A Decision Procedure for Alpha-Beta Privacy for a Bounded Number of Transitions

Laouen Fernet
DTU Compute
Danmarks Tekniske Universitet
Kgs. Lyngby, Danmark
lpkf@dtu.dk

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

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

***Abstract*—We present a decision procedure for verifying whether a protocol respects privacy goals, given a bound on the number of transitions. We consider *multi message-analysis problems*, where the intruder does not know exactly the structure of the messages but rather knows several possible structures and that the real execution corresponds to *one* of them. This allows for modeling a large class of security protocols, with standard cryptographic operators, non-determinism and branching. Our main contribution is the definition of a decision procedure for a fragment of alpha-beta privacy. Moreover, we have implemented a prototype tool as a proof-of-concept and a first step towards automation.**

***Index Terms*—Privacy, Security Protocols, Unlinkability, Formal Methods, Automated Verification.**

I. Introduction

The concept of (α, β)-privacy was introduced as an alternative way to define privacy-type properties in security protocols [1], [2]. The most widespread models of privacy use an equivalence notion between two processes to describe the goal that the intruder cannot distinguish between two possible realities. In contrast, (α, β)-privacy considers states that each represent one possible reality, and what the intruder knows about the reality in that state. This knowledge is not only in form of messages as in classic intruder models, but also in form of relations between messages, agents, etc. Together with a notion of what the intruder is allowed to know in a given state, we define a privacy violation if the intruder in any reachable state knows more than allowed. Privacy is then a question of reachability — a safety property — which is often easier to reason about and to specify than classical equivalence notions. First, one does not have to boil the privacy goal down to a distinction between two situations, which is often unnatural for more complicated properties. Second, one specifies goals positively by what the intruder is allowed to know rather than what they are not allowed to know (and thus unable to distinguish). This essentially means that in the worst case one is erring on the safe side, i.e., allowing less than the protocol actually reveals, and thus can be alerted by a counterexample. The expressive power of equivalence notions and of (α, β)-privacy is actually hard to relate in general, due to the different nature of the approaches. However, on concrete examples it seems one can always give reasonable adaptations from one approach to the other [1], [2].

(α, β)-privacy shifts the problem from a notion of equivalence (that is a challenge for automation) to a simple reachability problem where however the privacy check for each reached state is more involved. So far, there is only one work [3] that considers a solution to checking a given state in (α, β)-privacy. However, that work is only applicable to specifications without conditional branching and it is based on an exploration of all concrete messages that the intruder can send, which are infinitely many unless one bounds the intruder.

Our main contribution in this paper is a decision procedure for the full notion of transaction processes defined by [2] for constructor/destructor theories [4], [5], [6], [7], [8]. This notion in fact entails that the intruder performs a symbolic execution of the transaction that in general yields several possibilities (due to conditional branching if the intruder does not know the truth value of the condition) and the intruder can then contrast this with all observations and experiments (constructing different messages and comparing them) to potentially rule out some possibilities. The core of our work is in a procedure to model this intruder analysis without bounding the number of steps that the intruder can make in this process. To that end, we use a popular constraint-based technique to represent the intruder symbolically, i.e., without exploring infinite sets of possibilities. In fact, we use several layers of symbolic representation to make the approach feasible.

Our decision procedure tells us whether from a given state we can reach a state that violates privacy for a fixed bound on the number of transitions. Our procedure is limited to such a bound on transitions, corresponding to the restriction to a bounded number of sessions in many approaches [9]. This is similar to the bounds needed in tools like APTE [10], AKiSs [11], SPEC [12], [13] and DeepSec [6]. In fact, this paper draws from the techniques used in these approaches, such as the symbolic representation of the intruder, a notion of an analyzed intruder knowledge, and methods for deciding the equivalence of frames. There are, however, several basic differences and generalizations. In particular, we use a symbolic handling of privacy variables (that in the equivalence-based approaches are simply one binary choice) and this is linked to logical formulas about relations between elements of the considered universe. In fact, in the prototype implementation of our decision procedure that we provide as a further contribution, we employ the SMT solver cvc5 [14] to handle these logical evaluations. Moreover, we have multiple frames with constraints for the different possibilities resulting from conditional branching and we analyze if the intruder can rule out any possibilities in any instance.

In contrast, the tools ProVerif [4] and Tamarin [15] do handle unbounded sessions but require the restriction to so-called *diff-equivalence* [16], [8], which drastically limits the use of branching in security protocols, though [17] recently relaxes these restrictions a bit. It seems thus in general that one has to choose between expressive power and unbounded sessions, and our approach is decidedly on the side of expressive power.

We proceed as follows. In §II, we present the notion of (α, β)-privacy in transition systems and define the problem that our procedure decides. In §III, we define how we symbolically represent messages sent by the intruder and how to solve constraints with the *lazy intruder* rules. In §IV, we introduce the notion of symbolic states with their semantics. In §V, we explain how the intruder can perform experiments and make logical deductions relevant for privacy by comparing messages in their knowledge. In §VI, we summarize how the different parts of the procedure are integrated. In §VII, we discuss the prototype tool we have developed and its application to some examples. In §VIII, we discuss related and future work. The appendix contains additional technical details and the proofs of correctness.

II. Preliminaries and Problem Definition

[1] introduces (α, β)-privacy as a reachability problem in a state transition system, where each state contains two formulas α and β. Intuitively, α represents what the intruder may know (e.g., the result of an election) and β what the intruder has seen (e.g., the encrypted votes). Then, a state (α, β) violates privacy iff some model of α can be excluded by the intruder knowing β, i.e., the intruder in that state can rule out more than allowed. The entire transition system violates (α, β)-privacy iff some reachable state does.

A. (α, β)-*Privacy for a State*

[1] focuses on how to define (α, β) pairs for a fixed state, and describes a state transition relation only briefly by an example. Let us also start with a fixed state. The formulas α and β are in *Herbrand logic* [18], a variant of First-Order Logic (FOL), with the difference that the universe is the quotient algebra of the *Herbrand universe* (the set of all terms that can be built with the function symbols) modulo a congruence relation $\approx$. This congruence specifies algebraic properties of cryptographic operators. For concreteness, we use the congruence defined in Fig. 1; for the class of properties supported by our result see Definition VI.1. The quotient algebra consists of the $\approx$-equivalence classes of terms.

Given an alphabet Σ, an *interpretation* $\mathcal{I}$ interprets variable and relation symbols as usual (the interpretation of the function symbols is determined by the Herbrand universe) and we have a model relation $\models_\Sigma$ as expected. By construction, $\mathcal{I} \models_\Sigma s \doteq t$ iff $\mathcal{I}(s) \approx \mathcal{I}(t)$. We say that ϕ *entails* ψ, and write $\phi \models_\Sigma \psi$, when all models of ϕ are models of ψ. We write $\phi \equiv \psi$ when $\phi \models_\Sigma \psi$ and $\psi \models_\Sigma \phi$; we may also use $\equiv$ to define formulas.

$$
\begin{array}{ll}
\mathsf{dcrypt}(s_1, s_2) \approx t & \text{if } s_1 \approx \mathsf{inv}(k) \text{ and } s_2 \approx \mathsf{crypt}(k, t, r) \\
\mathsf{dscrypt}(k, s) \approx t & \text{if } s \approx \mathsf{scrypt}(k, t, r) \\
\mathsf{open}(k, s) \approx t & \text{if } s \approx \mathsf{sign}(\mathsf{inv}(k), t) \\
\mathsf{pubk}(s) \approx k & \text{if } s \approx \mathsf{inv}(k) \\
\mathsf{proj}_1(s) \approx t_1 & \text{if } s \approx \mathsf{pair}(t_1, t_2) \\
\mathsf{proj}_2(s) \approx t_2 & \text{if } s \approx \mathsf{pair}(t_1, t_2) \\
\text{and } \ldots \approx \mathtt{ff} & \text{otherwise}
\end{array}
$$

Fig. 1. The congruence used in this paper: crypt and dcrypt are asymmetric encryption and decryption, scrypt and dscrypt are symmetric encryption and decryption, sign and open are signing and verification/opening, pair is a transparent function and the proj_i are the projections, inv gives the private key corresponding to a public key, and pubk gives the public key from a private key. Here k, t, r and the t_i are variables standing for arbitrary messages. When the conditions are not met, the functions give $\mathtt{ff}$, which is a constant indicating failure of decryption or parsing. If crypt and scrypt are used as binary functions, we consider their deterministic variants where the random factor r has been fixed and is omitted for simplicity.

We now fix the alphabet Σ that contains all symbols we use, namely cryptographic functions, a countable set of constants representing agents, nonces and so on, and some relation symbols. We also have the set of variable symbols $\mathcal{V}$. Each protocol specification will fix a sub-alphabet $\Sigma_0 \subset \Sigma$ of *payload symbols*; we call $\Sigma \setminus \Sigma_0$ the *technical symbols*. All α formulas use only symbols in Σ_0 (besides variables). In the rest of the paper, we often omit the alphabet and just write $\models$ to mean $\models_{\Sigma_0}$.

The main idea of (α, β)-privacy is that we distinguish between the actual privacy goal (e.g., an unlinkability goal talking only about agents) and the means to achieve it (e.g., the cryptographic messages exchanged).

Definition II.1 (Adapted from [1])**.** *Given two formulas α over Σ_0 and β over Σ with $fv(\alpha) \subseteq fv(\beta)$, where fv denotes the free variables, we say that (α, β)-*privacy holds *iff for every $\mathcal{I} \models_{\Sigma_0} \alpha$ there exists $\mathcal{I}' \models_\Sigma \beta$ such that $\mathcal{I}$ and $\mathcal{I}'$ agree on the variables in $fv(\alpha)$ and on the relation symbols in Σ_0.*

Payload. We call the formula α the *payload*, defining the privacy goal. For example, for unlinkability in an RFID-tag protocol, we may have a fixed set $\{t_1, t_2, t_3\}$ of tags and in a concrete state, the intruder has observed that two tags have run a session. Then α in that state may be $x_1, x_2 \in \{t_1, t_2, t_3\}$, meaning that the intruder is only allowed to know that both x_1 and x_2 are indeed tags, but not, for instance, whether $x_1 \doteq x_2$. In our approach, the formulas α that can occur fall into a fragment where we can always compute a finite representation of all models, in particular the variables like the x_i in the example will always be from a fixed finite domain.

Frames. For the formula β, we employ the concept of frames: a *frame* has the form $F = l_1 \mapsto t_1. \cdots .l_n \mapsto t_n$, where the l_i are distinguished constants called *labels* and the t_i are *messages* (that do not contain labels). This represents that the intruder has observed (or initially knows) messages $t_1, \ldots, t_n$ and we give each message a unique label. We call the set $\{l_1, \ldots, l_n\}$ the *domain* of F. A frame can be used as a substitution, mapping labels to messages.

Recipes. To describe intruder deductions, we define a subset Σ_{pub} of the function symbols to be *public*: they represent operations the intruder can perform on known messages. For instance, all symbols used in Fig. 1 are public except for inv, since getting the private key is not an operation that everyone can do themselves.[1] A *recipe* (in the context of a frame F) is any term that consists of only labels (in the domain of F) and public function symbols, so it represents a computation that the intruder can perform on F. We write $F\{\!|\, r \,|\!\}$ for the message generated by the recipe r with the frame F.

Static equivalence. Two frames F_1 and F_2 with the same domain are *statically equivalent*, written $F_1 \sim F_2$, iff for every pair (r_1, r_2) of recipes, we have $F_1\{\!|\, r_1 \,|\!\} \approx F_1\{\!|\, r_2 \,|\!\} \Leftrightarrow F_2\{\!|\, r_1 \,|\!\} \approx F_2\{\!|\, r_2 \,|\!\}$. This means that the intruder cannot distinguish F_1 and F_2, since any experiment they can make (i.e., compare the outcome of two computations r_1, r_2) either gives in both frames the same result or in both frames not.

Message-analysis problem. While static equivalence is typically used to formulate that two states are indistinguishable for the intruder, [1] employs instead *two* frames in each state: $concr$ representing the concrete knowledge of the intruder and $struct$ the structural knowledge. The messages in $struct$ contain the privacy variables from α and $concr$ is one concrete instance of $struct$, representing what is actually the case in that state. A *message-analysis problem* is then defined to have the form $\beta \equiv \alpha \wedge concr \sim struct$ (see [1] for details on formalizing frames in Herbrand logic), where $struct$ contains only variables from α and $concr = \mathcal{I}(struct)$ for one interpretation $\mathcal{I} \models \alpha$.

As an example, let $\alpha \equiv x_1, x_2 \in \{0, 1\}$, $struct = l_1 \mapsto \mathsf{h}(k, x_1).l_2 \mapsto \mathsf{h}(k, x_2)$ and $concr = l_1 \mapsto \mathsf{h}(k, 0).l_2 \mapsto \mathsf{h}(k, 1)$. Observe that there are four models $\mathcal{I} \models \alpha$, and in two of them $concr \sim \mathcal{I}(struct)$ while $concr \not\sim \mathcal{I}(struct)$ in the other two. The goal of the intruder is to rule out models that are not consistent with β. Note that β *requires* $concr \sim struct$: the intruder knows that $concr$ is an instance of $struct$ and thus, if an experiment distinguishes $concr$ and $\mathcal{I}(struct)$ then the intruder can rule out model $\mathcal{I}$. Thus, at this point, the intruder can exclude two models (namely those in which $x_1 \doteq x_2$), so (α, β)-privacy does not hold.

Automation. A naive way to decide (α, β)-privacy for a message-analysis problem (in an algebra where static equivalence is decidable) is to compute all models $\mathcal{I}_1, \ldots, \mathcal{I}_n$ of α and check whether $\mathcal{I}_1(struct) \sim \cdots \sim \mathcal{I}_n(struct)$ (note that in such problems $fv(\alpha) = fv(\beta)$). [3] gives a more efficient procedure that avoids the enumeration of all models: it generalizes the classical procedure for static equivalence of frames to deal with privacy variables, namely checking whether any experiment or decryption step works for every instance of the variables.

[1]The use of inv is just one possible model, and one could choose to model private keys differently, e.g., with public functions for key pair generation and secret seeds. In this paper we use inv as it makes our examples simpler.

B. (α, β)-Privacy for a Transition System

So far we have been talking about only a single (α, β) pair, i.e., a single state of a larger transition system. [2] defines a language for specifying transition systems where the reachable states and their (α, β) pairs are defined by executing atomic transactions. We present their formalization with some minor adaptations to ease our further development.

We distinguish two sorts of variables: the *privacy variables* $\mathcal{V}_{privacy}$, which are denoted with lower-case letters like x and are all introduced in the form $x \in D$ for a finite domain D of public constants from Σ_0, and the *intruder variables* $\mathcal{V}_{intruder}$, which are denoted with upper-case letters like X for messages received and cell reads in a transaction.

We also distinguish destructor and constructor function symbols. In Fig. 1 we have that dcrypt, dscrypt, open, pubk, proj_1 and proj_2 are destructors whereas the rest are constructors. Moreover, we call pair and inv transparent functions, because one can get all their arguments without any key (but recall that inv is not a public function).

Definition II.2 (Protocol specification)**.** *A* protocol specification *consists of*

- *a number of* transaction processes P_i*, where the* P_i *are left processes according to the syntax below, describing the atomic transactions that participants in the protocol can execute;*
- *a number of* memory cells*, e.g.,* $\mathsf{cell}(\cdot)$*, together with a ground context* $C[\cdot]$ *for each memory cell defining the initial value of the memory, so that initially* $\mathsf{cell}(t) = C[t]$.

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

P_l		*Left process*
$::=$	$\mathsf{mode}\ x \in D.P_l$	*Non-deterministic choice*
$\mid$	$\mathsf{rcv}(X).P_l$	*Receive*
$\mid$	P_c	*Center process*
P_c		*Center process*
$::=$	$\mathsf{try}\ X \doteq d(t,t)$ in P_c catch P_c	*Destructor application*
$\mid$	$X := \mathsf{cell}(t).P_c$	*Cell read*
$\mid$	if ϕ then P_c else P_c	*Conditional statement*
$\mid$	$\nu n_1, \ldots, n_k.P_r$	*Fresh constants*
P_r		*Right Process*
$::=$	$\mathsf{snd}(t).P_r$	*Send*
$\mid$	$\mathsf{cell}(t) := t.P_r$	*Cell write*
$\mid$	$\star\ \phi.P_r$	*Release*
$\mid$	0	*Terminate (nil process)*

where mode *is either* $\star$ *or* $\diamond$*,* ϕ *is a quantifier-free Herbrand formula, and* d *is a destructor. Destructors cannot occur elsewhere in terms. For simplicity, we have denoted destructors as binary functions, but we may similarly use unary destructors (like* proj_i *and* pubk *in the example).*

We require that a transaction P is a closed *left process, i.e., $fv(P) = \emptyset$ — we define the* free variables *$fv(P)$ of a process P as expected, where the non-deterministic choices, receives, cell reads and fresh constants are binding. Moreover, for destructor applications:*

$$fv(\mathsf{try}\ X \doteq d(k,t)\ \mathsf{in}\ P_1\ \mathsf{catch}\ P_2) = fv(d(k,t)) \cup (fv(P_1) \setminus \{X\}) \cup fv(P_2)$$

Finally, a bound variable cannot be instantiated a second time and the only place destructors are allowed is in a destructor application with try.

Example II.1 (Running example). Let us consider the following transaction where a server non-deterministically chooses an agent x and a yes/no-decision y, receives a message, tries to decrypt it with their own private key and then sends the decision encrypted with the public key of x:

$$\begin{aligned}
&\star\ x \in \mathsf{Agent}.\star\ y \in \{\mathsf{yes}, \mathsf{no}\}.\\
&\mathsf{rcv}(M).\\
&\mathsf{try}\ N \doteq \mathsf{dcrypt}(\mathsf{inv}(\mathsf{pk}(\mathsf{s})), M)\ \mathsf{in}\\
&\quad \mathsf{if}\ y \doteq \mathsf{yes}\ \mathsf{then}\\
&\qquad \nu r.\mathsf{snd}(\mathsf{crypt}(\mathsf{pk}(x), \mathsf{pair}(\mathsf{yes}, N), r)).0\\
&\quad \mathsf{else}\ \nu r.\mathsf{snd}(\mathsf{crypt}(\mathsf{pk}(x), \mathsf{no}, r)).0\\
&\mathsf{catch}\ 0
\end{aligned}$$

Here the $\star$ means that the choice of x and y is privacy relevant and the intruder may (at least for now) only learn that $x \in \mathsf{Agent}$ and $y \in \{\mathsf{yes}, \mathsf{no}\}$. The outgoing message has a different form depending on y: in the positive case the server also includes the content N of the encrypted message M they received (and if the message is not of the right format, then the transaction simply terminates); in either case the encryption is randomized with a fresh r. We may omit r if we want to model non-randomized encryption. pk is a public function (modeling a fixed public-key infrastructure known to everybody). $\triangleleft$

Much of these processes thus follows standard process calculus constructs. The special constructs of (α, β)-privacy are the non-deterministic choice and release. Choice comes in two flavors: $\star$ if the choice is privacy relevant (as in the example), and $\diamond$ if not. The latter means that the intruder does not a priori learn the choice, but if they find it out, it is not a violation of privacy as such. Accordingly the formula $x \in D$ is added to α when it is marked $\star$, and to β when it is marked $\diamond$. The release is used to declare that a certain fact ϕ may now be known to the intruder; we discuss this construct and what formulas can be released a bit later.

Observe that privacy variables are introduced only by non-deterministic choices mode $x \in D$. If the mode is $\star$, the transaction augments α by $x \in D$, thus specifying that the intruder may not know more about x unless we also explicitly release some information about x. If the mode is $\diamond$, the transaction augments β by $x \in D$. In this case it is *not* automatically a violation of privacy if the intruder learns more about x, but it may lead to a privacy violation if this allows for finding out more about the variables in α. This is useful if one wants to keep the privacy specification independent of some rather technical secret. In our model, the intruder knows which transaction is executed, but in general does not know which branch is taken. Using, for example, $\diamond\ z \in \{1, 2\}.\mathsf{if}\ z \doteq 1\ \mathsf{then}\ P_1\ \mathsf{else}\ P_2$, one can reduce the visibility of transactions P_1 and P_2 by putting them in a single transaction. In some execution the intruder may find out, e.g., $z \doteq 1$, and it is not a privacy violation in itself.

Semantics. The semantics follows [2], with small adaptations. It is defined as a state transition system where each transition corresponds to the execution of one transaction. Thus, transactions are atomic: they cannot run concurrently with another transaction. In particular, when reading from and writing to memory cells, no race conditions can occur and we thus do not need locking mechanisms. A transaction thus consists in receiving input, checking this input (possibly reading from memory), then making a decision (possibly updating the memory), and finally sending an output and releasing information.

The atomicity of transactions has an advantage: we can easily formalize how the intruder can reason about what is happening. In particular, we assume that the intruder at each point knows which transaction is executed and what process a transaction contains. What the intruder does not know in general are the concrete values of the variables and the truth values of conditions, and thus in which branch of an if-then-else or try-catch we are. However, the intruder can always contrast this knowledge with the observations about incoming and outgoing messages: if an observed sent message does not fit with one branch of the transaction, then the intruder knows that branch was not taken, and thus they also learn something about the truth value of the corresponding conditions. In other cases, the intruder may know what is in a received message and thus know the truth value of some condition. The intruder thus performs a symbolic execution of the transaction, leaving open what they do not know, keeping a list of possibilities, and in fact the semantics of transactions formally models this symbolic execution by the intruder.

To formalize the symbolic execution by the intruder, let a *possibility* be a tuple $(P, \phi, struct, \delta)$, where P is the transaction being executed, ϕ is the conditions under which this possibility was reached, $struct$ is the structural knowledge about the messages in this possibility and δ is a sequence of memory updates.

A *state* is a tuple $(\alpha, \beta_0, \gamma, \mathcal{P})$ where α, β_0 and γ are Σ_0-formulas, and $\mathcal{P}$ is a non-empty finite set of possibilities $\mathcal{P} = \{(P_1, \phi_1, struct_1, \delta_1), \dots, (P_n, \phi_n, struct_n, \delta_n)\}$, where one of the possibilities is marked by underlining as the possibility that is actually the case in the real execution (but the intruder does not know which one, in general). In this paper, we consider only well-formed states, where a *state* is *well-formed* iff all $struct_i$ have the same domain, γ describes a unique model of $\alpha \wedge \beta_0$ and the ϕ_i both are mutually exclusive, i.e., $\models \neg(\phi_j \wedge \phi_k)$, for $j \neq k$, and cover all models, i.e., $\alpha \wedge \beta_0 \models \bigvee_{j=1}^{n} \phi_j$. We define the concrete frame $concr$ as the instantiation of the $struct_i$ from the underlined possibility by the model γ.

Definition II.3 (Multi message-analysis problem (MMA))**.** *Given a well-formed state $S = (\alpha, \beta_0, \gamma, \mathcal{P})$, let $concr = \gamma(struct_i)$ where $(P_i, \phi_i, struct_i)$ is the marked possibility in $\mathcal{P}$. Define*

$$MMA(S) = \alpha \wedge \beta_0 \wedge \bigvee_{i=1}^{n} \phi_i \wedge concr \sim struct_i \; .$$

We say that S satisfies privacy *iff $(\alpha, MMA(S))$-privacy holds.*

The possibilities will be used to represent that the intruder in the symbolic execution of a transaction cannot tell which conditions are true, and thus which path the execution actually takes. The $struct_i$ will contain the structural messages (i.e., containing privacy variables) that the transaction sends in the respective case, and ϕ_i is the condition under which this case was entered. In contrast, $concr$ contains the actually observed concrete messages (i.e., privacy variables are instantiated according to the their true value described by γ). The intruder knows that exactly one of the ϕ_i is the case, and that $concr \sim struct_i$, i.e., the concrete messages are an instance of the messages sent in the execution path actually taken.

A state is called *finished* when all processes P_i are 0. The semantics thus defines an evaluation relation $\to$ on states that work off the processes in each possibility until a finished state is reached. This represents the symbolic execution by the intruder of a given a transaction. The branching of $\to$ represents the non-deterministic choices of the process as well as choices of messages by the intruder.

To give a gentle introduction to (α, β)-privacy in transition systems, we present the symbolic execution at hand of the running example from Example II.1. For the complete definition of the rules, see Appendix A. As a starting point for the symbolic execution, we use the singleton set of possibilities $\{\underline{(P, \mathsf{true}, [], [])}\}$ where P is the process from the running example. Let α, β_0, and γ be true; $[]$ denotes the empty frame and empty memory.

1) Non-Deterministic Choice: The first step in P in the example are two non-deterministic choices of privacy variables $\star\ x \in \mathsf{Agent}.\ \star\ y \in \{\mathsf{yes}, \mathsf{no}\}$. For this, the $\to$-relation has actually several successors, one for each possible choice of x and y. (In the decision procedure below we use a more clever way to handle all these successors as one.) The general rule defines for every $c \in D$ the following successor:

$$\begin{aligned}&\{\underline{(\mathsf{mode}\ x \in D.P_1, \phi_1, struct_1, \delta_1)}, \ldots,\\ &\ (\mathsf{mode}\ x \in D.P_n, \phi_n, struct_n, \delta_n)\}\\ &\to \{\underline{(P_1, \phi_1, struct_1, \delta_1)}, \ldots, (P_n, \phi_n, struct_n, \delta_n)\}\end{aligned}$$

where γ is augmented with $x \doteq c$, and if $\mathsf{mode} = \star$ (resp. $\mathsf{mode} = \diamond$) then α (resp. β_0) is augmented with $x \in D$. γ thus represents what really happened (which the intruder cannot see) and the information about the domain is released to α or β_0, depending on whether x is privacy relevant. Note that x is not replaced in the P_i — this is a symbolic execution by the intruder. Also note that this rule assumes that all possibilities start with the same mode $x \in D$; this is ensured since this choice can only occur in the left part of the transaction, before any branching on conditions and tries can occur.

