by the intruder. Again, this does not mean that the intruder learns the value of the vote at this point, but that it will not count as an attack if he does.

Finally, the only transaction still possible is the instantiation of a *Counter* process. This means the administrator is going through the votes and updates the result. Every time the administrator counts the vote of a voter, it is released in γ that the vote has been counted. At the end of the process, the result of the election is released in α.

Before concluding on the security of this "final" state, we need to recapitulate the privacy goals that we expressed. Let us have a look at the information we have

	α	β	γ	δ
1	$X_1 \in \mathsf{Voters}$ $\wedge\ V_1 \in \{\mathsf{candA}, \mathsf{candB}\}$	$concr[l_1] = \mathsf{scrypt}(\mathsf{sk}(x_1, a), \mathsf{ballot}(R_1, \mathsf{candA}))$ $\wedge\ struct[l_1] = \mathsf{scrypt}(\mathsf{sk}(X_1, a), \mathsf{ballot}(R_1, V_1))$	$X_1 \doteq x_1$ $\wedge\ V_1 \doteq \mathsf{candA}$ $\wedge\ v[x_1] \doteq \mathsf{candA}$	$\mathsf{voted}(X_1) := \mathsf{yes}$ if ϕ_1
2	$X_2 \in \mathsf{Voters}$ $\wedge\ V_2 \in \{\mathsf{candA}, \mathsf{candB}\}$	$concr[l_2] = \mathsf{scrypt}(\mathsf{sk}(x_2, a), \mathsf{ballot}(R_2, \mathsf{candB}))$ $\wedge\ struct[l_2] = \mathsf{scrypt}(\mathsf{sk}(X_2, a), \mathsf{ballot}(R_2, V_2))$	$X_2 \doteq x_2$ $\wedge\ V_2 \doteq \mathsf{candB}$ $\wedge\ v[x_2] \doteq \mathsf{candB}$	$\mathsf{voted}(X_2) := \mathsf{yes}$ if $\phi_1 \wedge \phi_2$
3				$\mathsf{cast}(X_1) := \mathsf{yes}$ if ϕ_3
4				$\mathsf{cast}(X_2) := \mathsf{yes}$ if $\phi_3 \wedge \phi_4$
5	$v[x_3] \doteq \mathsf{candA}$		$v[x_3] \doteq \mathsf{candA}$	$\mathsf{cast}(X_3) := \mathsf{yes}$ if $\phi_3 \wedge \phi_4 \wedge \phi_5$
6	$\mathsf{result\_candA} \doteq 2$ $\wedge\ \mathsf{result\_candB} \doteq 1$		$c[x_1] \doteq \mathsf{true}$ $\wedge\ c[x_2] \doteq \mathsf{true}$ $\wedge\ c[x_3] \doteq \mathsf{true}$	$\mathsf{status}(\cdot) := \mathsf{over}$ if ϕ_6 $\mathsf{result}(\mathsf{candA}) := 2$ if $\phi_6 \wedge \phi_{vote}$ $\mathsf{result}(\mathsf{candB}) := 1$ if $\phi_6 \wedge \phi_{vote}$

$\phi_1 \equiv \mathsf{status}(\cdot) \doteq \mathsf{voting} \wedge s_1 \doteq \mathsf{no} \wedge \neg\mathsf{dishonest}(X_1)$ $\qquad$ $\phi_2 \equiv \mathsf{status}(\cdot) \doteq \mathsf{voting} \wedge s_2 \doteq \mathsf{no} \wedge \neg\mathsf{dishonest}(X_2)$

$\phi_3 \equiv \phi_1 \wedge \phi_2 \wedge \phi_3 \wedge X_1 \in \mathsf{Voters} \wedge \mathsf{status}(\cdot) \doteq \mathsf{voting} \wedge s_3 \doteq \mathsf{no}$

$\phi_4 \equiv X_2 \in \mathsf{Voters} \wedge \mathsf{status}(\cdot) \doteq \mathsf{voting} \wedge s_4 \doteq \mathsf{no}$ $\qquad$ $\phi_5 \equiv X_3 \in \mathsf{Voters} \wedge \mathsf{status}(\cdot) \doteq \mathsf{voting} \wedge s_5 \doteq \mathsf{no}$

$$\phi_6 \equiv \begin{array}{l} \phi_3 \wedge \phi_4 \wedge \phi_5 \wedge s_6 \doteq \mathsf{yes} \\ \wedge\ s_8 \doteq \mathsf{yes} \wedge s_9 \doteq \mathsf{yes} \end{array} \qquad \phi_{vote} \equiv \begin{array}{l} (v[x_1] \doteq \mathsf{candA} \wedge v[x_2] \doteq \mathsf{candB} \wedge v[x_3] \doteq \mathsf{candA}) \\ \vee\ (v[x_1] \doteq \mathsf{candB} \wedge v[x_2] \doteq \mathsf{candA} \wedge v[x_3] \doteq \mathsf{candA}) \end{array}$$

Figure 8: Execution of the voting protocol

intentionally released, i.e., the formula α:

$$\begin{aligned}\alpha \equiv\ & X_1 \in \mathsf{Voters} \wedge V_1 \in \{\mathsf{candA}, \mathsf{candB}\}\\ & \wedge\ X_2 \in \mathsf{Voters} \wedge V_2 \in \{\mathsf{candA}, \mathsf{candB}\}\\ & \wedge\ v[x_3] \doteq \mathsf{candA}\\ & \wedge\ \mathsf{result\_candA} \doteq 2 \wedge \mathsf{result\_candB} \doteq 1\end{aligned}$$