For the example, let us follow $x = \mathsf{a}$ and $y = \mathsf{yes}$; this is added to γ, and we add to α that $x \in \mathsf{Agent}$ and $y \in \{\mathsf{yes}, \mathsf{no}\}$.

2) Receive: The next step is $\mathsf{rcv}(M)$. Again the construction ensures that every process in the possibilities starts with a receive step (with the same variable). Here, the intruder can choose an arbitrary recipe r (over the domain of the $struct_i$) for the message that should be received as M. In fact, in general, we have here infinitely many possible r and thus infinitely many successors. (Our decision procedure below uses a constraint-based approach to handle this in a finite way.) The general rule allows for every r over the domain of the $struct_i$ the following transition:

$$\begin{aligned}&\{\underline{(\mathsf{rcv}(X).P_1, \phi_1, struct_1, \delta_1)}, \ldots,\\ &\ (\mathsf{rcv}(X).P_n, \phi_n, struct_n, \delta_n)\}\\ &\to \{\underline{(P_1[X \mapsto struct_1\{\!\!|\, r \,|\!\!\}], \phi_1, struct_1, \delta_1)}, \ldots,\\ &\qquad (P_n[X \mapsto struct_n\{\!\!|\, r \,|\!\!\}], \phi_n, struct_n, \delta_n)\}\end{aligned}$$

Observe that the message that is being received depends on the possibility: it is $struct_i\{\!\!|\, r \,|\!\!\}$ in the ith possibility, i.e., whatever the recipe r yields in the respective intruder knowledge $struct_i$.

As the intruder knowledge at this point is empty in the example, r can only be a recipe built from public constants and functions. Let us consider $r = \mathsf{crypt}(\mathsf{pk}(\mathsf{s}), \mathsf{a}, \mathsf{h}(\mathsf{a}))$, which then replaces M in the process.

3) Cell Read: The memory δ contains the sequence of updates $\mathsf{cell}(s_1) := t_1. \cdots .\mathsf{cell}(s_k) := t_k$ for the given cell, and the initial value is given with ground context $C[\cdot]$.

$$\begin{aligned}&\{(X := \mathsf{cell}(s).P, \phi, struct, \delta)\} \uplus \mathcal{P}\\ &\to \{(\mathsf{if}\ s \doteq s_1\ \mathsf{then}\ P[X \mapsto t_1]\ \mathsf{else}\\ &\qquad \ldots\\ &\qquad \mathsf{if}\ s \doteq s_k\ \mathsf{then}\ P[X \mapsto t_k]\ \mathsf{else}\\ &\qquad P[X \mapsto C[s]], \phi, struct, \delta)\} \cup \mathcal{P}\end{aligned}$$

4) Cell Write: A memory update is added to the sequence δ. Note that it is important to *prepend* the update so that when we do a cell read, the most recent state is used first in a conditional, effectively overwriting the previous memory state.

$$\begin{aligned}&\{(\mathsf{cell}(s) := t.P, \phi, struct, \delta)\} \uplus \mathcal{P}\\ &\to \{(P, \phi, struct, \mathsf{cell}(s) := t.\delta)\} \cup \mathcal{P}\end{aligned}$$

5) Conditional Statement: The next step in the running example is $\mathsf{try}\ N \doteq \ldots\ \mathsf{in}\ P_0\ \mathsf{catch}\ 0$. For the sake of this semantics, we can just consider $\mathsf{try}\ X \doteq t\ \mathsf{in}\ P_1\ \mathsf{catch}\ P_2$ as syntactic sugar for $\mathsf{if}\ (t \doteq \mathsf{ff})\ \mathsf{then}\ P_2\ \mathsf{else}\ P_1[X \mapsto t]$. (For the decision procedure it is important that destructors only occur in this try-catch form, however.)

We have a general rule that can fire when the next step in one of the possibilities is an if-then-else. In this case we split that possibility into two, one for the case that the condition is true and we go into the then branch, and one for the else branch:

$$\{(\mathsf{if}\ \psi\ \mathsf{then}\ P_1\ \mathsf{else}\ P_2, \phi, \mathit{struct}, \delta)\} \uplus \mathcal{P} \\ \to \{(P_1, \phi \wedge \psi, \mathit{struct}, \delta), (P_2, \phi \wedge \neg\psi, \mathit{struct}, \delta)\} \cup \mathcal{P}$$

In our example, we thus have to evaluate the condition $\mathsf{dcrypt}(\mathsf{inv}(\mathsf{pk}(\mathsf{s})), \mathsf{crypt}(\mathsf{pk}(\mathsf{s}), \mathsf{a}, \mathsf{h}(\mathsf{a}))) \doteq \mathsf{ff}$, which we can simplify to false, i.e., the intruder knows that the received message will decrypt correctly. We thus have the two possibilities $\{(0, \mathsf{false}, [], []), \underline{(\mathsf{if}..., \mathsf{true}, [], [])}\}$, of which the second is underlined, and an evaluation rule allows removing possibilities with the condition false. The underlined possibility is what really happened (which is here obvious).

We thus apply a second time the condition rule, again splitting into two possibilities:

$$\{\underline{(\nu r.\mathsf{snd}(...(\mathsf{yes}, N), r), y \doteq \mathsf{yes}, [], [])}, \\ (\nu r.\mathsf{snd}(...\mathsf{no}, r), y \not\doteq \mathsf{yes}, [], [])\}$$

Here the first possibility is what really happens (as stated by γ) and is thus underlined, but here the intruder does not know which one is the case.

The ν operator can be implemented by replacing the placeholder by a fresh non-public constant, say r_1. We can in fact do this as a preparation before executing the transaction.

6) Send: When all the rules for the other constructs have been applied as far as possible, each of the remaining processes must be either a send or 0. If the intruder observes that a message is sent, this rules out all possibilities where the remaining process is 0. For all others, each struct_i is augmented by the message sent in the respective possibility:

$$\{\underline{(\mathsf{snd}(t_1).P_1, \phi_1, \mathit{struct}_1, \delta_1)}, \dots, \\ (\mathsf{snd}(t_k).P_k, \phi_k, \mathit{struct}_k, \delta_k)\} \uplus \mathcal{P} \\ \to \{\underline{(P_1, \phi_1, \mathit{struct}_1.l \mapsto t_1, \delta_1)}, \dots, \\ (P_k, \phi_k, \mathit{struct}_k.l \mapsto t_k, \delta_k)\}$$

where $\beta_0 \leftarrow \beta_0 \wedge \bigvee_{i=1}^{k} \phi_i$, l is a fresh label and all the processes in $\mathcal{P}$ must be the 0 process. This requirement forbids applying the send rule as long as the next step of any possibility is different from send and 0 (so some of the other rules has to be applied first).

In our example we thus reach the state with

$$\{\underline{(0, y \doteq \mathsf{yes}, l \mapsto \mathsf{crypt}(\mathsf{pk}(x), \mathsf{pair}(\mathsf{yes}, \mathsf{a}), r_1), [])}, \\ (0, y \not\doteq \mathsf{yes}, l \mapsto \mathsf{crypt}(\mathsf{pk}(x), \mathsf{no}, r_1), [])\}$$

and $\mathit{concr}\{\!\!\{ l \}\!\!\} = \mathsf{crypt}(\mathsf{pk}(\mathsf{a}), \mathsf{pair}(\mathsf{yes}, \mathsf{a}), r_1)$ which is a finished state, and the intruder has thus finished the symbolic execution of this transaction.

Example II.2. Let us point out a few more interesting features of our running example. At the finished state, without further knowledge, the intruder is unable to tell which of the two possibilities is the case. This would be different if the encryption were not randomized: suppose we drop the third argument of crypt. Then the intruder could now construct $\mathsf{crypt}(\mathsf{pk}(x'), \mathsf{no})$ for each value $x' \in \mathsf{Agent}$ and compare the result with the learned message. Since this does not succeed in any case, the intruder learns that the second possibility is excluded, thus $y \doteq \mathsf{yes}$, violating (α, β)-privacy. Even worse, if we look at the state where the non-deterministic choice was $y = \mathsf{no}$, the intruder would find out x because exactly one of the guesses succeeds.

Reverting to randomized encryption, suppose that there had been an earlier transaction where the intruder learned $l \mapsto \mathsf{crypt}(\mathsf{pk}(z), \mathsf{no}, r_2)$ for some privacy variable $z \in \mathsf{Agent}$. If the intruder uses this as input for the next transaction, then the decryption works iff $z \doteq \mathsf{s}$. Thus, we have a third possibility at the final sending step, namely $(0, z \not\doteq \mathsf{s}, l \mapsto \mathsf{crypt}(\mathsf{pk}(z), \mathsf{no}, r_2), [])$. Then from the fact that a message was sent, the intruder can rule out this third possibility and thus deduce that $z \doteq \mathsf{s}$, again violating (α, β)-privacy. ◁

7) Release: This construct is used to declare information that the intruder is now allowed to learn, for instance in some cases we may want to intruder to learn the true value of a privacy variable. In our running example, we do not use the release. However, in case the server is replying to the intruder, then the intruder can decrypt the message and observe what was the decision. Thus they would learn both the value of x (i.e., the agent was the intruder) and y (i.e., the know the server's decision). We could thus refine the transaction and add a release if x is the intruder.

The formula released by the marked possibility is added to the payload and formulas released by other possibilities are ignored.

$$\{\underline{(\star\ \psi.P, \phi, \mathit{struct}, \delta)}\} \uplus \mathcal{P} \to \{\underline{(P, \phi, \mathit{struct}, \delta)}\} \cup \mathcal{P}$$

and $\alpha \leftarrow \alpha \wedge \psi$. We consider it a specification error if when applying this symbolic execution rule, the formula ϕ released contains symbols which are in $\Sigma \setminus \Sigma_0$ and variables not in $\mathit{fv}(\alpha)$. Thus, the specification can use symbols from the technical level in a release *as long as* the evaluated terms use only symbols in Σ_0 and $\mathit{fv}(\alpha)$ (i.e., the payload level) when executing the protocol. This means that releasing technical information in the payload is not allowed. Additionally for our decision procedure below, the same requirement applies to a relational formula $R(t_1, \dots, t_n)$ in the symbolic execution of conditional statements. This kind of specification error can be detected during the symbolic execution and means that insufficient checks are made over the terms before the release or conditional statement.

8) Terminate: The intruder observes that the execution has terminated because no messages are sent, so they can rule out all possibilities that are not terminated.

$$\{\underline{(0, \phi_1, \mathit{struct}_1, \delta_1)}, \dots, (0, \phi_k, \mathit{struct}_k, \delta_k)\} \uplus \mathcal{P} \\ \to \{\underline{(0, \phi_1, \mathit{struct}_1, \delta_1)}, \dots, (0, \phi_k, \mathit{struct}_k, \delta_k)\}$$

where every process in $\mathcal{P}$ starts with a send step and $\beta_0 \leftarrow \beta_0 \wedge \bigvee_{i=1}^{k} \phi_i$.

9) The Problem: We have defined the relation $\rightarrow$ that works off the steps of a transaction, modeling an intruder's symbolic execution of a transaction P. We now define a transition relation $\longrightarrow$ on finished states (i.e., the process in every possibility is 0) such that $S \longrightarrow S'$ iff there is a transaction P such that $init(P, S) \rightarrow^* S'$, where $init(P, S)$ denotes replacing the 0-process in every possibility of S by process P. Let the initial state be $S_0 = (\mathsf{true}, \mathsf{true}, \mathsf{true}, \{\underline{(0, \mathsf{true}, [], [])}\})$.

Definition II.4 (The problem)**.** *A protocol specification* satisfies privacy *iff* (α, β)*-privacy holds for every* S *s.t.* $S_0 \longrightarrow^* S$.

The contribution of the present paper is a procedure to decide whether for a given bound k, a violation is reachable in at most k steps, under the restriction of the algebraic properties to constructor/destructor theories of Definition VI.1.

III. FLICs: Framed Lazy Intruder Constraints

The semantics of the transition system says that, in a state where the processes are receiving a message, the intruder can choose any recipe built on the domain of $concr$ (respectively, the $struct_i$: they all have the same domain). The problem is that there are in general infinitely many recipes the intruder can choose from. A classic technique for deciding such infinite spaces of intruder possibilities is a constraint-based approach that we call the *lazy intruder* [19], [9], [20]: it is lazy in that it avoids, as long as possible, instantiating the variables of receive steps like $\mathsf{rcv}(X)$. The concrete intruder choice at this point does not matter; only when we check a condition that depends on X, we consider possible instantiations of X as far as needed to determine the outcome of the condition. Note that this is another symbolic layer of our approach: a symbolic state with variable X represents all concrete states where X is replaced with a message that the intruder can construct. In fact what the intruder can construct depends on the messages the intruder knew at the time when the message represented by X was sent. Due to the symbolic execution, in a state there are in general several $struct_i$, and thus we need not only to represent the messages sent by the intruder with variables but also the recipes that they have chosen, because a given recipe can produce different messages in each $struct_i$.

To keep track of this, we define an extension of frames called *framed lazy intruder constraints (FLICs)*: the entries of a standard frame represent messages that the intruder received and we write them now with a minus sign: $-l \mapsto t$. We extend this by also writing entries for messages the intruder sends with a plus sign: $+R \mapsto t$, where R is a *recipe variable* (disjoint from privacy and intruder variables). When solving the constraints, R may be instantiated with a more concrete recipe, but only using labels that occurred in the FLIC before this receive step; the order of the entries is thus significant. The messages t can contain variables representing intruder choices that we have not yet made concrete. We require that the intruder variables first occur in positive entries as they represent intruder choices made when sending a message.

Since we deal with several possibilities in parallel, we will have several FLICs in parallel, replacing the $struct_i$ in the ground model. Each FLIC has the same sequence of incoming labels and outgoing recipes. The intruder does not know in general which possibility is the case, but knows how they constructed the message from their knowledge, i.e., the recipe, which may result in a different message in each possibility.

A FLIC is a constraint, namely that the intruder can indeed produce messages of the form needed to reach a particular state of the execution. We show that we can *solve* such FLICs, i.e., find a finite representation of all solutions (as said before, there are in general infinitely many possible concrete choices) using the lazy intruder technique, similarly to other works doing constraint-based solving with frames such as [21], [6]. In the rest of this section, we will focus first on defining and solving constraints by considering just one FLIC and not the rest of the possibilities, and we explain afterwards how the lazy intruder is used for the transition system with several possibilities.

A. Defining Constraints

Definition III.1 (FLIC)**.** *A* framed lazy intruder constraint (FLIC) $\mathcal{A}$ *is a sequence of mappings of the form* $-l \mapsto t$ *or* $+R \mapsto t$*, where each label* l *and recipe variable* R *occurs at most once, each term* t *is built from function symbols, privacy variables, and intruder variables. The first occurrence of each intruder variable must be in a message sent.*

We write $-l \mapsto t \in \mathcal{A}$ *if* $-l \mapsto t$ *occurs in* $\mathcal{A}$*, and similarly* $+R \mapsto t \in \mathcal{A}$*. The* domain $dom(\mathcal{A})$ *is the set of labels of* $\mathcal{A}$ *and* $vars(\mathcal{A})$ *are the privacy and intruder variables that occur in* $\mathcal{A}$*; similarly, we write* $rvars(\mathcal{A})$ *for the recipe variables.*

The message $\mathcal{A}\{\!|\, r \,|\!\}$ produced *by* r *in* $\mathcal{A}$ *is:*

$$\begin{aligned} \mathcal{A}\{\!|\, l \,|\!\} &= t && \text{if } -l \mapsto t \in \mathcal{A} \\ \mathcal{A}\{\!|\, R \,|\!\} &= t && \text{if } +R \mapsto t \in \mathcal{A} \\ \mathcal{A}\{\!|\, f(r_1, \dots, r_n) \,|\!\} &= f(\mathcal{A}\{\!|\, r_1 \,|\!\}, \dots, \mathcal{A}\{\!|\, r_n \,|\!\}) \end{aligned}$$

For recipes that use labels or recipe variables not defined in the FLIC, the result is undefined.

We also define an ordering between recipes and labels: $r <_{\mathcal{A}} l$ *iff every label* l' *in* r *occurs before* l *in* $\mathcal{A}$*.*

Example III.1. Consider the transaction from Example II.1, step $\mathsf{rcv}(M)$. Using FLICs, we add $+R \mapsto M$ to the FLIC (where both R and M are fresh variables). We are lazy in the sense that we do not explore at this point what R and M might be, because any value would do. Now the server checks whether M can be decrypted with the private key $\mathsf{inv}(\mathsf{pk}(\mathsf{s}))$. This is the case iff M has the form $\mathsf{crypt}(\mathsf{pk}(\mathsf{s}), \cdot, \cdot)$. In the positive case, M is instantiated with $\mathsf{crypt}(\mathsf{pk}(\mathsf{s}), X, Y)$ for two fresh intruder variables X and Y, thus requiring that R indeed yields a message of this form. The constraint solving in §III-B computes a finite representation of all solutions for R. The negative case is considered separately, where we remember the negated equality $M \not\doteq \mathsf{crypt}(\mathsf{pk}(\mathsf{s}), \cdot, \cdot)$. $\triangleleft$

Definition III.2 (Semantics of FLICs)**.** *Let* $\mathcal{A}$ *be a FLIC such that* $vars(\mathcal{A}) = \emptyset$*, i.e., the messages in* $\mathcal{A}$ *are ground,* *so $\mathcal{A}$ has only recipe variables. $\mathcal{A}$ is* constructable *iff there exists a ground substitution of recipe variables ρ_0 such that $\mathcal{A}_1\{\!|\, \rho_0(R) \,|\!\} \approx t$ for every recipe variable R where $\mathcal{A} = \mathcal{A}_1.{+R} \mapsto t.\mathcal{A}_2$. (This implies that only labels from $dom(\mathcal{A}_1)$ can occur in $\rho_0(R)$.) We then say that ρ_0* constructs *$\mathcal{A}$.*

Let $\mathcal{A}$ be an arbitrary FLIC and $\mathcal{I}$ be an interpretation of all privacy and intruder variables. We say that $\mathcal{I}$ is a model *of $\mathcal{A}$, written $\mathcal{I} \models \mathcal{A}$, iff $\mathcal{I}(A)$ is constructable. We say that $\mathcal{A}$ is* satisfiable *iff it has a model.*

A FLIC is thus satisfiable if there exist a suitable interpretation for the variables in messages and intruder choice for the variables in recipes such that all the constraints are satisfied.

Example III.2. Suppose that Alice sent a signed message m to the intruder i and the constraint is to send some signed message to Bob. This is recorded in the following FLIC $\mathcal{A}$:

$$-l_1 \mapsto \mathsf{inv}(\mathsf{pk}(\mathsf{i})).-l_2 \mapsto \mathsf{crypt}(\mathsf{pk}(\mathsf{i}), \mathsf{sign}(\mathsf{inv}(\mathsf{pk}(\mathsf{a})), \mathsf{m})).$$
$$+R \mapsto \mathsf{crypt}(\mathsf{pk}(\mathsf{b}), \mathsf{sign}(\mathsf{inv}(\mathsf{pk}(X)), Y))$$

Here $\mathcal{I}_1 = [X \mapsto \mathsf{a}, Y \mapsto \mathsf{m}]$ is a model, since $\mathcal{I}_1(\mathcal{A})$ is constructable using $R = \mathsf{crypt}(\mathsf{pk}(\mathsf{b}), \mathsf{dcrypt}(l_1, l_2))$. For every ground recipe r over $dom(\mathcal{A})$ also $\mathcal{I}_r = [X \mapsto \mathsf{i}, Y \mapsto \mathcal{A}\{\!|\, r \,|\!\}]$ is a model, using $R = \mathsf{crypt}(\mathsf{pk}(\mathsf{b}), \mathsf{sign}(l_1, r))$; note there are infinitely many such r. ◁

B. Solving Constraints

We now present how to solve constraints when the intruder does not have access to destructors, i.e., as if all destructors were private functions and thus cannot occur in recipes. Hence the only place where destructors can occur are in transactions using try-catch. This allows us to work in the free algebra *for now* and with only destructor-free terms. We show in §VI how to integrate the lazy intruder without destructors and special transactions, so that the correctness of our decision procedure is valid for the intruder model with access to destructors.

Definition III.3 (Simple FLIC)**.** *A FLIC $\mathcal{A}$ is called* simple *iff every message sent is an intruder variable, and each intruder variable is sent only once, i.e., every message sent is of the form $+R_i \mapsto X_i$ and the X_i are pairwise distinct.*

Simple FLICs are always satisfiable, since there are no more constraints on the messages, and the intruder can choose any recipes they want. In order to solve constraints in a non-simple FLIC, we instantiate privacy, intruder and recipe variables until we reach a simple FLIC. Computing a finite representation of all solutions is then done by keeping track of the substitutions applied to instantiate the variables.

Definition III.4. *Let σ be a substitution that does not contain recipe variables. We define $\sigma(-l \mapsto t.\mathcal{A}) = -l \mapsto \sigma(t).\sigma(\mathcal{A})$ and $\sigma(+R \mapsto t.\mathcal{A}) = +R \mapsto \sigma(t).\sigma(\mathcal{A})$.*

For the substitutions of recipe variables, however, we cannot directly define the instantiation of recipe variables for an arbitrary FLIC, because we always need to make sure we instantiate both the recipe and the intruder variables according to the constraints. We thus define how to apply a substitution of recipe variables for simple FLICs.

Definition III.5 (Choice of recipes)**.** *A* choice of recipes for a simple FLIC $\mathcal{A}$ *is a substitution ρ mapping recipe variables to recipes, where $dom(\rho) \subseteq rvars(\mathcal{A})$.*

Let $[R \mapsto r]$ be a choice of recipes for $\mathcal{A}$ that maps only one recipe variable, where $\mathcal{A} = \mathcal{A}_1.{+R} \mapsto X.\mathcal{A}_2$. Let $R_1, \dots, R_n$ be the fresh variables in r, i.e., $\{R_1, \dots, R_n\} = rvars(r) \setminus rvars(\mathcal{A})$, taken in a fixed order (e.g., the order in which they first occur in r). Let $X_1, \dots, X_n$ be fresh intruder variables. The application of $[R \mapsto r]$ to the FLIC $\mathcal{A}$ is defined as $[R \mapsto r](\mathcal{A}_1.{+R} \mapsto X.\mathcal{A}_2) = \mathcal{A}'.\sigma(\mathcal{A}_2)$ where $\mathcal{A}' = \mathcal{A}_1.{+R_1} \mapsto X_1.\cdots.{+R_n} \mapsto X_n$ and $\sigma = [X \mapsto \mathcal{A}'\{\!|\, r \,|\!\}]$.

For the general case, let ρ be a choice of recipes for $\mathcal{A}$. Then we define $\rho(\mathcal{A})$ recursively where one recipe variable is substituted at a time, and we follow the order in which the recipe variables occur in $\mathcal{A}$: if $\rho = [R \mapsto r]\rho'$, where R occurs in $\mathcal{A}$ before any $R' \in dom(\rho')$, then $\rho(\mathcal{A}) = \rho'([R \mapsto r](\mathcal{A}))$. Every application $[R \mapsto r](\mathcal{A})$ corresponds to a substitution $\sigma = [X \mapsto \mathcal{A}'\{\!|\, r \,|\!\}]$ (as defined above), and we denote with $\sigma_\rho^{\mathcal{A}}$ the idempotent substitution aggregating all these substitutions σ from applying ρ to $\mathcal{A}$.

Remark. If ρ is a choice of recipes for a simple FLIC $\mathcal{A}$, then $\rho(\mathcal{A})$ is simple, because the fresh recipe variables added in $\rho(\mathcal{A})$ map to fresh intruder variables. ◁

Unification. We use an adapted version of syntactic unification, where we orient so that privacy variables are never substituted with intruder variables, e.g., an equality $x \doteq X$ of a privacy variable x and an intruder variable X yields the unifier $[X \mapsto x]$. We denote with $mgu(s_1 \doteq t_1 \wedge \cdots \wedge s_n \doteq t_n)$ the result, called *most general unifier (mgu)*, of unifying the s_i and t_i, which is either some substitution or $\perp$ whenever no unifier exists. Slightly abusing notation, we consider a substitution $[x_1 \mapsto t_1, \dots, x_n \mapsto t_n]$ as the formula $x_1 \doteq t_1 \wedge \cdots \wedge x_n \doteq t_n$ and $\perp$ as false. Moreover, every privacy variable is associated to a domain by a formula $x \in D$, defining $dom(x) = D$. Thus, we filter out the mgus that are inconsistent w.r.t. the domain specifications (e.g., $[x \mapsto \mathsf{a}]$ is filtered out if $\mathsf{a} \notin dom(x)$).

The lazy intruder rules. In order to solve the constraints, we define a reduction relation $\rightsquigarrow$ on FLICs. The idea is that $\rightsquigarrow$ is Noetherian and a FLIC that cannot be further reduced is either simple or unsatisfiable. Moreover, $\rightsquigarrow$ is not confluent, but rather is meant to explore different ways for the intruder to satisfy constraints, and thus we will consider the set of all simple FLICs that are reachable from a given one: the simple FLICs together will be equivalent to the given FLIC. Since $\rightsquigarrow$ is not only Noetherian, but also finitely branching, the set of reachable simple FLICs is always finite by Kőnig's lemma.

Definition III.6 (Lazy intruder rules)**.** *The relation $\rightsquigarrow$ is a relation on triples $(\rho, \mathcal{A}, \sigma)$ of a choice of recipes ρ, a FLIC $\mathcal{A}$ and a substitution σ, where ρ and σ keep track of all variable substitutions performed in the reduction steps so far. We require that $dom(\rho) \cap rvars(\mathcal{A}) = \emptyset$ and $dom(\sigma) \cap vars(\mathcal{A}) = \emptyset$. The rules are defined in Table I.*

TABLE I
LAZY INTRUDER RULES

Unification	$(\rho, \mathcal{A}_1.-l \mapsto s.\mathcal{A}_2.+R \mapsto t.\mathcal{A}_3, \sigma) \rightsquigarrow (\rho', \sigma'(\mathcal{A}_1.-l \mapsto s.\mathcal{A}_2.\mathcal{A}_3), \sigma')$ where $\rho' = [R \mapsto l]\rho$ and $\sigma' = mgu(\sigma \wedge s \doteq t)$	if $\mathcal{A}_1.-l \mapsto s.\mathcal{A}_2$ is simple, $s, t \notin \mathcal{V}$ and $\sigma' \neq \bot$
Composition	$(\rho, \mathcal{A}_1.+R \mapsto f(t_1, \dots, t_n).\mathcal{A}_2, \sigma) \rightsquigarrow (\rho', \mathcal{A}_1.+R_1 \mapsto t_1.\cdots.+R_n \mapsto t_n.\mathcal{A}_2, \sigma)$ where the R_i are fresh recipe variables and $\rho' = [R \mapsto f(R_1, \dots, R_n)]\rho$	if $\mathcal{A}_1$ is simple, $f \in \Sigma_{pub}$ and $\sigma \neq \bot$
Guessing	$(\rho, \mathcal{A}_1.+R \mapsto x.\mathcal{A}_2, \sigma) \rightsquigarrow (\rho', \sigma'(\mathcal{A}_1.\mathcal{A}_2), \sigma')$ where $\rho' = [R \mapsto c]\rho$ and $\sigma' = mgu(\sigma \wedge x \doteq c)$	if $\mathcal{A}_1$ is simple, $c \in dom(x)$ and $\sigma' \neq \bot$
Repetition	$(\rho, \mathcal{A}_1.+R_1 \mapsto X.\mathcal{A}_2.+R_2 \mapsto X.\mathcal{A}_3, \sigma) \rightsquigarrow (\rho', \mathcal{A}_1.+R_1 \mapsto X.\mathcal{A}_2.\mathcal{A}_3, \sigma)$ where $\rho' = [R_2 \mapsto R_1]\rho$	if $\mathcal{A}_1.+R_1 \mapsto X.\mathcal{A}_2$ is simple and $\sigma \neq \bot$

Unification When the intruder has to send a message, they can use any message previously received and that unifies, by choosing a label for the recipe variable. Then there is one less message to send, but the unifier might make other constraints non-simple. This rule is *not* applicable for variables: the intruder is *lazy*.

Composition When the intruder has to send a composed message $f(t_1, \dots, t_n)$, they can generate it themselves if f is public and they can generate the t_i. The intruder thus chooses to compose the message themselves, so the recipe R is the application of f to other recipes.

Guessing When the intruder has to send a privacy variable x, they can guess the actual value of x, say c. In fact, this is a guessing attack as we let the privacy variables range over small domains of public constants. This rule represents the case that the intruder guesses correctly, and the variable x is replaced by the guessed value c. Note that using the **Guessing** rule does *not* yet mean that the intruder knows that c is the correct guess: in the rest of the procedure, whenever there is such a guess we model both the right and wrong guesses, and the intruder may not be able to tell what is the case.

Repetition If the intruder has to send an intruder variable that they have already sent earlier, they use the same recipe. Since there may be several ways to generate the same message, one may wonder if this is actually complete: could there be an attack where constructing the same messages in two different ways would tell the intruder anything more? In fact, for what concerns the behavior of the honest agents, it cannot make a difference, and comparing different ways to construct the same message is covered in the experiments later.

We now define the lazy intruder results as the set of recipe choices ρ that solve the constraint:

Definition III.7 (Lazy intruder results)**.** *Let $\mathcal{A}$ be a FLIC and σ be a substitution. Let ε be the identity substitution. We define $LI(\mathcal{A}, \sigma) = \{\rho \mid (\varepsilon, \sigma(\mathcal{A}), \sigma) \rightsquigarrow^* (\rho, \mathcal{A}', \_), \mathcal{A}'$ is simple$\}$.*

Example III.3. Following Example III.1, let us assume that the intruder has already observed a message encrypted for the server from another agent x', and is now symbolically executing the transaction. With the constraint induced by the decryption from the server, the FLIC is now $-l \mapsto \mathsf{crypt}(\mathsf{pk}(\mathsf{s}), x', r).+R \mapsto \mathsf{crypt}(\mathsf{pk}(\mathsf{s}), X, Y)$. Since pk is public, the lazy intruder returns two choices of recipes: $\rho_1 = [R \mapsto l]$, meaning the intruder replays the message from the knowledge (since it unifies), and $\rho_2 = [R \mapsto \mathsf{crypt}(\mathsf{pk}(\mathsf{s}), R_1, R_2)]$, meaning the intruder composes the message themselves where R_1 and R_2 stand for arbitrary recipes. $\lhd$

Definition III.8 (Representation of choice of recipes)**.** *Let $\mathcal{A}$ be a FLIC, $\mathcal{I} \models\!\!\!\equiv \mathcal{A}$, ρ_0 be a ground choice of recipes and ρ be a choice of recipes. We say that ρ* represents *ρ_0* w.r.t. *$\mathcal{A}$* and *$\mathcal{I}$ iff there exists ρ_0' such that ρ_0' is an instance of ρ and for every $R \in rvars(\mathcal{A})$, $\mathcal{I}(\mathcal{A})\{\!\!\{ \rho_0'(R) \}\!\!\} = \mathcal{I}(\mathcal{A})\{\!\!\{ \rho_0(R) \}\!\!\}$ and:*

- *If $\rho(R) \in dom(\mathcal{A})$, then $\rho_0(R) \in dom(\mathcal{A})$ and either $\rho_0'(R) = \rho_0(R)$ or $\rho_0'(R) <_{\mathcal{A}} \rho_0(R)$.*
- *If $\rho(R)$ is a composed recipe and $\rho_0(R) \in dom(\mathcal{A})$, then $\rho_0'(R) <_{\mathcal{A}} \rho_0(R)$.*

This notion of representation gives the lazy intruder some "liberty", namely to be lazy in not instantiating recipe variables that do not matter, and to replace subrecipes with equivalent ones (that may be smaller according to our ordering between recipes and labels). In the completeness proof we show that, despite all these liberties, every solution of the constraint is represented by some recipe choice that the lazy intruder finds. The lazy intruder rules are sound, complete and terminating:

Theorem III.1 (Lazy intruder correctness)**.** *Let $\mathcal{A}$ be a FLIC, σ be a substitution, $\mathcal{I} \models\!\!\!\equiv \mathcal{A}$ such that $\mathcal{I} \models \sigma$ and let ρ_0 be a ground choice of recipes. Then ρ_0 constructs $\mathcal{I}(\mathcal{A})$ iff there exists $\rho \in LI(\mathcal{A}, \sigma)$ such that ρ represents ρ_0 w.r.t. $\mathcal{A}$ and $\mathcal{I}$. Moreover, $LI(\mathcal{A}, \sigma)$ is finite.*

IV. THE SYMBOLIC STATES

Our approach explores a symbolic transition system, i.e., transitions on symbolic states, where each symbolic state represents an infinite set of ground states. Our notion of ground states is an adaptation of the states defined in [2]. We denote symbolic states by $\mathcal{S}$, $\mathcal{S}'$, etc., and ground states by S, S', etc.

A ground state may actually contain privacy variables, representing the possible uncertainty of the intruder in this state, but each variable has one concrete value that represents *the truth* in that state, which will be expressed by a formula γ that the intruder does not have access to (and the frame $concr$ is an instance of one of the $struct_i$ under γ). This is the reason why we call it a ground state, even though it contains variables. A symbolic state includes actually two symbolic layers. For the first symbolic layer, we define a symbolic state to merge all those ground states that differ only in the concrete γ and thus the concrete frame $concr$, i.e., where the intruder has the same uncertainty. Therefore, a symbolic state does not contain γ and $concr$, and has no underlined possibility. Thus, we need to keep track of the released formula α_i for each possibility separately. A second symbolic layer is to use intruder variables and FLICs to avoid enumerating the infinite choices that the intruder has when sending messages, thus the frames $struct_i$ are generalized to FLICs $\mathcal{A}_i$ in symbolic states.

Like a ground state, a symbolic state $\mathcal{S}$ contains processes (one for each $\mathcal{A}_i$) that represent pending steps of a transaction being executed. Only when these steps have been worked off and we have only 0-processes remaining (and certain evaluations have been made), the resulting *finished* symbolic state is a reachable (symbolic) state of the transition system. This in particular ensures that transactions can only be executed atomically. Moreover, to keep track of the intruder experiments that have already been performed (i.e., comparing the outcome of two recipes — details in §V), in a symbolic state we have a set *Checked* that contains pairs of a label and a recipe.

Definition IV.1 (Symbolic state)**.** *A* symbolic state *is a tuple* $(\alpha_0, \beta_0, \mathcal{P}, \mathit{Checked})$ *such that:*

- α_0 *is a* Σ_0*-formula, the* common payload*;*
- β_0 *is a* Σ_0*-formula, the* intruder reasoning *about possibilities and privacy variables;*
- $\mathcal{P}$ *is a set of* possibilities*, which are each of the form* $(P, \phi, \mathcal{A}, \mathcal{X}, \alpha, \delta)$*, where* P *is a process,* ϕ *is a* Σ_0*-formula,* $\mathcal{A}$ *is a FLIC,* $\mathcal{X}$ *is a disequalities formula,* α *is a* Σ_0*-formula called* partial payload*, and* δ *is a sequence of memory updates of the form* $\mathsf{cell}(s) := t$ *for messages* s *and* t*;*
- *Checked is a set of pairs* (l, r)*, where* l *is a label and* r *is a recipe.*

where disequalities formulas are of the following form:

$$\begin{array}{lcll} \mathcal{X} & := & \mathcal{X} \wedge \mathcal{X} \mid \forall \bar{X}.\ \neg \mathcal{X}_0 & \textit{Disequalities formula} \\ \mathcal{X}_0 & := & \mathcal{X}_0 \wedge \mathcal{X}_0 \mid t \doteq t & \textit{Equalities formula} \end{array}$$

A symbolic state is finished *iff all the processes in* $\mathcal{P}$ *are* 0*.*

We may write $\mathcal{S}[e \leftarrow e']$ to denote the symbolic state identical to $\mathcal{S}$ except that e is replaced with e'.

We have augmented the FLICs $\mathcal{A}_i$ here with disequalities $\mathcal{X}_i$, i.e., negated equality constraints, which allows us to restrict the choices of the intruder in a symbolic state. This is needed when we want to make a split between the case that the intruder makes a particular choice and the case that they choose anything else. This is formalized in the following definition of applying a recipe substitution which is only possible when all the respective $\mathcal{X}_i$ are consistent with it:

Definition IV.2 (Choice of recipes for a symbolic state)**.** *Let* $\mathcal{S} = (\_, \_, \mathcal{P}, \mathit{Checked})$ *be a symbolic state and* ρ *be a recipe substitution. We say that* ρ *is a* choice of recipes for $\mathcal{S}$ *iff* ρ *is a choice of recipes for all FLICs in* $\mathcal{P}$ *and for every FLIC* $\mathcal{A}$ *and associated disequalities* $\mathcal{X}$ *in* $\mathcal{P}$*, the formula* $\sigma_\rho^{\mathcal{A}}(\mathcal{X})$ *is satisfiable, i.e.,* ρ *does not contradict the disequalities attached to any FLIC. Moreover, we define*

$$\begin{aligned} \rho(\mathcal{P}) &= \{(\sigma_\rho^{\mathcal{A}}(P), \phi, \rho(\mathcal{A}), \sigma_\rho^{\mathcal{A}}(\mathcal{X}), \alpha, \sigma_\rho^{\mathcal{A}}(\delta)) \mid \\ &\qquad (P, \phi, \mathcal{A}, \mathcal{X}, \alpha, \delta) \in \mathcal{P}\} \\ \rho(\mathit{Checked}) &= \{(l, \rho(r)) \mid (l, r) \in \mathit{Checked}\} \\ \rho(\mathcal{S}) &= \mathcal{S}[\mathcal{P} \leftarrow \rho(\mathcal{P}), \mathit{Checked} \leftarrow \rho(\mathit{Checked})] \end{aligned}$$

When writing $\rho(\mathcal{S})$ in the following, we implicitly assume that all disequalities in $\mathcal{S}$ are satisfiable under ρ, and that $\rho(\mathcal{S})$ is discarded otherwise. To decide whether disequality $\mathcal{X}$ is satisfiable it suffices to replace the free variables with distinct fresh constants and check that the corresponding unification problems have no solution. Moreover, we will always use the lazy intruder in the context of a symbolic state, so we further assume that $LI(\cdot, \cdot)$ only returns choices of recipes for the current symbolic state, i.e., excluding any ρ that would contradict a disequality.

From a symbolic state we can define all the choices of recipes (instantiations of the recipe and intruder variables) for the messages sent by the intruder and all the concrete executions (instantiations of privacy variables) that the intruder considers possible. A symbolic state represents a set of ground states, where each ground state corresponds to one multi message-analysis problem. For every ground state, the common payload α_0 is augmented with the partial payload α_i released by the corresponding possibility. Moreover, every model γ of the privacy variables needs to be augmented by the interpretation of relation symbols. In our approach, we assume that the protocol specification contains a fixed interpretation of the relation symbols, formalized as a Σ_0-formula γ_0.

Meta-notation. In the specification of transactions, we allow in formulas released the use of the *meta-notation* $\gamma(t)$ for a message t: Recall that in every ground state, the real values of privacy variables is defined by a ground interpretation γ. Thus, for instance, releasing $\star\ x \doteq \gamma(x)$ means allowing the intruder to learn the true value of x. In the symbolic execution for ground states, the meta-notation can be implemented by using γ as a substitution before adding the formula to α.

Example IV.1. In Example II.1, in case $x \doteq \mathsf{i}$, then the intruder can decrypt the message and observe what was the decision. Thus they would learn both that $x \doteq \mathsf{i}$ as well as the value of y (i.e., they know the server's decision). This leads to a privacy violation, unless we "declassify" x and y with a release. If x is the intruder, we can release $\star\ x \doteq \gamma(x) \wedge y \doteq \gamma(y)$. Releasing this information is still not enough because in case $x \not\doteq \mathsf{i}$ the intruder can also deduce that; so we additionally need to release $\star\ x \not\doteq \mathsf{i}$ in that case to remove the privacy violation. $\triangleleft$

In a symbolic state, however, there is no γ since the symbolic state represents all possible γ at once. Hence, in order to define the semantics, we need to resolve the meta-notation that we allow in the α_i. Given α_i and the truth γ, let $[\alpha_i]^\gamma$ be the instantiation of the meta-notation in α_i, i.e., replacing every occurrence of a term $\gamma(x)$ in α_i (for a variable

x) with the actual value of x in the given γ. For instance, if $\gamma(x) = \mathsf{i}$, then $[x \doteq \gamma(x)]^\gamma = x \doteq \mathsf{i}$.

Definition IV.3 (Semantics of symbolic states)**.** *Let $\mathcal{S} = (\alpha_0, \beta_0, \mathcal{P}, \_)$ be a finished symbolic state. The ground states represented by $\mathcal{S}$ are given by*

$$\begin{aligned}\llbracket \mathcal{S} \rrbracket = \{(\alpha_0 \wedge [\alpha_i]^\gamma, \beta_0, \gamma, \rho(\mathcal{P})) \mid\ & (0, \phi_i, \mathcal{A}_i, \_, \alpha_i, \_) \in \mathcal{P}, \\ & \rho \text{ is a ground choice of recipes for } \mathcal{S}, \\ & \gamma \text{ is a model of } \alpha_0 \wedge \beta_0 \wedge \gamma_0 \wedge \phi_i\},\end{aligned}$$

where $\rho(\mathcal{P})$ returns possibilities of the form $(0, \phi_j, struct_j, \delta_j)$, i.e., the additional components of symbolic possibilities are dropped because they are irrelevant for ground states (note that the α_i have already been used as part of the payload α); moreover, the possibility for which $\gamma \models \phi_i$ is underlined.

We say that a symbolic state $\mathcal{S}$ satisfies privacy *iff every ground state $S \in \llbracket \mathcal{S} \rrbracket$ satisfies privacy.*

Remark. Given a symbolic state $\mathcal{S} = (\alpha_0, \beta_0, \mathcal{P}, \_)$ and a possibility with formula ϕ_i. If $\alpha_0 \wedge \beta_0 \wedge \gamma_0 \wedge \phi_i$ is unsatisfiable, then the possibility can be removed from $\mathcal{P}$, as it corresponds to no ground state. In our procedure, we discard such possibilities whenever a transition is taken. $\lhd$

When computing the mgu between messages or solving constraints with the lazy intruder rules, we may deal with substitutions that contain both privacy and intruder variables. However, it is important to remember that the instantiation of privacy variables does not depend on the intruder, it is actually the goal of the intruder to learn about the privacy variables. On the other hand, intruder variables are instantiated according to the recipes chosen by the intruder. Thus, we distinguish substitutions that only substitute privacy variables.

Definition IV.4 (Privacy substitution)**.** *Given a substitution σ, the predicate $isPriv$ is defined as: $isPriv(\sigma)$ iff $dom(\sigma) \subseteq \mathcal{V}_{privacy}$. Moreover, define $isPriv(\bot) = \mathsf{false}$.*[2]

The intruder can make experiments on their knowledge by comparing the outcome of two recipes in every FLIC. It can happen that a pair of recipes gives the same message in one FLIC and different messages in another FLIC, allowing conclusions about the respective ϕ_i. In §V, we show how to extract all these conclusions and obtain a set of symbolic states in which every experiment either gives the same result in all FLICs or different results in all FLICs. This is formalized in the following equivalence relation between recipes:

Definition IV.5. *Let $\mathcal{S} = (\alpha_0, \beta_0, \mathcal{P}, \_)$ be a symbolic state with $\mathcal{P} = \{(\_, \phi_1, \mathcal{A}_1, \_, \_, \_), \ldots, (\_, \phi_n, \mathcal{A}_n, \_, \_, \_)\}$. Let r_1 and r_2 be two recipes and $\sigma_i = mgu(\mathcal{A}_i\{\!\!\{ r_1 \}\!\!\} \doteq \mathcal{A}_i\{\!\!\{ r_2 \}\!\!\})$ ($i \in \{1, \ldots, n\}$). We define $r_1 \simeq r_2$ iff $r_1 \sqsupset\!\!\sqsubset r_2$ or $r_1 \bowtie r_2$, where*

$$\begin{aligned} r_1 \sqsupset\!\!\sqsubset r_2 \quad & \textit{iff} \quad \textit{for every } i \in \{1, \ldots, n\}, \\ & \qquad isPriv(\sigma_i) \textit{ and } \alpha_0 \wedge \beta_0 \wedge \phi_i \models \sigma_i \\ r_1 \bowtie r_2 \quad & \textit{iff} \quad \textit{for every } i \in \{1, \ldots, n\},\ LI(\mathcal{A}_i, \sigma_i) = \emptyset \\ & \qquad \textit{or } (isPriv(\sigma_i) \textit{ and } \alpha_0 \wedge \beta_0 \wedge \phi_i \models \neg\sigma_i) \end{aligned}$$

Intuitively, $r_1 \sqsupset\!\!\sqsubset r_2$ means that the two recipes produce the same message in every FLIC. Conversely, $r_1 \bowtie r_2$ means that the two recipes produce different messages in every FLIC, under any possible instantiation of the variables: either the unifier depends on intruder variables but the intruder cannot solve the constraints in any way, or the unifier depends only on privacy variables and its instances are already excluded by the intruder reasoning.

Example IV.2. Based on Example II.1, suppose that we reached the following symbolic state containing two possibilities with $\phi_1 \equiv y \doteq \mathsf{yes}$, $\phi_2 \equiv y \neq \mathsf{yes} \wedge x \neq \mathsf{a}$ and

$$\begin{aligned}\mathcal{A}_1 &= +R \mapsto N.-l \mapsto \mathsf{crypt}(\mathsf{pk}(x), \mathsf{pair}(\mathsf{yes}, N)) \\ \mathcal{A}_2 &= +R \mapsto N.-l \mapsto \mathsf{crypt}(\mathsf{pk}(x), \mathsf{no})\end{aligned}$$

Here we again assume non-randomized encryption for the sake of the example. Then we have $l \bowtie \mathsf{crypt}(\mathsf{pk}(\mathsf{a}), \mathsf{no})$, because in $\mathcal{A}_1$ there is no unifier and in $\mathcal{A}_2$ the unifier $[x \mapsto \mathsf{a}]$ is excluded by ϕ_2. $\lhd$

We define well-formed symbolic states, where in particular what has been checked cannot distinguish the possibilities.

Definition IV.6 (Well-formed symbolic state)**.** *Let $\mathcal{S} = (\alpha_0, \beta_0, \mathcal{P}, Checked)$ be a symbolic state, with the possibilities $\mathcal{P} = \{(\_, \phi_1, \mathcal{A}_1, \mathcal{X}_1, \alpha_1, \_), \ldots, (\_, \phi_n, \mathcal{A}_n, \mathcal{X}_n, \alpha_n, \_)\}$. We say that $\mathcal{S}$ is* well-formed *iff*

- *the ϕ_i are such that $\models \neg(\phi_i \wedge \phi_j)$ for $i \neq j$, $fv(\phi_i) \subseteq fv(\alpha_0) \cup fv(\beta_0)$ and $\alpha_0 \wedge \beta_0 \models \bigvee_{i=1}^n \phi_i$;*
- *the $\mathcal{A}_i$ are simple FLICs with the same labels and same recipe variables, occurring in the same order;*
- *the disequalities $\mathcal{X}_i$ are satisfiable;*
- *the α_i are such that $fv(\alpha_i) \subseteq fv(\alpha_0)$ and $\alpha_0 \wedge \beta_0 \wedge \gamma_0 \wedge \phi_i \models \alpha_i$; and*
- *for every $(l, r) \in Checked$, we have $l \simeq r$.*

Recipe variables can only occur in the FLICs $\mathcal{A}_i$. Since $dom(\mathcal{A}_1) = \cdots = dom(\mathcal{A}_n)$, we may write $dom(\mathcal{S})$ for the domain of the symbolic state.

The initially empty set $Checked$ keeps track of which experiments the intruder has performed (cf. §V) and well-formedness requires that these experiments indeed no longer distinguish the possibilities. We now define a set of experiments $Pairs(\mathcal{S})$ that will be relevant: for every label l in the state and every FLIC $\mathcal{A}$, we try any other way to construct $\mathcal{A}\{\!\!\{ l \}\!\!\}$ (except l). To that end, we use the lazy intruder to solve the constraint $\mathcal{A}.{+R} \mapsto \mathcal{A}\{\!\!\{ l \}\!\!\}$ for a fresh recipe variable R. For each solution ρ, the experiment is the pair $(l, \rho(R))$:

[2]In our procedure, we will apply $isPriv$ to mgus, which can be either substitutions or $\bot$.

Definition IV.7 (Pairs and normal symbolic state)**.** *Let $\mathcal{S} = (\_, \_, \mathcal{P}, \mathit{Checked})$ be a symbolic state. The set of pairs of recipes to compare in $\mathcal{S}$ is*

$$\begin{aligned} \mathit{Pairs}(\mathcal{S}) = \{(l, \rho(R)) \mid\ & l \in \mathit{dom}(\mathcal{S}), (\_, \_, \mathcal{A}, \_, \_, \_) \in \mathcal{P}, \\ & \rho \in \mathit{LI}(\mathcal{A}.{+}R \mapsto \mathcal{A}\{\!\!\{ l \}\!\!\}, \varepsilon), \rho(R) \neq l\} \\ & \setminus \mathit{Checked} \end{aligned}$$

We say that $\mathcal{S}$ is normal *iff $\mathcal{S}$ is finished and $\mathit{Pairs}(\mathcal{S}) = \emptyset$.*

In a normal symbolic state, there are no more pairs of recipes that could distinguish the possibilities (they have all been checked). Thus, given a ground choice of recipes, all the concrete instantiations of frames are statically equivalent.