α has just two models (before the release of the results, it has four models). Also, observe that the intruder can now deduce that V_1 and V_2 are different, and that is a legal consequence of α. This is something that the cardinality of the election allows, and the best protocol cannot prevent it. Let us have now a look on the technical information: β is the conjunction of ϕ_{gen}, ϕ_{hom}, $\phi_{\sim}$ (see preliminaries) and the following frames *concr* and *struct* (note that x_i are concrete voters and that X_i are the privacy variables picked during transitions):

$$\begin{aligned}\mathit{concr} &= \{\!|\ l_1 \mapsto \mathsf{scrypt}(\mathsf{sk}(x_1, a), \mathsf{ballot}(R_1, \mathsf{candA})),\\ &\qquad l_2 \mapsto \mathsf{scrypt}(\mathsf{sk}(x_2, a), \mathsf{ballot}(R_2, \mathsf{candB}))\,|\!\}\\ \mathit{struct} &= \{\!|\ l_1 \mapsto \mathsf{scrypt}(\mathsf{sk}(X_1, a), \mathsf{ballot}(R_1, V_1)),\\ &\qquad l_2 \mapsto \mathsf{scrypt}(\mathsf{sk}(X_2, a), \mathsf{ballot}(R_2, V_2))\,|\!\}\end{aligned}$$

The payload formula α specifies that the intruder can learn that the three voters voted (he can indeed observe that there are three votes on the bulletin board) and the result of the election. As explained, he can also learn the true vote of voter x_3 (in our specific case, he knows it since he knows $\mathsf{sk}(x_3, a)$). But he should not learn more. The technical information β takes into account the messages that the intruder observed. In any instantiations of the variables that is compatible with α, *concr* and *struct* are statically equivalent, since the intruder cannot decrypt anything else that the vote of the dishonest voter (or compose and check other messages). Thus, all the models of α can be extended to models of β, and this simple voting protocol ensures voting privacy in its "final" state.

Now let us say that we want this protocol to ensure stronger privacy goals, such as *receipt freeness*. The authors of [18] used the following definition: "no voter has a way to prove how they voted". The property is formalized with respect to a specific voter, say x_1, that the intruder is trying to influence. The question is whether voter x_1 can *prove* to the intruder how they voted by a kind of "receipt". The idea of [18] is to force the coerced voter to reveal their *entire knowledge*. But the voter can lie and give to the intruder anything that they can construct from their own knowledge, as long as their *story* is consistent with what the intruder already knows, e.g., from observing the messages exchanged between the voters and the administrator. Similarly to the frames *concr* and *struct* that represent the knowledge of the intruder, we reason about the frames concr_{x_1} and struct_{x_1} that represent the knowledge of voter x_1. The core idea is then that what voter x_1 can lie about is the content of concr_{x_1}. We augment β by the following axiom ϕ_{lie}, that is updated for every transition as $\phi_{\sim}$ for instance, and where $\{d_1, \ldots, d_l\}$ is the domain of the frames concr_{x_1} and struct_{x_1}:

$$\begin{aligned}\phi_{lie} \equiv\ & \mathit{struct}[d_1] = \mathit{struct}_{x_1}[d_1] \wedge \ldots \wedge\ \mathit{struct}[d_l] = \mathit{struct}_{x_1}[d_l]\\ & \wedge\ \exists s_1, \ldots, s_l.\ \mathit{gen}_{D_{x_1}}(s_1)\ \wedge\ \mathit{gen}_{D_{x_1}}(s_l).\\ & (\mathit{concr}[d_1] = \mathit{concr}_{x_1}[s_1] \wedge \ldots \wedge\ \mathit{concr}[d_l] = \mathit{concr}_{x_1}[s_l]])\end{aligned}$$

In our simple voting protocol, the knowledge of the voter x_1 is very simple. They know their shared key with the administrator, the random value that they generate when voting and they know the messages that they send:

$$\begin{aligned}
\mathit{concr}_{x_1} &= \{\!|\ d_1 \mapsto \mathsf{sk}(x_1, a), d_2 \mapsto R_1, \\
&\qquad d_3 \mapsto \mathsf{scrypt}(\mathsf{sk}(x_1, a), \mathsf{ballot}(R_1, \mathsf{candA}))\ |\!\} \\
\mathit{struct}_{x_1} &= \{\!|\ d_1 \mapsto \mathsf{sk}(X_1, a), d_2 \mapsto R_1, \\
&\qquad d_2 \mapsto \mathsf{scrypt}(\mathsf{sk}(X_1, a), \mathsf{ballot}(R_1, V_1))\ |\!\}
\end{aligned}$$

There is no way for the voter x_1 to lie about their true vote because they know neither the secret key that the other voters share with the administrator, nor the other random values. This means that the intruder can exclude some models of α when we require as part of β the receipt-freeness axiom: this protocol is not receipt-free.

Receipt-freeness is a difficult property to express with approaches such as the ones based on Applied-π calculus [12]. We have shown here how to refine our privacy goal in dynamic (α, β)-privacy. Note that was done with an additional axiom to β rather than something we could already express with (α, β)-privacy directly.