Lemma IV.1. *Let $\mathcal{S} = (\alpha_0, \beta_0, \_, \_)$ be a normal symbolic state, where the possibilities have conditions $\phi_1, \dots, \phi_n$ and FLICs $\mathcal{A}_1, \dots, \mathcal{A}_n$. Let $S \in [\![\mathcal{S}]\!]$, ρ_0 be the ground choice of recipes defining S and concr be the concrete frame in S. Let $\theta \models \alpha_0 \wedge \beta_0 \wedge \phi_i$ for some $i \in \{1, \dots, n\}$ and $\mathit{concr}' = \theta(\rho_0(\mathcal{A}_i))$. Then $\mathit{concr} \sim \mathit{concr}'$.*

The idea is now that in a normal symbolic state, the FLICs do not contain any more insights for the intruder, and all remaining violations of (α, β)-privacy can only result from any other information β_0 that the intruder has gathered. We thus define that a symbolic state is consistent iff β_0 cannot lead to violations either:

Definition IV.8 (Consistent symbolic state)**.** *We say that a finished symbolic state $\mathcal{S}$ is* consistent *iff (α, β_0)-privacy holds for every $(\alpha, \beta_0, \_, \_) \in [\![\mathcal{S}]\!]$.*

Remark. By construction, β_0 can only contain symbols in Σ_0. Even though $[\![\mathcal{S}]\!]$ is infinite, we need to consider only finitely many (α, β_0) pairs. This is because the corresponding α and β_0 in $\mathcal{S}$ do not contain intruder variables and we only need to resolve the meta-notation if present. For truth γ, we also have only to consider finitely many instances of the privacy variables (as they range over finite domains). For each α and β_0, the Σ_0-models are computable as we show in Appendix C. While that algorithm is based on an enumeration of models as a simple means to prove we are in a decidable fragment, our prototype tool uses the SMT solver cvc5 to check consistency more efficiently. $\lhd$

Example IV.3. Let us consider again Example II.1, where for now we assume that encryption is not randomized. Let $\mathcal{S} = (\alpha_0, \beta_0, \mathcal{P}, \emptyset)$ be the symbolic state such that:

$$\begin{aligned} \alpha_0 &\equiv x \in \mathsf{Agent} \wedge y \in \{\mathsf{yes}, \mathsf{no}\} \\ \beta_0 &\equiv y \doteq \mathsf{yes} \vee y \not\doteq \mathsf{yes} \\ \mathcal{P} &= \{(0, y \doteq \mathsf{yes}, \mathcal{A}_1, \mathsf{true}, \mathsf{true}, []), \\ &\qquad (0, y \not\doteq \mathsf{yes}, \mathcal{A}_2, \mathsf{true}, \mathsf{true}, [])\} \\ \mathcal{A}_1 &= {+}R \mapsto N.{-}l \mapsto \mathsf{crypt}(\mathsf{pk}(x), \mathsf{pair}(\mathsf{yes}, N)) \\ \mathcal{A}_2 &= {+}R \mapsto N.{-}l \mapsto \mathsf{crypt}(\mathsf{pk}(x), \mathsf{no}) \end{aligned}$$

Since there is no release in either possibility, we have that $\mathcal{S}$ is consistent iff (α_0, β_0)-privacy holds, i.e., iff for every $\mathcal{I} \models x \in \mathsf{Agent} \wedge y \in \{\mathsf{yes}, \mathsf{no}\}$, also $\mathcal{I} \models y \doteq \mathsf{yes} \vee y \not\doteq \mathsf{yes}$. This clearly holds, so $\mathcal{S}$ is consistent.

Note that if the intruder makes the experiment, e.g., of comparing l and $\mathsf{crypt}(\mathsf{pk}(\mathsf{a}), \mathsf{no})$ and considers the states where the recipes produce different messages, we would have $y \not\doteq \mathsf{yes} \wedge x \not\doteq \mathsf{a}$ for the second possibility and the symbolic state would then not be consistent (same payload but the new β_0 rules out the model $[x \mapsto \mathsf{a}, y \mapsto \mathsf{no}]$). $\lhd$

In a symbolic state that is both normal *and* consistent, we can combine the two properties to define, for each ground state in the semantics and model of the payload, a model of the full β and not just β_0, using the static equivalence between concrete frames. Thus, to verify whether a normal symbolic state satisfies privacy, it suffices to verify consistency.

Theorem IV.2. *Let $\mathcal{S}$ be a normal symbolic state. Then $\mathcal{S}$ satisfies privacy iff $\mathcal{S}$ is consistent.*

V. The Intruder Experiments

An *intruder experiment* is to compare pairs of recipes and the messages they produce in every frame: in a ground state, the intruder can check whether two messages are equal in the frame concr. In a symbolic state, each possibility considered by the intruder contains a different simple FLIC. When doing the comparison on the FLICs, the intruder may find out equalities that must hold (constraints on privacy and intruder variables) for messages to be equal. The intruder considers in separate symbolic states the possibilities where the two concrete messages are equal, and the possibilities where they are not. The result of such experiments can provide information about the values of privacy variables. Instead of comparing two arbitrary recipes, for every message t received, the intruder can try to compose t in a different way. We call these experiments *compose-checks*. We define a reduction relation $\rightarrowtail$ on symbolic states. Similarly to the lazy intruder rules, the idea is that $\rightarrowtail$ is Noetherian, but not confluent, and a symbolic state that cannot be reduced further is normal.

Definition V.1 (Compose-checks)**.** *The relation $\rightarrowtail$ is a binary relation on finished symbolic states. Let $\mathcal{S}$ be a symbolic state $(\_, \beta_0, \mathcal{P}, \mathit{Checked})$, with the possibilities $\mathcal{P} = \{(0, \phi_1, \mathcal{A}_1, \mathcal{X}_1, \alpha_1, \delta_1), \dots, (0, \phi_n, \mathcal{A}_n, \mathcal{X}_n, \alpha_n, \delta_n)\}$.*

Privacy split *When the intruder compares the messages produced by a label l and a recipe r, the messages may be equal under some unifiers, which depend only on privacy variables or which require a choice of recipes that has already been excluded. The formula β_0 is updated by considering in one symbolic state that the messages are equal ($l \sqsubset\!\!\sqsupset r$) and in the other symbolic state that the messages are unequal ($l \bowtie r$).*

$$\begin{aligned} \mathcal{S} \rightarrowtail \mathcal{S}[&\beta_0 \leftarrow \beta_0 \wedge \bigwedge_{i=1}^{n} \left(\phi_i \Rightarrow \begin{cases} \sigma_i & \mathit{if}\ \mathit{isPriv}(\sigma_i) \\ \mathsf{false} & \mathit{otherwise} \end{cases} \right) \\ &\mathcal{P} \leftarrow \{(0, \phi_i \wedge \sigma_i, \mathcal{A}_i, \mathcal{X}_i, \alpha_i, \delta_i) \mid \\ &\qquad i \in \{1, \dots, n\}, \mathit{isPriv}(\sigma_i)\} \\ &\mathit{Checked} \leftarrow \mathit{Checked} \cup \{(l, r)\}] \end{aligned}$$

$$\mathcal{S} \rightarrowtail \mathcal{S}[\beta_0 \leftarrow \beta_0 \wedge \bigwedge_{i=1}^{n} \left(\phi_i \Rightarrow \begin{cases} \neg\sigma_i & \textit{if isPriv}(\sigma_i) \\ \mathsf{true} & \textit{otherwise} \end{cases} \right)$$
$$\mathcal{P} \leftarrow \{(0, \phi_i \wedge \neg\sigma_i, \mathcal{A}_i, \mathcal{X}_i, \alpha_i, \delta_i) \mid i \in \{1, \dots, n\}, \textit{isPriv}(\sigma_i)\} \cup \{(0, \phi_i, \mathcal{A}_i, \mathcal{X}_i, \alpha_i, \delta_i) \mid i \in \{1, \dots, n\}, \textit{not isPriv}(\sigma_i)\}$$
$$\textit{Checked} \leftarrow \textit{Checked} \cup \{(l, r)\}]$$

if $(l, r) \in \textit{Pairs}(\mathcal{S})$ *and for every* $i \in \{1, \dots, n\}$*,* $\textit{isPriv}(\sigma_i)$ *or* $\textit{LI}(\mathcal{A}_i, \sigma_i) = \emptyset$*, where* $\sigma_i = \textit{mgu}(\mathcal{A}_i\{\!|\, l \,|\!\} \doteq \mathcal{A}_i\{\!|\, r \,|\!\})$.

Recipe split *When the intruder compares the messages produced by a label* l *and a recipe* r*, the messages may be equal under some unifiers, which at least in one FLIC depend on intruder variables. Such a unifier makes* one *FLIC non-simple. For each lazy intruder result, there is one symbolic state in which the intruder takes a choice of recipes* ρ *and the whole symbolic state is updated accordingly. Additionally, there is one symbolic state in which the intruder chooses something else for the recipes so one unifier is excluded.*

$$\mathcal{S} \rightarrowtail \rho_1(\mathcal{S}), \dots, \mathcal{S} \rightarrowtail \rho_k(\mathcal{S}), \mathcal{S} \rightarrowtail \mathcal{S}[\mathcal{X}_i \leftarrow \mathcal{X}_i \wedge \neg\sigma_i]$$

if $(l, r) \in \textit{Pairs}(\mathcal{S})$ *and there exists* $i \in \{1, \dots, n\}$ *such that not* $\textit{isPriv}(\sigma_i)$ *and* $\textit{LI}(\mathcal{A}_i, \sigma_i) = \{\rho_1, \dots, \rho_k\}$*, where* $\sigma_i = \textit{mgu}(\mathcal{A}_i\{\!|\, l \,|\!\} \doteq \mathcal{A}_i\{\!|\, r \,|\!\})$.

Example V.1. The symbolic state $\mathcal{S}$ from Example IV.3 is not normal since, e.g., $(l, \mathsf{crypt}(\mathsf{pk}(\mathsf{a}), \mathsf{no})) \in \textit{Pairs}(\mathcal{S})$.

We can perform a compose-check, in this case by applying the privacy split rule. In $\mathcal{A}_1$ we have to unify $\mathsf{crypt}(\mathsf{pk}(x), \mathsf{pair}(\mathsf{yes}, N))$ and $\mathsf{crypt}(\mathsf{pk}(\mathsf{a}), \mathsf{no})$, which is not possible. In $\mathcal{A}_2$ we have to unify $\mathsf{crypt}(\mathsf{pk}(x), \mathsf{no})$ and $\mathsf{crypt}(\mathsf{pk}(\mathsf{a}), \mathsf{no})$, which gives the mgu $\sigma = [x \mapsto \mathsf{a}]$.

Then we get two symbolic states $\mathcal{S}_1$ and $\mathcal{S}_2$, which have the same α_0 as $\mathcal{S}$ but we update β_0 and $\mathcal{P}$. Moreover, in both $\mathcal{S}_1$ and $\mathcal{S}_2$ we have $\textit{Checked} = \{(l, \mathsf{crypt}(\mathsf{pk}(\mathsf{a}), \mathsf{no}))\}$.

$$\mathcal{S}_1 \quad \beta_0 \equiv (y \doteq \mathsf{yes} \vee y \not\doteq \mathsf{yes}) \wedge (y \doteq \mathsf{yes} \Rightarrow \mathsf{false}) \wedge (y \not\doteq \mathsf{yes} \Rightarrow x \doteq \mathsf{a})$$
$$\mathcal{P} = \{(0, y \not\doteq \mathsf{yes} \wedge x \doteq \mathsf{a}, \mathcal{A}_2, \mathsf{true}, \mathsf{true}, [])\}$$
$$\mathcal{S}_2 \quad \beta_0 \equiv (y \doteq \mathsf{yes} \vee y \not\doteq \mathsf{yes}) \wedge (y \doteq \mathsf{yes} \Rightarrow \mathsf{true}) \wedge (y \not\doteq \mathsf{yes} \Rightarrow x \not\doteq \mathsf{a})$$
$$\mathcal{P} = \{(0, y \doteq \mathsf{yes}, \mathcal{A}_1, \mathsf{true}, \mathsf{true}, []), (0, y \not\doteq \mathsf{yes} \wedge x \not\doteq \mathsf{a}, \mathcal{A}_2, \mathsf{true}, \mathsf{true}, [])\} \quad \lhd$$

Using the compose-checks, we can transform a symbolic state into a set of normal symbolic states, since by definition a symbolic state is normal when there are no more pairs to compare. Moreover, the compose-checks preserve the semantics of symbolic states by partitioning the ground states represented.

Theorem V.1 (Compose-check correctness)**.** *Let* $\mathcal{S}$ *be a finished symbolic state,* $(l, r) \in \textit{Pairs}(\mathcal{S})$ *and* $\{\mathcal{S}_1, \dots, \mathcal{S}_n\}$ *be the symbolic states after one rule application given the pair* (l, r)*. Then* $[\![\mathcal{S}]\!] = \biguplus_{i=1}^{n} [\![\mathcal{S}_i]\!]$*, where* $\biguplus$ *denotes the disjoint union. Moreover, there is a finite number of* $\mathcal{S}'$ *such that* $\mathcal{S} \rightarrowtail^* \mathcal{S}'$ *and* $\mathcal{S}'$ *is normal.*

VI. Putting it All Together

We have so far just looked at a given symbolic state, how the intruder can solve constraints and make experiments on the FLICs — and that without destructors and algebraic properties. However, all important building blocks of the approach are now in place and we just have to use them.

We now first briefly summarize how all the rules defining the transitions from §II can be adapted to our symbolic representation. This gives us a decision procedure for a very restricted intruder model: where the intruder has no access to destructors. As a second step we then lift the entire approach to an intruder model with a destructor theory. This is quite economical as we do not have to integrate the destructor reasoning into the constraint solving and experiments.

We outline here only the most important points of the adaptation of the symbolic execution rules. The details are found in Appendix A. Let us follow the running example again. The non-deterministic choice is quite simple: instead of splitting into one successor state for each value in the domain, all these are handled in one symbolic state where we only add the domain constraint to α_0 or β_0, respectively. For a receiving step $\mathsf{rcv}(X)$, recall that the ground model has here an infinite branching over all the recipes that the intruder could use. This is the very reason for introducing the FLICs in the symbolic model: we simply choose a fresh recipe variable R and augment every FLIC with $+R \mapsto X$, saying that the intruder can choose any recipe R (over the labels of the FLIC so far) to form the input message X.

For conditions, we do not treat try as syntactic sugar in the symbolic approach. Consider a symbolic state $\mathcal{S}$ where one possibility has process $\mathsf{try}\ X \doteq d(t_1, t_2)\ \mathsf{in}\ P_1\ \mathsf{catch}\ P_2$, condition ϕ, and FLIC $\mathcal{A}$. We define below the precise class of algebraic theories we can support, but for now it suffices that there is only one rule for the destructor d, say $d(s_1, s_2) \to s_3$ where s_3 is a proper subterm of s_2 and let all variables in the s_i be renamed apart from the t_i and X. We compute the most general unifier σ for $s_1 \doteq t_1 \wedge s_2 \doteq t_2 \wedge s_3 \doteq X$ and use the lazy intruder to solve FLIC $\mathcal{A}$ for it: $\textit{LI}(\mathcal{A}, \sigma) = \{\rho_1, \dots, \rho_k\}$. Then we proceed with the following symbolic states $\mathcal{S}', \mathcal{S}_1, \dots, \mathcal{S}_k$, where $\mathcal{S}'$ is the symbolic state that results from replacing the try process with P_2 and adding the disequality $\forall \bar{Y}.\neg\sigma$ where $\bar{Y}$ are the intruder variables not bound by $\mathcal{A}$. This represents all grounds states where the intruder chose to send something for which the destructor would definitely fail. The other symbolic states $\mathcal{S}_i$ result from applying the choice of recipes ρ_i to $\mathcal{S}$, and then resolving the try as follows. With ρ_i the intruder has chosen messages that have the right structure for the destructor to succeed, but it may still depend on the privacy variables in general. For instance if the destructor is asymmetric description with key $\mathsf{inv}(\mathsf{pk}(\mathsf{a}))$ and the intruder chooses a message encrypted with $\mathsf{pk}(x)$, this succeeds iff $x \doteq \mathsf{a}$. Let σ_0 be the substitution on privacy variables under which the destructor works. We can replace the possibility in question by the following two: one where we set condition $\phi \wedge \sigma_0$ and go to process P_1; and one where we set condition $\phi \wedge \neg\sigma_0$ and go to process P_2. The if-then-else conditional is handled in a very similar way, obtaining a most general unifier σ under which the condition is true. (A condition composed with negation and disjunction can be first broken down into digestible pieces.) When the condition is a relation applied to some terms, the rule is the same as before: we simply split on whether the relation holds.

For releasing α information, recall that we have an α_i in each possibility that we can augment with the formula released. The rules for sending and 0 process require no changes. When all symbolic executions have terminated, we shall check whether the reached symbolic state satisfies privacy. For this, we apply the normalization procedure, i.e., perform all intruder experiments, and check consistency. Privacy is violated in the given symbolic state iff that check fails.

A. Lifting to Algebraic Properties

The above gives us a decision procedure for (α, β)-privacy (under a bound k on the number of transitions) as long as the intruder has no access to destructors. Note that transactions can apply destructors already. This allows for a very convenient and economical way to extend the intruder model with destructors as well without painfully extending all the above machinery to destructors: we define a set of special transactions called *destructor oracles*, one for each destructor. They receive a term and decryption key candidate, and send back the result of applying the destructor unless it fails. Note that these rules do not count towards the bound on the number of transitions, but rather we apply them to a reached symbolic state until destructors yield no further results.

1) The Supported Algebraic Theories: We have given in Fig. 1 a concrete example theory, but our result can be quite easily used for many similar theories. For instance, many modelers prefer for asymmetric cryptography that private keys are defined as atomic constants and the corresponding public key is obtained by a public function pub (so one can do without private functions). We like, in contrast, to start with public keys and have a private function inv to obtain the respective public key. This allows us to define a public function from agent names to public keys, which can be convenient in reasoning about privacy when the public-key infrastructure is fixed. Similarly, one may want to define further functions, in particular transparent functions like pair, i.e., functions that describe message serialization and where the intruder can extract every subterm. Finally, in some cases it is convenient to model some private extractor functions when we are dealing with messages where the recipient has to perform a small guessing attack. For instance, in a protocol like Basic Hash [22] (found also in our examples `basic_hash.nn`) the reader actually needs to try out every shared key with a tag to find out which tag it is. Rather than describing transitions that iterate over all tags and try to decrypt, it is convenient to model a private extract function that "magically" extracts the name of the tag, if the message is of the correct form, and returns false otherwise. This extraction must be a private function since the intruder should not be able to see this unless they know the respective shared keys; if they do, then the experiments in our method automatically allow the intruder to perform the guessing attack.

We thus distinguish three kinds of algebraic properties of destructors that can be used arbitrarily in our approach:

Definition VI.1 (Algebraic theory)**.** *A* constructor/destructor rule *is a rewrite rule of one of the following forms:*

- *Decryption:* $d(k, c(k', X_1, \ldots, X_n)) \to X_i$ *where* d *is a destructor symbol,* c *is a constructor symbol,* $i \in \{1, \ldots, n\}$*,* $fv(k) = fv(k')$ *and the* X_i *are variables.*
- *Transparency:* $d_i(c(X_1, \ldots, X_n)) \to X_i$ *where the* d_i *are destructors and* c *is a constructor* c *of arity* n*. We then say that* c *is* transparent.
- *Private extractors:* $d(c(t_1, \ldots, t_n)) \to t_0$ *where* d *is a* private *destructor,* c *is a constructor and* t_0 *is a subterm of one of the* t_i*.*

Let E *be a set of such rules, where we require that every destructor* d *occurs in exactly one rule of* E *and* E *forms a convergent term-rewriting system. Moreover, each constructor* c *cannot occur both in decryption and transparency rules.*

Define $\approx$ *to be the least congruence relation on ground terms such that*

$$d(k,t) \approx \begin{cases} t_i & \text{if } t \approx c(k', t_1, \ldots, t_n) \text{ and for some } \sigma, \\ & (d(k, c(k', t_1, \ldots, t_n)) \to t_i) \in \sigma(E) \\ \mathrm{ff} & \text{otherwise} \end{cases}$$

and for unary destructors the definition is the same but k *is omitted. Moreover, we require for every decryption rule* $d(k, c(k', X_1, \ldots, X_n)) \to X_i$ *that* $k = k'$ *or* $k \approx f(k')$ *or* $k' \approx f(k)$ *for some public function* f*.*

Remark. The requirement $k \approx f(k')$ or $k' \approx f(k)$ for some public f means that, given the decryption key k one can derive the encryption key k', or the other way around. In particular, in most asymmetric encryption schemes, the public key can be derived from the private key; for signatures the private key takes the role of the "encryption key". This requirement forces us to define in our example theory the rule $\mathsf{pubk}(\mathsf{inv}(k)) \to k$. Suppose that we omitted this rule, denying the intruder to derive the public key to a given private key. Suppose further that the intruder has received two messages $l_1 \mapsto \mathsf{inv}(\mathsf{pk}(x))$ and $l_2 \mapsto \mathsf{pk}(y)$ and is wondering whether maybe $x \doteq y$. Then they could make the experiment whether $\mathsf{dcrypt}(l_1, \mathsf{crypt}(l_2, m, r)) \approx \mathrm{ff}$ and this would be the case iff $x \not\doteq y$. For our method, we want however to ensure that the intruder never needs to decrypt messages that they encrypted themselves. In the example, with the public-key extraction rule, the intruder can derive $\mathsf{pubk}(\mathsf{inv}(\mathsf{pk}(x))) \approx \mathsf{pk}(x)$ and now directly compare this with l_2. The requirement allows us to show that the intruder cannot learn anything new from decrypting terms that they have encrypted themselves. $\triangleleft$

Observe that every ground term t is equivalent to a unique destructor-free ground term t_0 (that we call the $\approx$-normal form) and that can be computed by applying a rewrite rule, when possible, to an inner-most destructor[3] in t and replacing by ff if no rewrite rule is applicable, and repeating this until all destructors are eliminated.

2) Destructor Oracles: The idea is that transactions can already apply destructors and we can thus model oracles that provide decryption services for the intruder, namely the intruder has to provide a term to decrypt and the proposed decryption key and the oracle gives back the result of applying the destructor. More formally, given any decryption rule $(d(k, c(X_1, \ldots, X_n)) \to X_i) \in E$, we define the transaction

$$\mathsf{rcv}(X).\mathsf{rcv}(Y).\mathsf{try}\ Z \doteq d(X, Y)\ \mathsf{in}\ \mathsf{snd}(Z).\mathsf{snd}(X).0\ \mathsf{catch}\ 0$$

and call it the *destructor oracle* for said rewrite rule. For a transparent function of arity n, there is no need for a key and for each $i \in \{1, \ldots, n\}$, the ith subterm can be retrieved with destructor d_i, so we define one oracle per transparent function (returning all subterms) with the following transaction:

$$\mathsf{rcv}(Y).\mathsf{try}\ Z_1 \doteq d_1(Y)\ \mathsf{in}\ \ldots\ \mathsf{try}\ Z_n \doteq d_n(Y)\ \mathsf{in}$$
$$\mathsf{snd}(Z_1).\cdots.\mathsf{snd}(Z_n)\ \mathsf{catch}\ 0\ \ldots\ \mathsf{catch}\ 0$$

Finally, private extractors are not available to the intruder, anyway.

Obviously, such transactions are redundant if the intruder has access to the destructors and also it is sound to add such transactions. Also redundant is the output $\mathsf{snd}(X)$, because X is already an input, but this ensures that different ways of composing the key will be considered by our compose-checks.

The reader may wonder why we do not do the same also for constructors, e.g., $\mathsf{rcv}(X_1).\cdots.\mathsf{rcv}(X_n).\mathsf{snd}(c(X_1, \ldots, X_n))$, so we could use an intruder who neither encrypts nor decrypts and just uses oracles for both jobs. The reason is that constructors give rise to an infinite set of terms that can be generated and it is difficult to limit that—this is why we use the lazy intruder technique as a way to finitely represent the infinitely many choices in a finite and yet complete way. For destructors on the other hand, we do not have the same problem since it is limited what we can achieve here. In particular there is no need for the intruder to destruct terms that they have constructed themselves, thus allowing us to limit the use of destructors, respectively the destructor oracle rules, in a simple way.

3) Term Marking and Analysis Strategy: In general the oracle rules are applicable without boundary. We use a special strategy in which to apply them that does not lead into non-termination, but covers all applications that are necessary for any attack. Note also that the application of oracle rules does not count towards the bound on the number of transitions.

All received terms and subterms in a FLIC shall be marked with one of three possible markings: $\star$ for terms that may be decrypted but have not been; $+$ for terms that cannot be decrypted at the given intruder knowledge for any instance of the variables; and $\checkmark$ for terms that either have already been decrypted or have been composed by the intruder himself (so the intruder knows already the subterms that may result from a decryption). We call a symbolic state *analyzed* if it does not contain any $\star$-marked terms anymore.

Definition VI.2 (Term marking)**.** *We introduce first a marking for all terms that the intruder receives in a FLIC (i.e., that a label maps to) and their subterms. The default initial marking is $\star$, representing a term that can potentially be decomposed using the destructor oracles. The exceptions are privacy and intruder variables, as well as functions that do not have a public destructor; all such terms (and subterms if they have) are marked with $\checkmark$.*

We keep the marks throughout the state transition system, where marks can change according to the analysis strategy explained below. In particular, when a variable gets instantiated, the resulting term keeps its $\checkmark$ marking.

Besides $\star$ and $\checkmark$, we will also use the marking $+$ which represents that a term cannot presently be decomposed since the intruder currently does not know the decryption key, but may learn it later.

There is a strategy for applying the oracle rules to a given symbolic state $\mathcal{S}$ to obtain a finite set of analyzed symbolic states $\mathcal{S}_1, \ldots, \mathcal{S}_n$ that are together equivalent to $\mathcal{S}$ except that the FLICs are augmented with the results of decryptions, which we call *shorthands*.

Definition VI.3 (Destructor oracle application strategy)**.** *Let $\mathcal{S}$ be a normal symbolic state. (Recall that in $\mathcal{S}$ all FLICs are simple, and thus intruder variables represent messages the intruder composed; and $\mathcal{S}$ is normal, i.e., all compose-checks have been made.)*

We now define the following strategy that is applied as long as there is a label l that maps to a $\star$-marked term. Let l be the first label (in the order of the FLICs' domain) that maps to a $\star$-marked term $c(t_1, \ldots, t_n)$ in some FLIC; note that by construction, it can only be a constructor term. If c is an encryption and if $(d(k, c(t_1, \ldots, t_n)) \to t_i) \in \sigma(E)$ is an appropriate instance of a destructor rule (i.e., the intruder can decrypt iff they can produce k), then we apply the destructor oracle for that rule under the specialization that the recipe for Y (the oracle input for the constructor term) must be the label l. If c is a transparent function, then we use the appropriate oracle that applies all its destructors and returns all subterms.

Applying the oracle transaction leads to a finite number of successor states $\mathcal{S}_1, \ldots, \mathcal{S}_m$ (there is at least one, so $m \geq 1$) that are again normal and have simple FLICs. In each $\mathcal{S}_i$ the decryption has either worked in every FLIC, or failed in every FLIC. We now update the marks in the $\mathcal{S}_i$ as follows.

If $\mathcal{S}_i$ is a state where decryption has failed in every FLIC, assuming that c is the constructor for which we had attempted the destructor oracle rule, then in every FLIC where $l \mapsto c(t_1, \ldots, t_n)$ that is marked $\star$, we change to mark $+$ because

[3] Here by the expression "inner-most" we mean that no proper subterm has a destructor. This "call by value" reduction strategy is necessary as the following example shows: $\mathsf{dscrypt}(\mathsf{dscrypt}(k, c), \mathsf{scrypt}(\mathsf{dscrypt}(k, d), s, r)) \approx s$ and it is not equivalent to ff (which an "outer-most" or "call by name" strategy would produce), because $\mathsf{scrypt}(\mathsf{dscrypt}(k, d), s, r) \approx \mathsf{scrypt}(\mathsf{ff}, s, r) \approx \mathsf{scrypt}(\mathsf{scrypt}(k, c), s, r)$ and thus the outer-most destructor must result in s according to Definition VI.1. Also observe that at most one rewrite rule can be applied to an inner-most destructor subterm of t since E is convergent.

it is currently not decipherable. If it was already marked ✓, we do not change the label. (Note that in some FLIC, l may map to a term with a different constructor c'; if that term is marked $\star$, it maintains this marking, so that one of the next analysis steps will be to check if the respective destructor for c' can be applied.)

In an $\mathcal{S}_i$ where decryption has worked, we update and introduce markings in each FLIC as follows. If it was a decryption rule, and thus in a given FLIC, l maps to some term $c(k', t_1, \ldots, t_n)$, then the result of the analysis is bound to a new label $l' \mapsto t_i$ (for some $i \in \{1, \ldots, n\}$); the decryption key is bound to new label $l'' \mapsto k$. If m_i is the mark of t_i in l, then the new occurrence of t_i at l' shall also be marked with m_i. In turn, $c(k', t_1, \ldots, t_n)$ are now all marked ✓, because they are fully analyzed. Similarly the key term $l'' \mapsto k$ and all its subterms receive the ✓ mark, because they have been produced by the intruder already (and are thus taken from another label that is already analyzed, or composed by the intruder and thus not interesting for decryption). All labels that were marked $+$ are changed with marking $\star$, because the newly analyzed term may allow for some decryption that was impossible before. If the destructor is not a decryption but a transparency rule, the marking is similar for the new subterms.

We repeat this process of attempting to decrypt the first $\star$-marked term until there are no more $\star$-marks. A symbolic state is called analyzed *if it contains no more $\star$-marked terms.*

We also call a label l in a symbolic state $\mathcal{S}$ a shorthand*, if there exists a recipe r over labels before l such that $\mathcal{A}\{\!|\, l \,|\!\} \approx \mathcal{A}\{\!|\, r \,|\!\}$ for every FLIC $\mathcal{A}$ in $\mathcal{S}$. The destructor oracle application strategy augments FLICs only by shorthands and thus does not change what is derivable for an intruder who can decompose.*

Theorem VI.1 (Analysis correctness)**.** *For a symbolic state $\mathcal{S}$, the destructor oracle application strategy produces in finitely many steps a set $\{\mathcal{S}_1, \ldots, \mathcal{S}_n\}$ of symbolic states that are analyzed. Further, for every ground state $S \in [\![\mathcal{S}]\!]$ there exists $S' \in [\![\mathcal{S}_i]\!]$, for some $i \in \{1, \ldots, n\}$, such that S and S' are equivalent except that the frames in S' may contain further shorthands; and vice versa, for every $S' \in [\![\mathcal{S}_i]\!]$ there exists $S \in [\![\mathcal{S}]\!]$ such that S' is equivalent to S except for shorthands.*

For instance in the symbolic state reached after executing the transaction from Example II.1, there is one FLIC that contains a received message $-l \mapsto \mathsf{crypt}(\mathsf{pk}(x), \mathsf{pair}(\mathsf{yes}, N), r)$ (marked $\star$) as well as $-l_0 \mapsto \mathsf{inv}(\mathsf{pk}(\mathsf{i}))$, then the strategy will apply the oracle rule for asymmetric decryption for label l, and this gives two states $\mathcal{S}_1$ where for this possibility we unify $x \doteq \mathsf{i}$ and the intruder has a new label $-l_1 \mapsto \mathsf{pair}(\mathsf{yes}, N)$ and $\mathcal{S}_2$ where we have $x \not\doteq \mathsf{i}$ and the intruder cannot decrypt l (given the intruder knows no other private keys $\mathsf{inv}(\mathsf{pk}(\cdot))$). The encrypted message at label l is now marked ✓ in $\mathcal{S}_1$ and $+$ in $\mathcal{S}_2$. In analyzed states, the intruder does not need any destructors anymore:

Lemma VI.2. *Let $\mathcal{S}$ be a normal analyzed state, $S \in [\![\mathcal{S}]\!]$ and r be any recipe over the domain of S. Then there is a destructor-free recipe r' such that $struct\{\!|\, r \,|\!\} \approx struct\{\!|\, r' \,|\!\}$ in every frame $struct$ of S.*

In the definition of normal state, all destructors are private functions, so the intruder can only make experiments using destructor-free recipes. Call a state *normal w.r.t. arbitrary recipes* when we allow destructors in the experiments:

Lemma VI.3. *Let $\mathcal{S}$ be an analyzed state and normal. Then it is also normal w.r.t. arbitrary recipes.*

We can now conclude the correctness of our decision procedure. All the proofs are in Appendix B. Note that we need a bound on the number of transitions, and this bound is restricting the number of transactions that are executed. All "internal" transitions taken by our compose-checks and analysis steps do not count towards that bound.

Theorem VI.4 (Procedure correctness)**.** *Given a protocol specification for (α, β)-privacy, a bound on the number of transitions and an algebraic theory allowed by Definition VI.1, our decision procedure is sound, complete and terminating.*

VII. Tool support

We have developed a prototype tool called `noname` implementing our decision procedure. The tool is a proof-of-concept showing that automation for (α, β)-privacy is achievable and practical. The user must provide as input the protocol specification, consisting of the transactions that can be executed, and a bound on the number of transactions to execute. For the cryptographic operators, we make available by default primitives for asymmetric encryption/decryption, symmetric encryption/decryption, signatures and pairing (cf. Fig. 1). The user can define custom operators with the restriction to constructor/destructor theories (cf. Definition VI.1). We have also implemented an interactive running mode (the default is automatic, i.e., exploring of all reachable states) in which the user is prompted whenever there are multiple successor states, so that one can manually explore the symbolic transition system.

In case there is a privacy violation, the tool provides an attack trace that includes the sequence of atomic transactions executed and steps taken by the intruder (i.e., the recipes they have chosen) to reach an attack state, as well as a countermodel proving that the privacy goals in that state do not hold, i.e., a witness that the intruder has learned more in that state than what is allowed by the payload.

As case studies, we have focused on unlinkability goals: for the running example, we get a violation in presence of a corrupted agent. When permitting that in the corrupted case the intruder can learn the identity, the tool discovers another problem, namely that the intruder now also learns in the uncorrupted case that the involved agent is not corrupted. When releasing also that information, no more violations are found. This illustrates how the tool can help to discover all private information that is leaked, and thus either fix the protocol or permit that leak, and then finally verify that no additional information is leaked. We plan to strengthen the tool support further to make this exploration easier.

We also applied our approach to the Basic Hash [23] and the OSK [24] protocols, where OSK is particularly challenging as a stateful protocol. We have verified that the Basic Hash protocol satisfies unlinkability, but fails to provide forward privacy [22]. For the OSK protocol, we have modeled two variants where, respectively, no de-synchronization and one step de-synchronization is tolerated. For both versions the tool finds the known linkability flaws [25].

As further benchmarks we use the formalization in (α,β)-privacy of several variants of the BAC protocol by the ICAO [26] and the private authentication protocol by Abadi and Fournet [27] (denoted AF for short) that is found in [28]. For BAC, the tool finds the known problems in some implementations [29], [30], [31].

Table II gives an overview of the results of our tool. Finding a privacy violation is usually fast, because the tool stops as soon as it finds one without exploring the rest of the transition system. Most protocols take a few seconds to analyze, but when incrementing the bound on the number of transitions we can notice a steep increase in the verification time. Indeed, in our model, transactions can always be executed so there is in general a large number of possible interleavings. The tool seems thus to be limited by the substantial size of the search space, like earlier tools for deciding equivalence (APTE [10]). In our decision procedure, we are not deciding static equivalence between frames, but the experiments made by the intruder to try and distinguish the different possibilities seem to have a comparable complexity. For unlinkability goals, in particular, our tool and others (for bounded sessions) essentially provide similar privacy guarantees. We share the challenges and techniques such as symbolic representation of constraints for the unbounded intruder. Thus, we believe that optimizations implemented in tools such as DeepSec [32], e.g., forms of symmetries and partial order reductions, could be adapted to our decision procedure.

TABLE II
EVALUATION OF THE TOOL

Protocol	Bound	Result	Time
Runex	2	ϟ	0.35s
Runex (fix attempt)	2	ϟ	0.42s
Runex (fixed)	2	✔	0.45s
Basic Hash	4	✔	1.28s
Basic Hash (compromised tag)	2	ϟ	0.13s
OSK (no desynchronization)	3	ϟ	0.21s
OSK (1 desynchronization step)	4	ϟ	0.89s
BAC (different error messages)	3	ϟ	0.14s
BAC (same error message)	4	✔	0.62s
BAC (parallel)	4	✔	0.80s
BAC (sequential)	4	✔	0.82s
AF0	2	ϟ	0.98s
AF0 (fixed)	2	✔	3.14s
AF0 (fixed)	3	✔	3min52s
AF	2	✔	5.21s
AF	3	✔	8min41s

✔ = No violation, ϟ = Violation
Machine used: laptop with i7-4720HQ @ 2.60GHz, 8GB RAM
GHC 9.6.2, cvc5 1.0.8

VIII. RELATED AND FUTURE WORK

It is a striking parallel between (α,β)-privacy and equivalence-based privacy models that the vast amount of possibilities leads to very high complexity for procedures, see, e.g., [16]. In equivalence-based approaches, the underlying problem is the static equivalence of (concrete) frames, representing two possible intruder knowledges. In (α,β)-privacy, we have instead the multi message-analysis problem: there is just one concrete frame $concr$, the observed messages, and one or more $struct_i$ that result from a symbolic execution of the transactions by the intruder, where the privacy variables are not instantiated. Each possibility has a corresponding condition ϕ_i, exactly one of which is actually true, and the intruder knows that $concr$ is an instance of the corresponding $struct_i$, i.e., under the true instance of the privacy variables, $concr \sim struct_i$ for the true ϕ_i. Thus, evaluating the static equivalence can exclude several instantiations of privacy variables (even if there is just one $struct$) or rule out an entire possibility ϕ_i. The methods for solving these two problems bear many similarities, in particular one essentially in both cases looks for a pair of recipes that distinguishes the frames, i.e., the experiments that the intruder can do on their knowledge.

Like many other tools for a bounded number of sessions such as APTE [10] and DeepSec [6], we also use the symbolic representation of the lazy intruder, using variables for messages sent by the intruder that are instantiated only in a demand driven way when solving intruder constraints, turning frames into FLICs. This makes the frame distinction problems a magnitude harder (see for instance [33]). In recipes we have to also take into account variables that represent what the intruder has sent earlier and the actual choice may allow for different experiments now. We tackle this problem by first considering a model where the intruder cannot use destructors. It suffices then to check only if any message in any $struct_i$ can be composed in a different way, which in turn can be solved with intruder constraint solving. This is the idea behind the notion of a *normal* state, i.e., where all said experiments have been done, and we can thus check if the results of the experiments exclude any model of α.

What makes the handling of destructors relatively easy is our requirement that all destructors yield a subterm or $\mathtt{ff}$, which the intruder and honest agents can see. Thus we have no problem with "garbage terms" like decryption of a nonce. This allows us to show that it is sufficient that the intruder has applied destructors as far as possible to their knowledge using the oracles — the notion of an *analyzed knowledge*: for any recipe that contains destructors, there is an equivalent recipe that uses the result of a destructor oracle.

One may wonder if a procedure for an unbounded number of steps is possible. If we look at the equivalence-based approaches, it seems the best option for this is the notion of diff-equivalence [16], [8] as used in ProVerif [4] and Tamarin [15]. Roughly speaking, diff-equivalence sidesteps the problem of the intruder's uncertainty in branching by requiring that the conditions are either true in both executions or both false. This seems to correspond to the restriction in (α, β)-privacy that the intruder can always observe whether a condition was true or false, and we thus have just one $struct_i$ in each state. We are currently investigating whether this can allow for a unbounded-step procedure similar to ProVerif for (α, β)-privacy. Again it is a striking similarity with equivalence-based approaches that one may either need a tight bound on the number of transitions or substantial restrictions on the processes one can model.

The main difference with other tools is in the properties being verified. Our tool looks at the reachable states from an (α, β)-privacy specification of a protocol, and the privacy goals are constructed by the tool when exploring the transition system. Instead of verifying whether a number of properties hold, we thus verify whether the intruder is ever able to learn more than the information allowed (payload α). One advantage is that, in case of successful verification, we ensure that the intruder cannot learn *anything more* (about the privacy variables) than what the protocol is intentionally releasing. For some protocols such as AF, we believe that a characterization of the privacy goals with (α, β)-privacy can give a better understanding of which guarantees the protocol actually provides, as we do not see an obvious way of expressing all the privacy goals with equivalences between processes.

References

[1] S. Mödersheim and L. Viganò, "Alpha-beta privacy," *ACM Trans. Priv. Secur.*, vol. 22, no. 1, pp. 1–35, 2019.

[2] S. Gondron, S. Mödersheim, and L. Viganò, "Privacy as reachability," in *CSF 2022*. IEEE, 2022, pp. 130–146.

[3] L. Fernet and S. Mödersheim, "Deciding a fragment of (alpha, beta)-privacy," in *STM*, ser. LNCS, vol. 13075. Springer, 2021, pp. 122–142.

[4] B. Blanchet, "An efficient cryptographic protocol verifier based on Prolog rules," in *CSFW 2001*. IEEE, 2001, pp. 82–96.

[5] B. Blanchet, M. Abadi, and C. Fournet, "Automated verification of selected equivalences for security protocols," *J Log Algebr Program*, vol. 75, no. 1, pp. 3–51, 2008.

[6] V. Cheval, S. Kremer, and I. Rakotonirina, "DEEPSEC: Deciding equivalence properties in security protocols theory and practice," in *SP 2018*. IEEE, 2018, pp. 529–546.

[7] D. Aparicio-Sánchez, S. Escobar, R. Gutiérrez, and J. Sapiña, "An optimizing protocol transformation for constructor finite variant theories in Maude-NPA," in *ESORICS 2020*, ser. LNCS, vol. 12309. Springer, 2020, pp. 230–250.

[8] V. Cheval, S. Kremer, and I. Rakotonirina, "The hitchhiker's guide to decidability and complexity of equivalence properties in security protocols," in *Logic, Language, and Security*, ser. LNCS. Springer, 2020, vol. 12300, pp. 127–145.

[9] M. Rusinowitch and M. Turuani, "Protocol insecurity with a finite number of sessions and composed keys is NP-complete," *Theor. Comput. Sci.*, vol. 299, no. 1, pp. 451–475, 2003.

[10] V. Cheval, "APTE: An algorithm for proving trace equivalence," in *TACAS 2014*, ser. LNCS, vol. 8413. Springer, 2014, pp. 587–592.

[11] R. Chadha, V. Cheval, Ş. Ciobâcă, and S. Kremer, "Automated verification of equivalence properties of cryptographic protocols," *ACM Trans. Comput. Logic*, vol. 17, no. 4, pp. 1–32, 2016.

[12] A. Tiu and J. Dawson, "Automating open bisimulation checking for the spi calculus," in *CSF*. IEEE, 2010, pp. 307–321.

[13] A. Tiu, N. Nguyen, and R. Horne, "SPEC: An equivalence checker for security protocols," in *APLAS*, ser. LNCS, vol. 10017. Springer, 2016, pp. 87–95.

[14] H. Barbosa, C. W. Barrett, M. Brain, G. Kremer, H. Lachnitt, M. Mann, A. Mohamed, M. Mohamed, A. Niemetz, A. Nötzli, A. Ozdemir, M. Preiner, A. Reynolds, Y. Sheng, C. Tinelli, and Y. Zohar, "cvc5: A versatile and industrial-strength SMT solver," in *TACAS 2022*, ser. LNCS, vol. 13243. Springer, 2022, pp. 415–442.

[15] S. Meier, B. Schmidt, C. Cremers, and D. Basin, "The TAMARIN prover for the symbolic analysis of security protocols," in *CAV 2013*, ser. LNCS, vol. 8044. Springer, 2013, pp. 696–701.

[16] S. Delaune and L. Hirschi, "A survey of symbolic methods for establishing equivalence-based properties in cryptographic protocols," *J. Log. Algebraic Methods Program.*, vol. 87, pp. 127–144, 2017.

[17] V. Cheval and I. Rakotonirina, "Indistinguishability beyond diff-equivalence in ProVerif," in *CSF 2023*. IEEE, 2023, pp. 184–199.

[18] T. Hinrichs and M. Genesereth, "Herbrand logic," Stanford University, USA, Tech. Rep. LG-2006-02, 2006. [Online]. Available: http://logic.stanford.edu/reports/LG-2006-02.pdf

[19] J. Millen and V. Shmatikov, "Constraint solving for bounded-process cryptographic protocol analysis," in *CCS*. ACM, 2001, pp. 166–175.

[20] D. Basin, S. Mödersheim, and L. Viganò, "OFMC: A symbolic model checker for security protocols," *Int. J. Inf. Secur.*, vol. 4, no. 3, pp. 181–208, 2005.

[21] V. Cheval, H. Comon-Lundh, and S. Delaune, "A procedure for deciding symbolic equivalence between sets of constraint systems," *Inf Comput*, vol. 255, pp. 94–125, 2017.

[22] M. Brusó, K. Chatzikokolakis, and J. den Hartog, "Formal verification of privacy for RFID systems," in *CSF 2010*. IEEE, 2010, pp. 75–88.

[23] S. A. Weis, S. E. Sarma, R. L. Rivest, and D. W. Engels, "Security and privacy aspects of low-cost radio frequency identification systems," in *Security in Pervasive Computing*, ser. LNCS, vol. 2802. Springer, 2004, pp. 201–212.

[24] M. Ohkubo, K. Suzuki, and S. Kinoshita, "Cryptographic approach to "privacy-friendly" tags," in *RFID Privacy Workshop 2003*, 2003.

[25] D. Baelde, S. Delaune, and S. Moreau, "A method for proving unlinkability of stateful protocols," in *CSF 2020*. IEEE, 2020, pp. 169–183.

[26] ICAO, "Machine readable travel documents," Doc Series, Doc 9303, https://www.icao.int/publications/pages/publication.aspx?docnum=9303.

[27] M. Abadi and C. Fournet, "Private authentication," *Theor. Comput. Sci.*, vol. 322, no. 3, pp. 427–476, 2004.

[28] L. Fernet and S. Mödersheim, "Private authentication with alpha-beta-privacy," in *OID 2023*, ser. LNI. GI, 2023. [Online]. Available: http://imm.dtu.dk/~samo/alphabeta/

[29] M. Arapinis, T. Chothia, E. Ritter, and M. Ryan, "Analysing unlinkability and anonymity using the applied pi calculus," in *CSF 2010*. IEEE, 2010, pp. 107–121.

[30] T. Chothia and V. Smirnov, "A traceability attack against e-passports," in *FC 2010*, ser. LNCS, vol. 6052. Springer, 2010, pp. 20–34.

[31] I. Filimonov, R. Horne, S. Mauw, and Z. Smith, "Breaking unlinkability of the ICAO 9303 standard for e-passports using bisimilarity," in *ESORICS 2019*, ser. LNCS, vol. 11735. Springer, 2019, pp. 577–594.

[32] V. Cheval, S. Kremer, and I. Rakotonirina, "Exploiting symmetries when proving equivalence properties for security protocols," in *CCS 2019*. ACM, 2019, pp. 905–922.

[33] M. Baudet, "Deciding security of protocols against off-line guessing attacks," in *CCS*. ACM, 2005, pp. 16–25.

Appendix

A. Correctness of the Representation with Symbolic States

The authors of [2] define rules for the symbolic execution of transactions and explain how to define (α, β)-privacy as reachability, in a transition system with ground states. We follow a similar approach but we have two additional layers of symbolic representation, namely: the merging of ground states that differ only in the truth formula, and the intruder variables for the lazy intruder. We say that the rules from [2] are working on "the ground level", while the adapted rules from this paper are working on "the symbolic level".

On the ground level, there is always one possibility that is *marked* (underlined in the rules). The marked possibility corresponds to the concrete execution observed by the intruder.

On the symbolic level, there is no marked possibility because we actually represent together all different instantiations for the marked possibility. Our rules work so that each possibility "could" be the marked one, and which one is marked is defined in the semantics of the symbolic states (Definition IV.3).

We now argue that our rules on the symbolic level are correct w.r.t. the ground level, i.e., the symbolic states generated by our rules represent the ground states generated by the rules on the ground level. In the following, we recall the rules on the ground level and present our version of the rules next to each other.

We now define how to perform the symbolic execution of a number of transactions. Whenever an atomic transaction P is executed, we need to consider what can happen for the different possibilities in the symbolic state. This is done with evaluation rules that work out the steps of the processes. Once all the processes are 0, we have reached a finished symbolic state. The rules thus generate a transition system representing all the reachable states of the protocol.

Definition A.1 (Initial symbolic state)**.** *Let $\mathcal{S} = (\_, \_, \mathcal{P}, \_)$ be a finished symbolic state, P be a transaction process and σ be a substitution such that σ substitutes the variables in $n_1, \ldots, n_k$ (from a $\nu n_1, \ldots, n_k.P_r$ specification) with fresh and distinct constants from $\Sigma \setminus \Sigma_0$ that do not occur elsewhere in $\mathcal{S}$ or P, and such that σ substitutes all other variables with fresh variables that do not occur elsewhere. The* initial symbolic state for P w.r.t. $\mathcal{S}$ and σ *is*

$$\mathcal{S}[\mathcal{P} \leftarrow \{(\sigma(P), \phi, \mathcal{A}, \mathcal{X}, \alpha, \delta) \mid (0, \phi, \mathcal{A}, \mathcal{X}, \alpha, \delta) \in \mathcal{P}\}]$$

We write $init(P, \mathcal{S})$ and omit σ to denote an initial symbolic state, because the point is that all variables are substituted with fresh and distinct constants or fresh variables, so the actual values are not relevant.

The definition is the same for ground states, thus we also write $init(P, S)$ for a state S.

We define the rules by focusing on the set of possibilities $\mathcal{P}$, which contains the transaction to execute. The rules update the current (symbolic) state by updating $\mathcal{P}$, and the rest of the (symbolic) state is not changed unless explicitly stated. We use the symbol $\uplus$ to denote the disjoint union of sets. For simplicity, we ignore the redundancy rules defined in [2] in this paper: the redundant possibilities do not change the semantics of symbolic states, and we already said that they are discarded in the procedure.

1) Non-Deterministic Choice: All possibilities have this choice step at the same time. On the ground level, the variable x is chosen non-deterministically from the values in the domain D. There is a transition for every value $c \in D$ that the variable can take:

$$\begin{aligned}&\{(\mathsf{mode}\ x \in D.P_1, \phi_1, struct_1, \delta_1), \ldots,\\ &\ (\mathsf{mode}\ x \in D.P_n, \phi_n, struct_n, \delta_n)\}\\ &\ \to \{(P_1, \phi_1, struct_1, \delta_1), \ldots, (P_n, \phi_n, struct_n, \delta_n)\}\end{aligned}$$

where γ is augmented with $x \doteq c$, and if $\mathsf{mode} = \star$ (resp. $\mathsf{mode} = \diamond$) then α (resp. β_0) is augmented with $x \in D$.

On the symbolic level, we have a single transition and we only update α_0 or β_0 with the formula $x \in D$.

$$\begin{aligned}&\{(\mathsf{mode}\ x \in D.P_1, \phi_1, \mathcal{A}_1, \mathcal{X}_1, \alpha_1, \delta_1), \ldots,\\ &\ (\mathsf{mode}\ x \in D.P_n, \phi_n, \mathcal{A}_n, \mathcal{X}_n, \alpha_n, \delta_n)\}\\ &\ \Rightarrow \{(P_1, \phi_1, \mathcal{A}_1, \mathcal{X}_1, \alpha_1, \delta_1), \ldots, (P_n, \phi_n, \mathcal{A}_n, \mathcal{X}_n, \alpha_n, \delta_n)\}\end{aligned}$$

and if $\mathsf{mode} = \star$ (resp. $\mathsf{mode} = \diamond$) then α_0 (resp. β_0) is augmented with $x \in D$.

The semantics of the symbolic states include all models of $\alpha_0 \wedge \beta_0 \wedge \gamma_0 \wedge \phi_i$, so we represent all models $\gamma \models x \doteq c$ for every $c \in D$ (and such that γ is consistent with the rest of the formulas).

2) Receive: By construction, every possibility starts with a receive step (with the same variable). On the ground level, there is a transition for every recipe r that the intruder can generate, and the variable standing for the message received is directly substituted with what the recipe produces in each structural frame.

$$\begin{aligned}&\{\underline{(\mathsf{rcv}(X).P_1, \phi_1, struct_1, \delta_1)}, \ldots,\\ &\ (\mathsf{rcv}(X).P_n, \phi_n, struct_n, \delta_n)\}\\ &\ \to \{\underline{(P_1[X \mapsto struct_1\{\!|\, r \,|\!\}], \phi_1, struct_1, \delta_1)}, \ldots,\\ &\quad (P_n[X \mapsto struct_n\{\!|\, r \,|\!\}], \phi_n, struct_n, \delta_n)\}\end{aligned}$$

On the symbolic level, we have the lazy intruder representation. Thus, we have a single transition, where the recipe and the corresponding message are left as recipe and intruder variables, respectively.

$$\begin{aligned}&\{(\mathsf{rcv}(X).P_1, \phi_1, \mathcal{A}_1, \mathcal{X}_1, \alpha_1, \delta_1), \ldots,\\ &\ (\mathsf{rcv}(X).P_n, \phi_n, \mathcal{A}_n, \mathcal{X}_n, \alpha_n, \delta_n)\}\\ &\ \Rightarrow \{(P_1, \phi_1, \mathcal{A}_1.{+}R \mapsto X, \mathcal{X}_1, \alpha_1, \delta_1), \ldots,\\ &\quad (P_n, \phi_n, \mathcal{A}_n.{+}R \mapsto X, \mathcal{X}_n, \alpha_n, \delta_n)\}\end{aligned}$$

where R is a fresh recipe variable.

Note that there are several transitions on the symbolic level, while on the ground level there is just one transition. This is because we have removed the infinite number of transitions in the receive steps with the lazy intruder. The intruder variables are not instantiated, unless we need to consider different values in order to resolve the conditions. The semantics of the symbolic states include all ground choices of recipes, so all instantiations for the recipe variable (which determine the instantiations of the intruder variable in each structural frame).

3) Cell Read: On the ground level, the memory δ contains the sequence $\mathsf{cell}(s_1) := t_1. \cdots .\mathsf{cell}(s_k) := t_k$ for the given cell, and the initial value is given with ground context $C[\cdot]$.

$$\begin{aligned}&\{(X := \mathsf{cell}(s).P, \phi, struct, \delta)\} \uplus \mathcal{P}\\ &\ \to \{(\mathsf{if}\ s \doteq s_1\ \mathsf{then}\ P[X \mapsto t_1]\ \mathsf{else}\\ &\qquad \ldots\\ &\qquad \mathsf{if}\ s \doteq s_k\ \mathsf{then}\ P[X \mapsto t_k]\ \mathsf{else}\\ &\qquad P[X \mapsto C[s]], \phi, struct, \delta)\} \cup \mathcal{P}\end{aligned}$$

On the symbolic level, the rule is the same.

$$\begin{aligned}&\{(X := \mathsf{cell}(s).P, \phi, \mathcal{A}, \mathcal{X}, \alpha, \delta)\} \uplus \mathcal{P}\\ &\Rightarrow \{(\mathsf{if}\ s \doteq s_1\ \mathsf{then}\ P[X \mapsto t_1]\ \mathsf{else}\\ &\qquad \dots\\ &\qquad \mathsf{if}\ s \doteq s_k\ \mathsf{then}\ P[X \mapsto t_k]\ \mathsf{else}\\ &\qquad P[X \mapsto C[s]], \phi, \mathcal{A}, \mathcal{X}, \alpha, \delta)\} \cup \mathcal{P}\end{aligned}$$

4) Cell Write: On the ground level, a memory update is prepended to the sequence δ.

$$\begin{aligned}&\{(\mathsf{cell}(s) := t.P, \phi, \mathit{struct}, \delta)\} \uplus \mathcal{P}\\ &\to \{(P, \phi, \mathit{struct}, \mathsf{cell}(s) := t.\delta)\} \cup \mathcal{P}\end{aligned}$$

On the symbolic level, the rule is the same.

$$\begin{aligned}&\{(\mathsf{cell}(s) := t.P, \phi, \mathcal{A}, \mathcal{X}, \alpha, \delta)\} \uplus \mathcal{P}\\ &\Rightarrow \{(P, \phi, \mathcal{A}, \mathcal{X}, \alpha, \mathsf{cell}(s) := t.\delta)\} \cup \mathcal{P}\end{aligned}$$

5) Destructor Application: On the ground level, try-catch is syntactic sugar around if-then-else, so there is only the rule for conditional statements. On the symbolic level, we handle in a specific way the destructor applications in try-catch because of our assumptions that this is the only place in a specification where destructors are allowed.

This rule is applicable whenever a process is trying to apply a destructor, e.g., decrypting a message. For every destructor d, there must be a unique constructor c and a unique rewrite rule $d(k, c(k', X_1, \dots, X_n)) \to X_i$ (for some $i \in \{1, \dots, n\}$), and assuming the variables in k, k' and the X_j have been renamed with fresh intruder variables. To resolve the destructor application $\mathsf{try}\ X \doteq d(t_1, t_2)$, we compute the unifier $\sigma = mgu(t_1 \doteq k \wedge t_2 \doteq c(k', X_1, \dots, X_n) \wedge X \doteq X_i)$. The meaning is that t_2 must be of the form $c(k', X_1, \dots, X_n)$ and t_1 must be the corresponding decryption key term, otherwise the destructor application would yield $\mathtt{ff}$, and X is bound to the result of the destructor application. If d is actually a unary destructor, then there are no terms t_1 and k but all the rest is done in the same way.

We would like to split the possibility into two possibilities: one in which σ holds and one in which it does not. However, we cannot in general split with $\phi \wedge \sigma$ and $\phi \wedge \neg\sigma$ because σ may contain intruder variables and we need to reason about solving the constraints.

- If there is no unifier, i.e., $\sigma = \bot$, then the process simply goes to the catch branch.

$$\begin{aligned}&\{(\mathsf{try}\ X \doteq d(t_1, t_2)\ \mathsf{in}\ P_1\ \mathsf{catch}\ P_2, \phi, \mathcal{A}, \mathcal{X}, \alpha, \delta)\} \uplus \mathcal{P}\\ &\Rightarrow \{(P_2, \phi, \mathcal{A}, \mathcal{X}, \alpha, \delta)\} \cup \mathcal{P}\end{aligned}$$

- Otherwise, we use the lazy intruder to solve the constraints in FLIC $\mathcal{A}$: $LI(\mathcal{A}, \sigma) = \{\rho_1, \dots, \rho_m\}$. There is one transition for every choice of recipes returned, where the intruder applies ρ_i to the entire symbolic state. This resolves the constraints on intruder variables. Let σ_0 be the substitution of privacy variables for which the decryption succeeds. Then we split into two possibilities: one with $\phi \wedge \sigma_0$ and we continue with process P_1, and one with $\phi \wedge \neg\sigma_0$ and we continue with process P_2.

$$\begin{aligned}&\{(\mathsf{try}\ X \doteq d(t_1, t_2)\ \mathsf{in}\ P_1\ \mathsf{catch}\ P_2, \phi, \mathcal{A}, \mathcal{X}, \alpha, \delta)\} \uplus \mathcal{P}\\ &\Rightarrow \rho_i(\{(P_1, \phi \wedge \sigma_0, \mathcal{A}, \mathcal{X}, \alpha, \delta),\\ &\qquad (P_2, \phi \wedge \neg\sigma_0, \mathcal{A}, \mathcal{X}, \alpha, \delta)\} \cup \mathcal{P})\end{aligned}$$

Moreover, the intruder can always take a choice of recipes which is definitely not a solution to the constraints, so we also have an additional transition where σ is excluded.

$$\begin{aligned}&\{(\mathsf{try}\ X \doteq d(t_1, t_2)\ \mathsf{in}\ P_1\ \mathsf{catch}\ P_2, \phi, \mathcal{A}, \mathcal{X}, \alpha, \delta)\} \uplus \mathcal{P}\\ &\Rightarrow \{(P_2, \phi, \mathcal{A}, \mathcal{X} \wedge \forall \bar{Y}.\ \neg\sigma, \alpha, \delta)\} \cup \mathcal{P}\end{aligned}$$

where $\bar{Y} = ivars(\sigma) \setminus ivars(\mathcal{A})$, i.e., the intruder variables that are not occurring in the FLIC are universally quantified when excluding the unifier. The function $ivars$ gives the intruder variables of a FLIC, i.e., $ivars(\mathcal{A}) = vars(\mathcal{A}) \cap \mathcal{V}_{intruder}$; we extend this function to substitutions.

Example A.1. Executing the transaction Example II.1 leads to a possibility with the process $\mathsf{try}\ N \doteq \mathsf{dcrypt}(\mathsf{inv}(\mathsf{pk}(\mathsf{s})), M)\ \mathsf{in}\ P_0\ \mathsf{catch}\ 0$ and the FLIC $\mathcal{A} = +R \mapsto M$. Then we use the rewrite rule $\mathsf{dcrypt}(\mathsf{inv}(X_1), \mathsf{crypt}(X_1, X_2, X_3)) \to X_2$ and we get $\sigma = [M \mapsto \mathsf{crypt}(\mathsf{pk}(\mathsf{s}), N, X_3), X_1 \mapsto \mathsf{pk}(\mathsf{s}), X_2 \mapsto N]$. We have for instance $\rho = [R \mapsto \mathsf{crypt}(\mathsf{pk}(\mathsf{s}), R_2, R_3)] \in LI(\mathcal{A}, \sigma)$, so the intruder considers one symbolic state where they have chosen ρ and the unifier does not depend on intruder variables anymore. In this case, the split leads to one possibility with process P_0 and FLIC $\mathcal{A}' = +R_2 \mapsto X_2.+R_3 \mapsto X_3$, i.e., the intruder knows that the try succeeds (the other possibility is deleted immediately because it would have the condition false). Moreover, there is a symbolic state where we remember the disequality $\forall X_1, X_2, X_3, N.\ M \not\doteq \mathsf{crypt}(\mathsf{pk}(\mathsf{s}), N, X_3) \vee X_1 \not\doteq \mathsf{pk}(\mathsf{s}) \vee X_2 \not\doteq N$, which can be simplified to $\forall X_3, N.\ M \not\doteq \mathsf{crypt}(\mathsf{pk}(\mathsf{s}), N, X_3)$. $\lhd$

6) Conditional Statement: On the ground level, we split a possibility into two, one for the case that the condition is true and we go into the then branch, and one for the else branch. By construction, if the marked possibility is split then there is only one branch that is consistent with the current truth γ and it is marked accordingly.

$$\begin{aligned}&\{(\mathsf{if}\ \psi\ \mathsf{then}\ P_1\ \mathsf{else}\ P_2), \phi, \mathit{struct}\} \uplus \mathcal{P}\\ &\to \{(P_1, \phi \wedge \psi, \mathit{struct}), (P_2, \phi \wedge \neg\psi, \mathit{struct})\} \cup \mathcal{P}\end{aligned}$$

On the symbolic level, there are two base cases: when the condition is a relation $R(t_1, \dots, t_n)$ and when the condition is an equality $s \doteq t$. For an arbitrary formula, we can eliminate the negation by swapping the branches and eliminate the conjunction by nesting conditional statements.

- If the condition is a relation: The possibility is split into two possibilities, just like on the ground level.

$$\begin{aligned}&\{(\mathsf{if}\ R(t_1,\dots,t_n)\ \mathsf{then}\ P_1\ \mathsf{else}\ P_2,\phi,\mathcal{A},\mathcal{X},\alpha,\delta)\}\uplus\mathcal{P}\\&\Rightarrow\{(P_1,\phi\wedge R(t_1,\dots,t_n),\mathcal{A},\mathcal{X},\alpha,\delta),\\&\quad(P_2,\phi\wedge\neg R(t_1,\dots,t_n),\mathcal{A},\mathcal{X},\alpha,\delta)\}\cup\mathcal{P}\end{aligned}$$

  Recall that all the t_i must be terms using only symbols from Σ_0 and $fv(\alpha_0)$ a that point, otherwise we consider it a specification error.
- If the condition is an equality: We first compute the unifier $\sigma = mgu(s \doteq t)$ and then the transitions are just like for destructor application.

Example A.2. Suppose that there is the possibility ($\mathsf{if}\ X \doteq Y\ \mathsf{then}\ P_1\ \mathsf{else}\ P_2,\phi,\mathcal{A},\mathcal{X},\alpha,\delta$) in the current symbolic state, where $\mathcal{A} = +R_1 \mapsto X.+R_2 \mapsto Y$. Then $\sigma = [X \mapsto Y]$ and the only intruder result is $\rho = [R_2 \mapsto R_1]$. The intruder considers one symbolic state where they have chosen ρ (all the FLICs and the rest of the symbolic state is updated accordingly), and one symbolic state where σ is excluded for the FLIC $\mathcal{A}$, so we remember that $X \not\doteq Y$ in this possibility. ◁

We "unfold" the condition until we reach atomic formulas, by nesting conditional statements or swapping the branches. This does not change the semantics. When the condition does not depend on intruder variables, we then split into two possibilities. However, when the condition introduces constraints to solve on intruder variables, we have a transition for every solution to the constraints returned by the lazy intruder. After applying a choice of recipes that solves the constraints, the condition does not depend on intruder variables anymore so we can then split in two possibilities, as on the ground level. Additionally, we also have one transition corresponding to any choice of recipes which is not a solution. By correctness of the lazy intruder, we thus represent all ground choices of recipes. Therefore, we are simply partitioning the ground choices of recipes.

7) Send: On the ground level, if the intruder observes that a message is sent, then they can rule out all possibilities where the remaining process is 0. Note that this rule can only be applied if all possibilities start either with $\mathsf{snd}(\cdot)$ or 0; otherwise another evaluation rule must be applied. For all others, each $struct_i$ is augmented by the message sent in the respective possibility:

$$\begin{aligned}&\{\underline{(\mathsf{snd}(t_1).P_1,\phi_1,struct_1,\delta_1)},\dots,\\&(\mathsf{snd}(t_k).P_k,\phi_k,struct_k,\delta_k)\}\uplus\mathcal{P}\\&\to\{\underline{(P_1,\phi_1,struct_1.l\mapsto t_1,\delta_1)},\dots,\\&\quad(P_k,\phi_k,struct_k.l\mapsto t_k,\delta_k)\}\end{aligned}$$

where $\beta_0 \leftarrow \beta_0 \wedge \bigvee_{i=1}^{k}\phi_i$, l is a fresh label and all the processes in $\mathcal{P}$ must be the 0 process.

On the symbolic level, the rule is the same.

$$\begin{aligned}&\{(\mathsf{snd}(t_1).P_1,\phi_1,\mathcal{A}_1,\mathcal{X}_1,\alpha_1,\delta_1),\dots,\\&(\mathsf{snd}(t_k).P_k,\phi_k,\mathcal{A}_k,\mathcal{X}_k,\alpha_k,\delta_k)\}\uplus\mathcal{P}\\&\Rightarrow\{(P_1,\phi_1,\mathcal{A}_1.-l\mapsto t_1,\mathcal{X}_1,\alpha_1,\delta_1),\dots,\\&\quad(P_k,\phi_k,\mathcal{A}_k.-l\mapsto t_k,\mathcal{X}_k,\alpha_k,\delta_k)\}\end{aligned}$$

where $\beta_0 \leftarrow \beta_0 \wedge \bigvee_{i=1}^{k}\phi_i$, l is a fresh label and all the processes in $\mathcal{P}$ must be the 0 process.

8) Release: On the ground level, the formula released by the marked possibility is added to the payload or the truth (depending on the mode), and formulas released by other possibilities are ignored.

$$\{\underline{(\mathsf{mode}\ \psi.P,\phi,struct)}\}\uplus\mathcal{P}\to\{\underline{(P,\phi,struct)}\}\cup\mathcal{P}$$

and $\alpha \leftarrow \alpha\wedge\psi$ if $\mathsf{mode} = \star$ or $\gamma\leftarrow\gamma\wedge\psi$ if $\mathsf{mode} = \diamond$.

On the symbolic level, we have that each possibility could be the marked one. Therefore, we do not update the common payload but rather the partial payload attached to the given possibility.

$$\{(\star\ \psi.P,\phi,\mathcal{A},\mathcal{X},\alpha,\delta)\uplus\mathcal{P}\Rightarrow\{(P,\phi,\mathcal{A},\mathcal{X},\alpha\wedge\psi,\delta)\}\cup\mathcal{P}$$

The formula released should be consistent with all models of $\alpha_0 \wedge \beta_0 \wedge \gamma_0 \wedge \phi$, i.e., the truths that this possibility symbolically represents. If that is not the case, it counts as a privacy violation: it would mean that after the transaction for some ground state we have a pair (α,β) where the payload is inconsistent and thus (α,β)-privacy trivially holds. For the procedure we can either assume that all releases are consistent or, as an option, we can detect inconsistent releases and give a warning to the user, because this indicates that something is wrong in the model.

In the semantics of the symbolic states, we consider all payloads $\alpha_0 \wedge [\alpha_i]^\gamma$ that the intruder can observe so our rules cover the releases with $\mathsf{mode} = \star$. We do *not* support releases with $\mathsf{mode} = \diamond$ in γ in this paper. We have not seen examples of protocols requiring this construct so it is left out at the moment. However, we could include them if needed, for instance we could add a component for "partial truth" γ_i similarly to the partial payloads, that would be used in the semantics when defining the models.

9) Terminate: On the ground level, the intruder observes that the execution has terminated because no messages are sent, so they can rule out all possibilities that are not terminated.

$$\begin{aligned}&\{\underline{(0,\phi_1,struct_1,\delta_1)},\dots,(0,\phi_k,struct_k,\delta_k)\}\uplus\mathcal{P}\\&\to\{\underline{(0,\phi_1,struct_1,\delta_1)},\dots,(0,\phi_k,struct_k,\delta_k)\}\end{aligned}$$

where every process in $\mathcal{P}$ starts with a send step and $\beta_0 \leftarrow \beta_0 \wedge \bigvee_{i=1}^{k}\phi_i$.

On the symbolic level, the rule is the same.

$$\begin{aligned}&\{(0,\phi_1,\mathcal{A}_1,\mathcal{X}_1,\alpha_1,\delta_1),\dots,(0,\phi_k,\mathcal{A}_k,\mathcal{X}_k,\alpha_k,\delta_k)\}\uplus\mathcal{P}\\&\Rightarrow\{(0,\phi_1,\mathcal{A}_1,\mathcal{X}_1,\alpha_1,\delta_1),\dots,(0,\phi_k,\mathcal{A}_k,\mathcal{X}_k,\alpha_k,\delta_k)\}\end{aligned}$$

where every process in $\mathcal{P}$ starts with a send step and the rest of the symbolic state is updated as follows: $\beta_0 \leftarrow \beta_0 \wedge \bigvee_{i=1}^{k} \phi_i$.

Note that on the ground level, eventually the marked possibility either sends or terminates and the corresponding rule is applied. Since other steps are done in different evaluation rules that must be applied before, the processes that do not send are actually terminating (nil process). Thus, on the symbolic level both the send and terminate rules are in general applicable at the same time.

10) Correctness: Our evaluation rules correspond to internal transitions for the symbolic execution of transactions, which is distinct from the overall transition system where the transactions are atomic. We define $\underset{P}{\longrightarrow}$ to be the relation between an initial ground state of a transaction P and a finished state, using the relation $\rightarrow$ of evaluation rules until all processes in $\mathcal{P}$ have terminated. Then we can define $S_{\underset{P}{\longrightarrow}}$ the set of ground states that are reached by executing the transaction P, i.e., we can talk about one transition corresponding to the execution of one atomic transaction. Similarly, we define $\mathcal{S}_{\underset{P}{\Longrightarrow}}$ to be the set of symbolic states after the execution of one atomic transaction:

$$S_{\underset{P}{\longrightarrow}} = \{S' \mid \mathit{init}(P, S) \underset{P}{\longrightarrow} S'\}$$
$$\mathcal{S}_{\underset{P}{\Longrightarrow}} = \{\mathcal{S}' \mid \mathit{init}(P, \mathcal{S}) \underset{P}{\Longrightarrow} \mathcal{S}'\}$$

The symbolic version of the transitions is correct w.r.t. the transitions that can happen on the ground level.

Proposition A.1 (Reachability correctness)**.** *Let $\mathcal{S}$ be a finished symbolic state and P be a transaction process. Let $[\![\mathcal{S}_{\underset{P}{\Longrightarrow}}]\!]$ be the ground states after transitions between symbolic states, and $[\![\mathcal{S}]\!]_{\underset{P}{\longrightarrow}}$ be the ground states after transitions between ground states:*

$$[\![\mathcal{S}_{\underset{P}{\Longrightarrow}}]\!] = \{S \mid \mathcal{S}' \in \mathcal{S}_{\underset{P}{\Longrightarrow}} \text{ and } S \in [\![\mathcal{S}']\!]\}$$
$$[\![\mathcal{S}]\!]_{\underset{P}{\longrightarrow}} = \{S' \mid S \in [\![\mathcal{S}]\!] \text{ and } S' \in S_{\underset{P}{\longrightarrow}}\}$$

Then we have $[\![\mathcal{S}_{\underset{P}{\Longrightarrow}}]\!] = [\![\mathcal{S}]\!]_{\underset{P}{\longrightarrow}}$.

B. Proofs

In this appendix, we give the proofs of the theorems and lemmas that we stated in the body of the paper. To this end, we also prove a number of auxiliary results.

Lemma A.2. *Let $\mathcal{A}$ be a FLIC, ρ be a choice of recipes such that $dom(\rho) \cap rvars(\mathcal{A}) = \emptyset$ and let σ be a substitution such that $dom(\sigma) \cap vars(\mathcal{A}) = \emptyset$. Let $(\rho', \mathcal{A}', \sigma')$ such that $(\rho, \mathcal{A}, \sigma) \rightsquigarrow (\rho', \mathcal{A}', \sigma')$. Then for every recipe r, we have $\sigma'(\mathcal{A}\{\!\!\{ r \}\!\!\}) = \sigma'(\mathcal{A}'\{\!\!\{ \rho'(r) \}\!\!\})$.*

Proof. For a recipe variable that is changed by the rule application:

- **Unification**: $\mathcal{A} = \mathcal{A}_1.{-l} \mapsto s.\mathcal{A}_2.{+R} \mapsto t.\mathcal{A}_3$, $\mathcal{A}' = \sigma'(\mathcal{A}_1.{-l} \mapsto s.\mathcal{A}_2.\mathcal{A}_3)$, $\rho'(R) = l$ and $\sigma' \models s \doteq t$ so $\sigma'(\mathcal{A}'\{\!\!\{ \rho'(R) \}\!\!\}) = \sigma'(s) = \sigma'(t) = \sigma'(\mathcal{A}\{\!\!\{ R \}\!\!\})$.
- **Composition**: $\mathcal{A} = \mathcal{A}_1.{+R} \mapsto f(t_1, \ldots, t_n).\mathcal{A}_2$, $\mathcal{A}' = \mathcal{A}_1.{+R_1} \mapsto t_1.\cdots.{+R_n} \mapsto t_n.\mathcal{A}_2$ and $\rho'(R) = f(R_1, \ldots, R_n)$ so $\mathcal{A}'\{\!\!\{ \rho'(R) \}\!\!\} = f(t_1, \ldots, t_m) = t = \mathcal{A}\{\!\!\{ R \}\!\!\}$.
- **Guessing**: $\mathcal{A} = \mathcal{A}_1.{+R} \mapsto x.\mathcal{A}_2$, $\mathcal{A}' = \sigma'(\mathcal{A}_1.\mathcal{A}_2)$, $\rho'(R) = c$ and $\sigma' \models x \doteq c$ so $\sigma'(\mathcal{A}'\{\!\!\{ \rho'(R) \}\!\!\}) = \sigma'(c) = \sigma'(x) = \sigma'(\mathcal{A}\{\!\!\{ R \}\!\!\})$.
- **Repetition**: $\mathcal{A} = \mathcal{A}_1.{+R_1} \mapsto X.\mathcal{A}_2.{+R_2} \mapsto X.\mathcal{A}_3$, $\mathcal{A}' = \mathcal{A}_1.{+R_1} \mapsto X.\mathcal{A}_2.\mathcal{A}_3$ and $\rho'(R_2) = R_1$ so $\mathcal{A}'\{\!\!\{ \rho'(R_2) \}\!\!\} = X = \mathcal{A}\{\!\!\{ R_2 \}\!\!\}$.

For a recipe variable R that is not changed by the rule application, we also have $\sigma'(\mathcal{A}\{\!\!\{ R \}\!\!\}) = \sigma'(\mathcal{A}'\{\!\!\{ \rho'(R) \}\!\!\})$ and similarly for labels. For a composed recipe, this holds by induction on the structure of the recipe. ☐

The next four lemmas prove the soundness, completeness, correctness and termination of the lazy intruder that we consider in this paper.

Lemma A.3 (Lazy intruder soundness)**.** *Let $\mathcal{A}$ be a FLIC, ρ be a choice of recipes such that $dom(\rho) \cap rvars(\mathcal{A}) = \emptyset$, let σ be a substitution such that $dom(\sigma) \cap vars(\mathcal{A}) = \emptyset$, let $\mathcal{I} \models \mathcal{A}$ such that $\mathcal{I} \models \sigma$ and let ρ_0 be a ground choice of recipes such that $\rho_0 \models \rho$. Let $(\rho', \mathcal{A}', \sigma')$ such that $(\rho, \mathcal{A}, \sigma) \rightsquigarrow (\rho', \mathcal{A}', \sigma')$, ρ' represents ρ_0 with ρ'_0 w.r.t. $\mathcal{A}$ and $\mathcal{I}$, ρ'_0 constructs $\mathcal{I}(\mathcal{A}')$ and $\mathcal{I} \models \sigma'$. Then ρ_0 constructs $\mathcal{I}(\mathcal{A})$.*

Proof. We start by showing that ρ'_0 constructs $\mathcal{I}(\mathcal{A})$. Let R be a recipe variable such that $\mathcal{I}(\mathcal{A}) = \mathcal{A}_1.{+R} \mapsto t.\mathcal{A}_2$. First, we consider the case that $R \notin dom(\rho')$. Then $\mathcal{I}(\mathcal{A}') = \mathcal{A}'_1.{+R} \mapsto t.\mathcal{A}'_2$ and $\mathcal{A}'_1\{\!\!\{ \rho'_0(R) \}\!\!\} = t$, so $\mathcal{A}_1\{\!\!\{ \rho'_0(R) \}\!\!\} = t$.

Next, we consider the case that $R \in dom(\rho')$. We proceed by distinguishing which lazy intruder rule has been applied.

- **Unification**: Then $\mathcal{I}(\mathcal{A}) = \mathcal{A}_1.{-l} \mapsto t.\mathcal{A}_2.{+R} \mapsto t.\mathcal{A}_3$ and $\rho'_0(R) = l$ so $\mathcal{A}_1\{\!\!\{ \rho'_0(R) \}\!\!\} = t$.
- **Composition**: Then

$$\mathcal{I}(\mathcal{A}) = \mathcal{A}_1.{+R} \mapsto f(t_1, \ldots, t_n).\mathcal{A}_2$$
$$\mathcal{I}(\mathcal{A}') = \mathcal{A}'_1.{+R_1} \mapsto t_1.\cdots.{+R_n} \mapsto t_n.\mathcal{A}'_3$$
$$\rho'_0(R) = f(\rho'_0(R_1), \ldots, \rho'_0(R_n))$$

  Therefore

$$\begin{aligned}\mathcal{A}_1\{\!\!\{ \rho'_0(R) \}\!\!\} &= f(\mathcal{I}(\mathcal{A}')\{\!\!\{ \rho'_0(R_1) \}\!\!\}, \ldots, \mathcal{I}(\mathcal{A}')\{\!\!\{ \rho'_0(R_n) \}\!\!\}) \\ &= f(t_1, \ldots, t_n) = t\end{aligned}$$

- **Guessing**: Then $\mathcal{I}(\mathcal{A}) = \mathcal{A}_1.{+R} \mapsto c.\mathcal{A}_2$ and $\rho'_0(R) = c$ so $\mathcal{A}_1\{\!\!\{ \rho'_0(R) \}\!\!\} = c$.
- **Repetition**: Then $\mathcal{I}(\mathcal{A}) = \mathcal{A}_1.{+R'} \mapsto t.\mathcal{A}_2.{+R} \mapsto t.\mathcal{A}_3$, ${+R'} \mapsto t \in \mathcal{I}(\mathcal{A}')$ and $\rho'_0(R) = \rho'_0(R')$. Let $\mathcal{A}_0 = \mathcal{A}_1.{+R'} \mapsto t.\mathcal{A}_2$. Then $\mathcal{A}_0\{\!\!\{ \rho'_0(R) \}\!\!\} = \mathcal{I}(\mathcal{A}')\{\!\!\{ \rho'_0(R') \}\!\!\} = t$.

We have shown that ρ'_0 constructs $\mathcal{I}(\mathcal{A})$. Since ρ' represents ρ_0 with ρ'_0, for every $R \in rvars(\mathcal{A})$, we have $\mathcal{I}(\mathcal{A})\{\!\!\{ \rho'_0(R) \}\!\!\} = \mathcal{I}(\mathcal{A})\{\!\!\{ \rho_0(R) \}\!\!\}$. Thus ρ_0 constructs $\mathcal{I}(\mathcal{A})$. ☐

Lemma A.4 (Lazy intruder completeness)**.** *Let $\mathcal{A}$ be a FLIC, ρ be a choice of recipes such that $dom(\rho) \cap rvars(\mathcal{A}) = \emptyset$, let σ be a substitution such that $dom(\sigma) \cap vars(\mathcal{A}) = \emptyset$, let $\mathcal{I} \models \mathcal{A}$ such that $\mathcal{I} \models \sigma$ and let ρ_0 be a ground choice of recipes such* *that $\rho_0 \models \rho$ and ρ_0 constructs $\mathcal{I}(\mathcal{A})$. Then either $\mathcal{A}$ is simple or there exists $(\rho', \mathcal{A}', \sigma')$ such that $(\rho, \mathcal{A}, \sigma) \rightsquigarrow (\rho', \mathcal{A}', \sigma')$, ρ' represents ρ_0 with ρ'_0 w.r.t. $\mathcal{A}$ and $\mathcal{I}$, ρ'_0 constructs $\mathcal{I}(\mathcal{A}')$ and $\mathcal{I} \models \sigma'$.*

Proof. Assume that $\mathcal{A}$ is not simple. Let $+R \mapsto t \in \mathcal{A}$ denote the first non-simple constraint. First, we consider the case that $\rho_0(R) \in dom(\mathcal{A})$ and there exists a label l, either the same label or occurring before $\rho_0(R)$, such that $\mathcal{A} = \mathcal{A}_1.-l \mapsto s.\mathcal{A}_2.+R \mapsto t.\mathcal{A}_3$, $s, t \notin \mathcal{V}$ and $mgu(\sigma \wedge s \doteq t) \neq \bot$. Then **Unification** is applicable, producing $(\rho', \mathcal{A}', \sigma')$ where $\rho' = [R \mapsto l]\rho$, $\mathcal{A}' = \sigma'(\mathcal{A}_1.\mathcal{A}_2)$ and $\sigma' = mgu(\sigma \wedge s \doteq t)$. Let $\rho'_0(R) = l$ and $\rho'_0(R') = \rho_0(R')$ for other recipe variables. Then ρ' represents ρ_0 with ρ'_0 w.r.t. $\mathcal{A}$ and $\mathcal{I}$, because the only recipe variable in $rvars(\mathcal{A}) \cap dom(\rho')$ is R, and either $\rho'_0(R) = \rho_0(R) = l$ or $\rho'_0(R) = l <_{\mathcal{A}} \rho_0(R)$. Moreover, since ρ_0 constructs $\mathcal{I}(\mathcal{A})$, we have ρ'_0 constructs $\mathcal{I}(\mathcal{A}')$ and also $\mathcal{I}(s) = \mathcal{I}(t)$, thus $\mathcal{I} \models \sigma'$.

Next, we consider the case that $t = f(t_1, \ldots, t_n)$ and either $\rho_0(R) = f(r_1, \ldots, r_n)$ or $\rho_0(R) \in dom(\mathcal{A})$ but there is no label l, either the same or occurring before $\rho_0(R)$, such that $\mathcal{A}\{\!|\, l \,|\!\} \notin \mathcal{V}$ and $mgu(\sigma \wedge \mathcal{A}\{\!|\, l \,|\!\} \doteq t) \neq \bot$. Then $\mathcal{A} = \mathcal{A}_1.+R \mapsto f(t_1, \ldots, t_n).\mathcal{A}_2$. Therefore, **Composition** is applicable, producing $(\rho', \mathcal{A}', \sigma')$ where $\rho' = [R \mapsto f(R_1, \ldots, R_n)]\rho$, $\mathcal{A}' = \mathcal{A}_1.+R_1 \mapsto t_1. \cdots .+R_n \mapsto t_n.\mathcal{A}_2$ and $\sigma' = \sigma$, where the R_i are fresh recipe variables. If $\rho_0(R) = f(r_1, \ldots, r_n)$, let $\rho'_0(R_i) = r_i$ for $i \in \{1, \ldots, n\}$ and $\rho'_0(R') = \rho_0(R')$ for other recipe variables. Otherwise there exists $r <_{\mathcal{A}} \rho_0(R)$ such that $\mathcal{I}(\mathcal{A})\{\!|\, \rho_0(r) \,|\!\} = t$ and $\rho_0(r) = f(r^1, \ldots, r^n)$; we let $\rho'_0(R_i) = r^i$ for $i \in \{1, \ldots, n\}$ and $\rho'_0(R') = \rho_0(R')$ for other recipe variables. Then ρ' represents ρ_0 with ρ'_0 w.r.t. $\mathcal{A}$ and $\mathcal{I}$, because the only recipe variable in $rvars(\mathcal{A}) \cap dom(\rho')$ is R, and either $\rho_0(R) = \rho'_0(R) = f(r_1, \ldots, r_n)$ or $\rho'_0(R) = f(r^1, \ldots, r^n) <_{\mathcal{A}} \rho_0(R)$. Moreover, since ρ_0 constructs $\mathcal{I}(\mathcal{A})$, we have ρ'_0 constructs $\mathcal{I}(\mathcal{A}')$.

Next, we consider the case that $t \in \mathcal{V}_{privacy}$. Then $\mathcal{A} = \mathcal{A}_1.+R \mapsto t.\mathcal{A}_2$ and $\mathcal{I}(t) = c$ for some $c \in dom(t)$. Therefore, **Guessing** is applicable, producing $(\rho', \mathcal{A}', \sigma')$ where $\rho' = [R \mapsto c]\rho$, $\mathcal{A}' = \sigma'(\mathcal{A}_1.\mathcal{A}_2)$ and $\sigma' = mgu(\sigma \wedge t \doteq c)$. If $\rho_0(R) = c$, let $\rho'_0 = \rho_0$. Otherwise let $\rho'_0(R) = c$ and $\rho'_0(R') = \rho_0(R')$ for other recipe variables. Then ρ' represents ρ_0 with ρ'_0 w.r.t. $\mathcal{A}$ and $\mathcal{I}$, because the only recipe variable in $rvars(\mathcal{A}) \cap dom(\rho')$ is R and either $\rho_0(R) = \rho'_0(R) = c$ or $\rho'_0(R) = c <_{\mathcal{A}} \rho_0(R)$. Moreover, since ρ_0 constructs $\mathcal{I}(\mathcal{A})$, we have ρ'_0 constructs $\mathcal{I}(\mathcal{A}')$ and also $\mathcal{I}(t) = c$, thus $\mathcal{I} \models \sigma'$.

Finally we consider the case that $t \in \mathcal{V}_{intruder}$. Then $\mathcal{A} = \mathcal{A}_1.+R' \mapsto t.\mathcal{A}_2.+R \mapsto t.\mathcal{A}_3$. Therefore, **Repetition** is applicable, producing $(\rho', \mathcal{A}', \sigma')$ where $\rho' = [R \mapsto R']\rho$, $\mathcal{A}' = \mathcal{A}_1.+R' \mapsto t.\mathcal{A}_2.\mathcal{A}_3$ and $\sigma' = \sigma$. Let $\rho'_0(R) = \rho_0(R')$ and $\rho'_0(R'') = \rho_0(R'')$ for other recipe variables. Then ρ' represents ρ_0 with ρ'_0 w.r.t. $\mathcal{A}$ and $\mathcal{I}$, because the only recipe variable in $rvars(\mathcal{A}) \cap dom(\rho')$ is R and $\mathcal{I}(\mathcal{A})\{\!|\, \rho_0(R) \,|\!\} = \mathcal{I}(\mathcal{A})\{\!|\, \rho'_0(R) \,|\!\}$. Moreover, since ρ_0 constructs $\mathcal{I}(\mathcal{A})$, we have ρ'_0 constructs $\mathcal{I}(\mathcal{A}')$. ☐

Lemma A.5. *Let $\mathcal{A}$ be a FLIC, ρ be a choice of recipes such that $dom(\rho) \cap rvars(\mathcal{A}) = \emptyset$, let σ be a substitution such that $dom(\sigma) \cap vars(\mathcal{A}) = \emptyset$, let $\mathcal{I} \models \mathcal{A}$ such that $\mathcal{I} \models \sigma$ and let ρ_0 be a ground choice of recipes such that $\rho_0 \models \rho$. Then ρ_0 constructs $\mathcal{I}(\mathcal{A})$ iff there exists $(\rho', \mathcal{A}', \sigma')$ such that $(\rho, \mathcal{A}, \sigma) \rightsquigarrow^* (\rho', \mathcal{A}', \sigma')$, ρ' represents ρ_0 with ρ'_0 w.r.t. $\mathcal{A}$ and $\mathcal{I}$, ρ'_0 constructs $\mathcal{I}(\mathcal{A}')$ and $\mathcal{I} \models \sigma'$.*

Proof. By induction, using Lemmas A.3 and A.4. ☐

Lemma A.6 (Lazy intruder termination)**.** *Let $\mathcal{A}$ be a FLIC, ρ be a choice of recipes such that $dom(\rho) \cap rvars(\mathcal{A}) = \emptyset$ and let σ be a substitution such that $dom(\sigma) \cap vars(\mathcal{A}) = \emptyset$. Then there is a finite number of $(\rho', \mathcal{A}', \sigma')$ such that $(\rho, \mathcal{A}, \sigma) \rightsquigarrow^* (\rho', \mathcal{A}', \sigma')$.*

Proof. We define the weight of a FLIC $\mathcal{A}$ to be the pair (v, s), where

- v is the number of intruder variables in the FLIC: $v = \#ivars(\mathcal{A})$; and
- s is the sum of the size of the messages sent: $s = \sum_{+R \mapsto t \in \mathcal{A}} size(t)$, where the size of a message is defined as 1 for a variable and $size(f(t_1, \ldots, t_n)) = 1 + \sum_{i=1}^{n} size(t_i)$ for a composed message.

The weights with the lexicographic order form a well-founded ordering. Every rule decreases the weight.

- **Unification**: The mgu may instantiate intruder variables so v would decrease, and if not then v stays the same but one message sent is removed so s decreases.
- **Composition**: v stays the same, but the message is decomposed by removing the outermost function application so s decreases (by 1).
- **Guessing** and **Repetition**: v stays the same, but one message sent is removed so s decreases (by 1).

There cannot be an infinite sequence of decreasing weights so the lazy intruder terminates. ☐

Theorem III.1 (Lazy intruder correctness)**.** *Let $\mathcal{A}$ be a FLIC, σ be a substitution, $\mathcal{I} \models \mathcal{A}$ such that $\mathcal{I} \models \sigma$ and let ρ_0 be a ground choice of recipes. Then ρ_0 constructs $\mathcal{I}(\mathcal{A})$ iff there exists $\rho \in LI(\mathcal{A}, \sigma)$ such that ρ represents ρ_0 w.r.t. $\mathcal{A}$ and $\mathcal{I}$. Moreover, $LI(\mathcal{A}, \sigma)$ is finite.*

Proof. This follows directly from Lemmas A.5 and A.6. ☐

We now prove our results for normal symbolic states.

Lemma IV.1. *Let $\mathcal{S} = (\alpha_0, \beta_0, \_, \_)$ be a normal symbolic state, where the possibilities have conditions $\phi_1, \ldots, \phi_n$ and FLICs $\mathcal{A}_1, \ldots, \mathcal{A}_n$. Let $S \in [\![\mathcal{S}]\!]$, ρ_0 be the ground choice of recipes defining S and $concr$ be the concrete frame in S. Let $\theta \models \alpha_0 \wedge \beta_0 \wedge \phi_i$ for some $i \in \{1, \ldots, n\}$ and $concr' = \theta(\rho_0(\mathcal{A}_i))$. Then $concr \sim concr'$.*

Proof. Assume that the frames are not statically equivalent. This means there exists a witness, i.e., a pair of ground recipes (r_1, r_2) such that

$$concr\{\!|\, r_1 \,|\!\} = concr\{\!|\, r_2 \,|\!\}$$
$$concr'\{\!|\, r_1 \,|\!\} \neq concr'\{\!|\, r_2 \,|\!\}$$

We show that for each witness (r_1, r_2), either it contradicts that $\mathcal{S}$ is normal or there is a smaller witness according to the following well-founded ordering:

$$\begin{aligned}(r_1, r_2) < (r_1', r_2') \quad \text{iff} \quad & w(r_1) < w(r_1') \text{ and } w(r_2) \leq w(r_2') \\ & \text{or } w(r_1) \leq w(r_1') \text{ and } w(r_2) < w(r_2') \\ & \text{or } w(r_1) < w(r_2') \text{ and } w(r_2) \leq w(r_1') \\ & \text{or } w(r_1) \leq w(r_2') \text{ and } w(r_2) < w(r_1')\end{aligned}$$

where the weight $w(r)$ of recipe r is defined as the lexicographically ordered pair (s, h) where s is the size of $concr\{\!|\, r \,|\!\}$ and h is the number of the highest label in r, i.e., that occurs on the hth position in $concr$; and $h = 0$ if there are no labels in r.

We first handle the case that both r_1 and r_2 are composed. Then $r_1 = f(r_1^1, \ldots, r_1^n)$ and $r_2 = f(r_2^1, \ldots, r_2^n)$ for the same f (otherwise they cannot produce the same value in $concr$). Then at least one of the pairs (r_1^i, r_2^i) is already a witness that is smaller in the ordering.

Thus, in all remaining cases we have a pair (l, r) where l is a label and r is a ground recipe. Without loss of generality, we can assume that if r is also a label then l occurs after r in the frames. By definition of $[\![\mathcal{S}]\!]$, there exist $j \in \{1, \ldots, n\}$, one FLIC $\mathcal{A}_j$ and one model $\gamma \models \alpha_0 \wedge \beta_0 \wedge \gamma_0 \wedge \phi_j$ such that $concr = \gamma(\rho_0(\mathcal{A}_j))$. Let R be a fresh recipe variable and $\mathcal{A} = \mathcal{A}_j.{+}R \mapsto \mathcal{A}_j\{\!|\, l \,|\!\}$. Let $\mathcal{I}$ be the interpretation such that $\mathcal{I}$ and γ agree on the privacy variables and for every R' such that $\mathcal{A}_j = \mathcal{A}_0.{+}R' \mapsto X.\mathcal{A}_0'$, $\mathcal{I}(X) = concr\{\!|\, \rho_0(R') \,|\!\}$. Let us extend ρ_0 with $\rho_0(R) = r$, where r is the ground recipe such that (l, r) is a witness. Then we have that ρ_0 constructs $\mathcal{I}(\mathcal{A})$. By Theorem III.1, there exists $\rho \in LI(\mathcal{A}, \varepsilon)$ such that ρ represents ρ_0 w.r.t. $\mathcal{A}$ and $\mathcal{I}$. Let ρ_0' be the respective instance of ρ. Since $\mathcal{S}$ is normal, we know that $l \simeq \rho(R)$, i.e., we have checked that for every ground choice of recipes ρ', $(l, \rho'(\rho(R)))$ is not a witness.

Let us consider the case that $\rho(R) = R' \in rvars(\mathcal{A}_j)$, which can only happen if the repetition rule has been used, which in turn can only happen if $\mathcal{A}_j = \mathcal{A}_0.{+}R' \mapsto X.\mathcal{A}_0'.{-}l \mapsto X.\mathcal{A}_0''$, so l maps to a message that the intruder has sent earlier and that they received back from some agent. As mentioned above, since $\mathcal{S}$ is normal, the pair $(l, \rho_0(R'))$ is not a witness (the intruder can check that in all FLICs they received back at l whatever they sent at R'). Thus, the pair $(\rho_0(R'), r)$ must be a witness, and this is smaller than (l, r), because the size of the produced message is the same, but $\rho_0(R')$ can only use labels from $\mathcal{A}_0$ and has thus a lower weight than l.

Next we consider the case that $\rho(R) \in dom(\mathcal{S})$. Since ρ represents ρ_0 with ρ_0', we have $\rho_0(R) \in dom(\mathcal{S})$ and either $\rho_0'(R) = \rho_0(R)$ or $\rho_0'(R) <_{\mathcal{A}} \rho_0(R)$. The subcase $\rho_0'(R) = \rho_0(R) = l'$ is however impossible, because as mentioned above, $\mathcal{S}$ is normal so (l, l') is checked and cannot be a witness. For the subcase $\rho_0'(R) = l' <_{\mathcal{A}} \rho_0(R) = l''$, we have that (l, l'') is a witness and $l \simeq l'$, so (l', l'') must be a witness, and this is smaller because $l' <_{\mathcal{A}} l'' <_{\mathcal{A}} l$.

Finally we consider the case that $\rho(R)$ is a composed recipe. Since ρ represents ρ_0 with ρ_0', we have either $\rho_0'(R) = f(r_1', \ldots, r_n')$ and $\rho_0(R) = f(r_1, \ldots, r_n)$ such that $\mathcal{I}(\mathcal{A})\{\!|\, r_i' \,|\!\} = \mathcal{I}(\mathcal{A})\{\!|\, r_i \,|\!\}$ for $i \in \{1, \ldots, n\}$ or $\rho_0(R) \in dom(\mathcal{S})$ and $\rho_0'(R) <_{\mathcal{A}} \rho_0(R)$. For the subcase $\rho_0'(R) = f(r_1', \ldots, r_n')$ and $\rho_0(R) = f(r_1, \ldots, r_n)$ such that $\mathcal{I}(\mathcal{A})\{\!|\, r_i' \,|\!\} = \mathcal{I}(\mathcal{A})\{\!|\, r_i \,|\!\}$ for $i \in \{1, \ldots, n\}$, again since $\mathcal{S}$ is normal, $(l, f(r_1', \ldots, r_n'))$ has been checked and cannot be a witness. Thus, for $(l, f(r_1, \ldots, r_n))$ to be a witness, at least one of the pairs (r_i', r_i) has to be a witness. This is smaller than (l, r) since the recipes r_i', r_i produce proper subterms of the message $concr\{\!|\, l \,|\!\}$. For the subcase $\rho_0(R) \in dom(\mathcal{S})$ and $\rho_0'(R) <_{\mathcal{A}} \rho_0(R) = l'$, we have that (l, l') is a witness and $l \simeq \rho_0'(R)$, so $(\rho_0'(R), l')$ must be a witness, and this is smaller because $\rho_0'(R) <_{\mathcal{A}} l' <_{\mathcal{A}} l$.

Thus for every witness we can find a smaller witness, which is impossible along a well-founded ordering, and thus we can be sure that there are no witnesses. $\square$

Theorem IV.2. *Let $\mathcal{S}$ be a normal symbolic state. Then $\mathcal{S}$ satisfies privacy iff $\mathcal{S}$ is consistent.*

Proof. Let $\mathcal{P} = \{(0, \phi_1, \mathcal{A}_1, \_, \_, \_), \ldots, (0, \phi_n, \mathcal{A}_n, \_, \_, \_)\}$ be the possibilities in $\mathcal{S}$. First we assume that $\mathcal{S}$ satisfies privacy and show that $\mathcal{S}$ is consistent. Let $S = (\alpha, \_, \_, \_) \in [\![\mathcal{S}]\!]$, $\beta \equiv MMA(S)$ and $\mathcal{I} \models_{\Sigma_0} \alpha$. Since $\mathcal{S}$ satisfies privacy, (α, β)-privacy holds so there exists $\mathcal{I}' \models_{\Sigma} \beta$ such that $\mathcal{I}$ and $\mathcal{I}'$ agree on $fv(\alpha)$ and on the relations in Σ_0. Since $\beta \models \beta_0$, $\mathcal{I}' \models_{\Sigma_0} \beta_0$. Therefore (α, β_0)-privacy holds. Thus $\mathcal{S}$ is consistent.

Next we assume that $\mathcal{S}$ is consistent and show that $\mathcal{S}$ satisfies privacy. Let $S = (\alpha, \beta_0, \_, \_) \in [\![\mathcal{S}]\!]$, ρ be the ground choice of recipes defining S and $concr$ be the concrete frame in S. Let $\beta \equiv MMA(S)$, $struct_i = \rho(\mathcal{A}_i)$ for $i \in \{1, \ldots, n\}$ and $\mathcal{I} \models_{\Sigma_0} \alpha$. Since $\mathcal{S}$ is consistent, (α, β_0)-privacy holds, i.e., there exists $\mathcal{I}' \models_{\Sigma_0} \beta_0$ such that $\mathcal{I}$ and $\mathcal{I}'$ agree on $fv(\alpha)$ and the relations in Σ_0. Since $\alpha \wedge \beta_0 \models \bigvee_{i=1}^{n} \phi_i$, there exists $i \in \{1, \ldots, n\}$ such that $\mathcal{I}' \models \phi_i$. By Lemma IV.1, $concr \sim \mathcal{I}'(struct_i)$ so $\mathcal{I}' \models_{\Sigma} concr \sim struct_i$. Therefore $\mathcal{I}' \models_{\Sigma} \beta$, so (α, β)-privacy privacy holds, i.e., S satisfies privacy. This is true for every $S \in [\![\mathcal{S}]\!]$, thus $\mathcal{S}$ satisfies privacy. $\square$

The following lemma is used to prove the termination of the compose-checks in the next theorem, and we then show that these intruder experiments are correct.

Lemma A.7. *Let $\mathcal{A}$ be a simple FLIC, r_1, r_2 be recipes and $\sigma = mgu(\mathcal{A}\{\!|\, r_1 \,|\!\} \doteq \mathcal{A}\{\!|\, r_2 \,|\!\})$.*

- *If $isPriv(\sigma)$, then for every choice of recipes ρ, we have $isPriv(\sigma')$, where $\sigma' = mgu(\rho(\mathcal{A})\{\!|\, \rho(r_1) \,|\!\} \doteq \rho(\mathcal{A})\{\!|\, \rho(r_2) \,|\!\})$.*
- *If not $isPriv(\sigma)$, then for every $\rho \in LI(\mathcal{A}, \sigma)$, we have $isPriv(\sigma')$, where $\sigma' = mgu(\rho(\mathcal{A})\{\!|\, \rho(r_1) \,|\!\} \doteq \rho(\mathcal{A})\{\!|\, \rho(r_2) \,|\!\})$.*

Proof. First we consider the case that $isPriv(\sigma)$. Let ρ be a choice of recipes and $\sigma' = mgu(\rho(\mathcal{A})\{\!|\, \rho(r_1) \,|\!\} \doteq \rho(\mathcal{A})\{\!|\, \rho(r_2) \,|\!\})$. If $\mathcal{A}\{\!|\, r_1 \,|\!\}$ contains an intruder variable as a subterm, then $\mathcal{A}\{\!\!\{ r_2 \}\!\!\}$ contains the same intruder variable in the same position; otherwise, the intruder variable would be substituted and we would not have $isPriv(\sigma)$. The argument is similar if $\mathcal{A}\{\!\!\{ r_2 \}\!\!\}$ contains intruder variables. Since the intruder variables are not relevant for unifying the two messages, the intruder variables can be instantiated in any way. Then we have $\sigma(\rho(\mathcal{A})\{\!\!\{ \rho(r_1) \}\!\!\}) = \sigma(\rho(\mathcal{A})\{\!\!\{ \rho(r_2) \}\!\!\})$, which means that σ is an instance of σ' and thus $isPriv(\sigma')$.

Next we consider the case that not $isPriv(\sigma)$. Let $\rho \in LI(\mathcal{A}, \sigma)$, $\sigma' = mgu(\rho(\mathcal{A})\{\!\!\{ \rho(r_1) \}\!\!\} \doteq \rho(\mathcal{A})\{\!\!\{ \rho(r_2) \}\!\!\})$ and $\mathcal{A}'$, σ'' be such that $(\varepsilon, \sigma(\mathcal{A}), \sigma) \rightsquigarrow^* (\rho, \mathcal{A}', \sigma'')$ and $\mathcal{A}'$ is simple. By definition of ρ and $\mathcal{A}'$, $\mathcal{A}'\{\!\!\{ \rho(r_1) \}\!\!\} = \mathcal{A}'\{\!\!\{ \rho(r_2) \}\!\!\}$. Moreover, $\rho(\mathcal{A})$ and $\mathcal{A}'$ are the same up to renaming of intruder variables and up to substitution of privacy variables, so there exists a substitution τ such that $isPriv(\tau)$ and $\tau(\rho(\mathcal{A})\{\!\!\{ \rho(r_1) \}\!\!\}) = \tau(\rho(\mathcal{A})\{\!\!\{ \rho(r_2) \}\!\!\})$. Note that we have $isPriv(\tau)$ because the fresh intruder variables, i.e., the ones that are introduced by ρ, are just renamed compared to the intruder variables in $\mathcal{A}'$. Then τ is an instance of σ' and thus $isPriv(\sigma')$. $\square$

Theorem A.8 (Compose-check termination)**.** *Let $\mathcal{S}$ be a symbolic state. Then there is a finite number of symbolic states $\mathcal{S}'$ such that $\mathcal{S} \rightarrowtail^* \mathcal{S}'$.*

Proof. Let $\mathcal{A}_1, \dots, \mathcal{A}_n$ be the FLICs in $\mathcal{S}$. We define the weight of $\mathcal{S}$ to be the pair (p, s), where

- p is the number of pairs recipes to check: $p = \#Pairs(\mathcal{S})$; and
- s is the sum, over the pairs of recipes, of the number of FLICs in which the unifier depends on intruder variables and there exists a solution to the constraints: $s = \sum_{(l,r)\in Pairs(\mathcal{S})} \#\{\mathcal{A}_i \mid \text{not } isPriv(\sigma_i) \text{ and } LI(\mathcal{A}_i, \sigma_i) \neq \emptyset\}$, where $\sigma_i = mgu(\mathcal{A}_i\{\!\!\{ l \}\!\!\} \doteq \mathcal{A}_i\{\!\!\{ r \}\!\!\})$ for $i \in \{1, \dots, n\}$ (and $(l, r) \in Pairs(\mathcal{S})$).

The weights with the lexicographic order form a well-founded ordering. Every rule decreases the weight. Let $\mathcal{S}'$ be a symbolic state such that $\mathcal{S} \rightarrowtail \mathcal{S}'$. First we consider that $\mathcal{S}'$ is produced by the rule **Privacy split**. One pair (l, r) is now checked and the FLICs are not changed, so p decreases.

Next we consider the case that $\mathcal{S}'$ is produced by the rule **Recipe split**. There exist $(l, r) \in Pairs(\mathcal{S})$ and $i \in \{1, \dots, n\}$ such that not $isPriv(\sigma_i)$ and $LI(\mathcal{A}_i, \sigma_i) \neq \emptyset$, where $\sigma_i = mgu(\mathcal{A}_i\{\!\!\{ l \}\!\!\} \doteq \mathcal{A}_i\{\!\!\{ r \}\!\!\})$. The first subcase is that $\mathcal{S}'$ is produced by applying some choice of recipes $\rho \in LI(\mathcal{A}_i, \sigma_i)$. For every pair $(l', r') \in Pairs(\mathcal{S})$, there is at most one corresponding pair $(l', \rho(r')) \in Pairs(\mathcal{S}')$ so p may decrease (e.g., if some choice of recipes used to compute the pairs in $\mathcal{S}'$ is not an instance of ρ) but p cannot increase. By Lemma A.7, if the unifier only depends on privacy variables, this is still the case in $\mathcal{S}'$, and for the FLIC $\rho(\mathcal{A}_i)$, the unifier does not depend on intruder variables anymore, thus s decreases.

The second subcase is that $\mathcal{S}'$ is produced by excluding σ_i. Then the FLICs are not changed so p stays the same, but s decreases because now $LI(\mathcal{A}_i, \sigma_i) = \emptyset$, since σ_i is excluded.

There cannot be an infinite sequence of decreasing weights so the compose-checks terminate. $\square$

Theorem V.1 (Compose-check correctness)**.** *Let $\mathcal{S}$ be a finished symbolic state, $(l, r) \in Pairs(\mathcal{S})$ and $\{\mathcal{S}_1, \dots, \mathcal{S}_n\}$ be the symbolic states after one rule application given the pair (l, r). Then $[\![\mathcal{S}]\!] = \biguplus_{i=1}^{n} [\![\mathcal{S}_i]\!]$, where $\biguplus$ denotes the disjoint union. Moreover, there is a finite number of $\mathcal{S}'$ such that $\mathcal{S} \rightarrowtail^* \mathcal{S}'$ and $\mathcal{S}'$ is normal.*

Proof. Let $\mathcal{P}$ denote the possibilities in $\mathcal{S}$, where $\mathcal{P} = \{(0, \phi_1, \mathcal{A}_1, \_, \_, \_), \dots, (0, \phi_n, \mathcal{A}_n, \_, \_, \_)\}$. First we consider the case that **Privacy split** is applicable. For every $i \in \{1, \dots, n\}$, $isPriv(\sigma_i)$ or $LI(\mathcal{A}_i, \sigma_i) = \emptyset$, where $\sigma_i = mgu(\mathcal{A}_i\{\!\!\{ l \}\!\!\} \doteq \mathcal{A}_i\{\!\!\{ r \}\!\!\})$. We are partitioning the set of ground states based on the interpretations of privacy variables. Let $\mathcal{S}_1$ and $\mathcal{S}_2$ be the symbolic states produced by the first and second subcase of the rule, respectively. We start by showing that $[\![\mathcal{S}]\!] \subseteq [\![\mathcal{S}_1]\!] \uplus [\![\mathcal{S}_2]\!]$. Let $S = (\alpha, \beta_0, \gamma, \mathcal{P}') \in [\![\mathcal{S}]\!]$, ρ be the ground choice of recipes defining S and $concr = \gamma(struct_i)$ for some $i \in \{1, \dots, n\}$ be the concrete frame in S, where $struct_j = \rho(\mathcal{A}_j)$ for $j \in \{1, \dots, n\}$. Let $\beta \equiv MMA(S)$.

- If $isPriv(\sigma_i)$ and $\gamma \models \sigma_i$: Then we show that $S \in [\![\mathcal{S}_1]\!]$. Define
$$\beta' \equiv \beta \wedge \bigwedge_{j=1}^{n} \left(\phi_j \Rightarrow \begin{cases} \sigma_j & \text{if } isPriv(\sigma_j) \\ \mathsf{false} & \text{otherwise} \end{cases} \right)$$
We need to show that $\beta \equiv \beta'$. Let $\mathcal{I} \models_\Sigma \beta$. There exists $j \in \{1, \dots, n\}$ such that $\mathcal{I} \models_\Sigma \phi_j \wedge concr \sim struct_j$. Since $\gamma \models \sigma_i$ and $concr = \gamma(\rho(\mathcal{A}_i))$, $concr\{\!\!\{ l \}\!\!\} = concr\{\!\!\{ r \}\!\!\}$. Then $\mathcal{I}(struct_j)\{\!\!\{ l \}\!\!\} = \mathcal{I}(struct_j)\{\!\!\{ r \}\!\!\}$, so $\mathcal{I} \models \sigma_j$. Then $\mathcal{I} \models_\Sigma \phi_j \wedge \sigma_j \wedge concr \sim struct_j$, so $\mathcal{I} \models_\Sigma \beta'$. Conversely, for every $\mathcal{I} \models_\Sigma \beta'$, we have $\mathcal{I} \models_\Sigma \beta$. Thus $\beta \equiv \beta'$.
- Otherwise, we show that $S \in [\![\mathcal{S}_2]\!]$. Define
$$\beta' \equiv \beta \wedge \bigwedge_{j=1}^{n} \left(\phi_j \Rightarrow \begin{cases} \neg\sigma_j & \text{if } isPriv(\sigma_j) \\ \mathsf{true} & \text{otherwise} \end{cases} \right)$$
We need to show that $\beta \equiv \beta'$. Let $\mathcal{I} \models_\Sigma \beta$. There exists $j \in \{1, \dots, n\}$ such that $\mathcal{I} \models_\Sigma \phi_j \wedge concr \sim struct_j$. Since $\gamma \models \neg\sigma_i$ or $LI(\mathcal{A}_i, \sigma_i) = \emptyset$, $concr\{\!\!\{ l \}\!\!\} \neq concr\{\!\!\{ r \}\!\!\}$. Then $\mathcal{I}(struct_j)\{\!\!\{ l \}\!\!\} \neq \mathcal{I}(struct_j)\{\!\!\{ r \}\!\!\}$, so if $isPriv(\sigma_j)$ then $\mathcal{I} \models_\Sigma \phi_j \wedge \neg\sigma_j \wedge concr \sim struct_j$. Then $\mathcal{I} \models_\Sigma \beta'$. Conversely, for every $\mathcal{I} \models_\Sigma \beta'$, we have $\mathcal{I} \models_\Sigma \beta$. Thus $\beta \equiv \beta'$.

The cases are mutually exclusive, so $[\![\mathcal{S}]\!] \subseteq [\![\mathcal{S}_1]\!] \uplus [\![\mathcal{S}_2]\!]$. Similarly, we have $[\![\mathcal{S}_1]\!] \uplus [\![\mathcal{S}_2]\!] \subseteq [\![\mathcal{S}]\!]$.

Next we consider the case that **Recipe split** is applicable. There exists $i \in \{1, \dots, n\}$ such that not $isPriv(\sigma_i)$ and $LI(\mathcal{A}_i, \sigma_i) = \{\rho_1, \dots, \rho_k\}$, where $\sigma_i = mgu(\mathcal{A}_i\{\!\!\{ l \}\!\!\} \doteq \mathcal{A}_i\{\!\!\{ r \}\!\!\})$. We are partitioning the set of ground states based on the ground choices of recipes. Let $\mathcal{S}_j = \rho_j(\mathcal{S})$ for $j \in \{1, \dots, k\}$, and $\mathcal{S}'$ be the symbolic state in which σ_i is excluded for $\mathcal{A}_i$. Let $S \in [\![\mathcal{S}]\!]$ and ρ be the corresponding ground choice of recipes. Then $S \in [\![\mathcal{S}_j]\!]$ if ρ is represented by ρ_j (note that the ρ_j are mutually exclusive); otherwise $S \in [\![\mathcal{S}']\!]$. Conversely, $[\![\mathcal{S}']\!] \uplus \biguplus_{j=1}^{n} [\![\mathcal{S}_j]\!] \subseteq [\![\mathcal{S}]\!]$.

The termination follows from Theorem A.8. $\square$

Theorem VI.1 (Analysis correctness)**.** *For a symbolic state $\mathcal{S}$, the destructor oracle application strategy produces in finitely many steps a set $\{\mathcal{S}_1, \ldots, \mathcal{S}_n\}$ of symbolic states that are analyzed. Further, for every ground state $S \in [\![\mathcal{S}]\!]$ there exists $S' \in [\![\mathcal{S}_i]\!]$, for some $i \in \{1, \ldots, n\}$, such that S and S' are equivalent except that the frames in S' may contain further shorthands; and vice versa, for every $S' \in [\![\mathcal{S}_i]\!]$ there exists $S \in [\![\mathcal{S}]\!]$ such that S' is equivalent to S except for shorthands.*

Proof. It is quite straightforward to see that all states that we reach by analysis steps are equivalent modulo the augmentation with shorthands: the intruder learns only terms that could be obtained with access to destructors anyway, and none of the transactions puts a constraint on the intruder since in the worst case the decryption fails and the intruder just does not learn anything from it.

For termination, we define a measure (a, b) for symbolic states $\mathcal{S}$ as a lexicographical ordering of the following two well-founded components a and b:

- a is the total number of $\star$ marks and $+$ marks in the FLICs.
- b is the total number of $\star$ marks in the FLICs.

Consider going from a symbolic state $\mathcal{S}$ to $\mathcal{S}'$ with a destructor oracle transaction according to our strategy. We show that on the transition from $\mathcal{S}$ to $\mathcal{S}'$ the measure can only decrease. In an intermediate state of the symbolic execution, when we evaluate the try-catch, we split each possibility into two further cases (the one where the try succeeds, and where it fails), but from the snd steps only one possibility survives — the intruder observes from the outcome whether the destructor works or not. Thus the number of possibilities can only remain the same or decrease from $\mathcal{S}$ to $\mathcal{S}'$. (We have a decrease if in some FLICs the decryption works and in others not, because then each $\mathcal{S}'$ is reduced either to those that worked or those that did not.) Any instantiations of intruder variables that happen are neutral for the measure, because intruder variables in received messages are already marked ✓, and thus also the instantiation is marked ✓. The only changes in the measures are from updating the mark of the term under analysis and the marking of the newly received terms (i.e., the result of the analysis and the decryption key that is repeated by the oracle).

We now distinguish the two cases whether $\mathcal{S}'$ represents a successful decryption or failure (w.r.t. the destructor oracle rule that brings us from $\mathcal{S}$ to $\mathcal{S}'$).

In the first case, if the destructor fails, then in every FLIC where l maps to a term marked $\star$, we replace it by $+$ (others we leave alone). This does not change the a measure, but reduces the b measure by at least one (since there was at least one $\star$-marked term we have addressed).

In the second case, if the destructor is successful, let us consider decryption again. In every FLIC where the label l maps to $c(k', t_1, \ldots, t_n)$ marked $\star$, recall that the strategy marks the newly received $l' \mapsto t_i$ with the same mark as the respective subterm t_i in l; in turn the term $c(k', t_1, \ldots, t_n)$ with all its subterms gets marked ✓ (and similarly in a transparency rule). This reduces the a measure by at least one: even if $l' \mapsto t_i$ now contains several $\star$ or $+$ marks, these marks were counted in the previous marking of $l \mapsto c(k', t_1, \ldots, t_n)$, which is now marked with ✓ for c and the subterms, so the mark $\star$ that c bore is not counted anymore. If there are any FLICs where l is mapped to a term marked $+$ or ✓, we do not necessarily have a reduction, but new l'-terms can only contain $\star$ and $+$ marks that are removed from l. Since there is always at least one $\star$-marked term in $\mathcal{S}$ to apply a destructor oracle rule, the a measure is strictly reduced from $\mathcal{S}$ to $\mathcal{S}'$.

The measure is well-founded and thus proves there is no infinite chain of analysis steps, and since the branching is also finite (because applying a transaction leads to finitely many successor states), it thus follows by Kőnig's lemma that for every state $\mathcal{S}$, we obtain a finite number of analyzed states $\mathcal{S}_1, \ldots, \mathcal{S}_n$ with the destructor oracle strategy. $\square$

Lemma VI.2. *Let $\mathcal{S}$ be a normal analyzed state, $S \in [\![\mathcal{S}]\!]$ and r be any recipe over the domain of S. Then there is a destructor-free recipe r' such that $struct\{\!\!\{ r \}\!\!\} \approx struct\{\!\!\{ r' \}\!\!\}$ in every frame $struct$ of S.*

Proof. Note that this proof works on a ground state S which does not contain intruder variables anymore (but still privacy variables). Thus, the FLICs are now frames that contain just incoming messages. We also formulate this only for decryption, transparency is in all cases very similar.

We have to show how to replace any subterm $r_d = d(r_1, r_2)$ of r with a destructor-free equivalent recipe. We can also w.l.o.g. assume that r_1 and r_2 are destructor-free (by starting with the inner-most occurrence of a destructor). Thus r_2 is either a label or a composed recipe:

1) Case $r_2 = c(r'_1, \ldots, r'_n)$ for some public function c. If c is not a constructor corresponding to destructor d, then we can already replace r_d with $\mathtt{ff}$ and are done. Otherwise r_d means the intruder applies a destructor to a term they constructed themselves. We distinguish three subcases:
   a) If r_d does not yield $\mathtt{ff}$ in any frame, then the result of the destructor must be the ith subterm (for some $i \in \{1, \ldots, n\}$) of r_2 in every frame, i.e., $struct\{\!\!\{ r_d \}\!\!\} \approx struct\{\!\!\{ r'_i \}\!\!\}$ for every FLIC $struct$, and we can thus replace r_d by r'_i.
   b) If r_d yields $\mathtt{ff}$ in all frames, i.e. $struct\{\!\!\{ r_d \}\!\!\} \approx \mathtt{ff}$ in every frame $struct$, we can just replace r_d by $\mathtt{ff}$.
   c) If r_d yields $\mathtt{ff}$ in some frame $struct_1$ and does not yield $\mathtt{ff}$ in another frame $struct_2$, it means that comparing r_d with $\mathtt{ff}$ is an intruder experiment that distinguishes the frames. We show that this contradicts the fact that $\mathcal{S}$ is analyzed and normal. The only reason that $struct_1$ and $struct_2$ give different results is that the encryption and decryption key

do not match in $struct_1$ but do match in $struct_2$. Recall that in a decryption rule with decryption key k and encryption key k', we require that either $k = k'$ or $k \approx f(k')$ or $k' \approx f(k)$ for some public function f. If $k = k'$, then comparing r_1 with r_1' is an experiment that distinguishes the frames, which contradicts that $\mathcal{S}$ is normal. Otherwise, we only prove the case $k \approx f(k')$, the other case is analogous. In $struct_2$, r_1 and r_1' correspond to k and k', respectively. Thus, comparing r_1 with $f(r_1')$ is also an experiment that distinguishes the frames. If f is a constructor, this directly contradicts that $\mathcal{S}$ is normal. If f is a destructor, we now show that this has already been analyzed, i.e., there must be a label l' that is a shorthand for $f(r_1')$ and thus this contradicts that $\mathcal{S}$ is normal (because then the intruder has already compared r_1 with l'). If r_1' is a label, then directly the analysis rule $f(r_1')$ must have been applied; if $r_1' = c(r_1'', \ldots, r_n'')$ and since f is unary, c is transparent, i.e., it is directly equivalent to one of r_i''. Thus the experiment to compare r_1 with r_i'' already distinguishes the frames and that must have been done already since $\mathcal{S}$ is normal and these recipes are destructor-free. Thus, in all cases this contradicts that $\mathcal{S}$ is normal.

2) Case $r_2 = l$ for a label l. We distinguish two subcases:
   a) Case l maps to a term t in at least one of the frames such that t was at some point marked $\star$, i.e., t is a term for which a destructor exists and the respective destructor rule has been tried for l by the analysis strategy. (The other cases being that the t in every frame is marked ✓, because it has no destructor or originated from the intruder.) The state resulting from the application of the respective destructor oracle rule has the property that the destructor either succeeded in all frames or failed in all frames. In the case of failure, we can simply replace r_d by $\mathtt{ff}$ and are done. In the case of success, there are labels holding the result of the destructor, say, l_1 for decryption result and l_2 repeating the decryption key if it is a decryption rule. (For the case of transparency the proof is similar.) One may wonder if comparing r_1 with l_2 could distinguish the frames. This would contradict that $\mathcal{S}$ is normal because r_1 and l_2 have no destructors. Thus, $struct\{\!| r_1 |\!\} \approx struct\{\!| l_2 |\!\}$ in every frame $struct$, and thus $struct\{\!| r_d |\!\} \approx struct\{\!| l_1 |\!\}$ and we can replace r_d by l_1.
   b) Case l maps in all frames to terms that have been marked ✓ throughout. If they are all terms that have no destructor, then we can of course directly replace r_d with $\mathtt{ff}$. Otherwise, in at least one frame $struct$, l maps to a term $c(s_1, \ldots, s_m)$ which was composed by the intruder, i.e., there are destructor-free recipes $r_1', \ldots, r_m'$ that produce s_i in $struct$, thus $struct\{\!| l |\!\} = struct\{\!| c(r_1', \ldots, r_m') |\!\}$. As these recipes are all destructor-free, this is an experiment that must work in all frames (otherwise $\mathcal{S}$ is not normal). Thus, we can first replace $r_d = d(r_1, c(r_1', \ldots, r_m'))$ which then can be reduced to a destructor-free recipe following the case 1) of this proof.

□

Lemma VI.3. *Let $\mathcal{S}$ be an analyzed state and normal. Then it is also normal w.r.t. arbitrary recipes.*

Proof. Suppose $\mathcal{S}$ is analyzed and normal w.r.t. destructor-free recipes, and let $S \in [\![\mathcal{S}]\!]$. Suppose there are recipes r_1 and r_2 with destructors such that comparing r_1 and r_2 is an experiment that distinguishes $concr$ from a $struct_i$ in S, then by Lemma VI.2, there exist equivalent destructor-free r_1' and r_2' that thus also distinguish $concr$ and $struct_i$ and thus S (thus $\mathcal{S}$) is not normal w.r.t. destructor-free recipes. □

Theorem VI.4 (Procedure correctness)**.** *Given a protocol specification for (α, β)-privacy, a bound on the number of transitions and an algebraic theory allowed by Definition VI.1, our decision procedure is sound, complete and terminating.*

Proof. This is essentially lifting Proposition A.1 to the case where the intruder has access to destructors (except private extractors, of course). A problem is however that the states that our lifting produces include shorthands, i.e., the terms obtained from the destructor oracle rules. The construction ensures that such shorthands are indeed just shorthands in the sense that each corresponds to a recipe with destructor (that gives the same term in each FLIC as the shorthand). We can thus regard a state with shorthands as an equivalent representation of the state without shorthands.

Let now $\mathcal{S}$ be a symbolic state that is analyzed and normal w.r.t. destructor-free recipes. By Lemma VI.3, it is also normal w.r.t. arbitrary recipes. In the model where destructors are private, by Proposition A.1, we have for transaction process P that $[\![\mathcal{S}_{\underset{P}{\Longrightarrow}}]\!] = [\![\mathcal{S}]\!]_{\underset{P}{\longrightarrow}}$, i.e., what is reachable on the symbolic level is equivalent to what is reachable on the ground level using P. We now show how to arrive at the same result for the case where the intruder can access destructors (except private extractors). Consider first the recipes for messages that the intruder may send during this transaction. These recipes can only use labels that already occur in $\mathcal{S}$ — whatever messages the process sends out in response is not available to the intruder when sending. Given a ground state $S \in [\![\mathcal{S}]\!]$ and some recipes with destructors that the intruder sends during this transition, they are equivalent to destructor-free recipes due to Lemma VI.2. Thus, $[\![\mathcal{S}]\!]_{\underset{P}{\longrightarrow}}$ is the same when allowing destructors in recipes the intruder sends for the messages that P receives.

Observe that the symbolic states $\mathcal{S}_{\underset{P}{\Longrightarrow}}$ that are reached from $\mathcal{S}$ with P are not yet analyzed and only normalized w.r.t. destructor-free experiments. By applying the destructor oracle strategy to every symbolic state in $\mathcal{S}_{\underset{P}{\Longrightarrow}}$, we obtain finitely many analyzed states $\mathcal{S}_1, \ldots, \mathcal{S}_n$ such that $[\![\mathcal{S}_{\underset{P}{\Longrightarrow}}]\!] = \bigcup_{i=1}^{n} [\![\mathcal{S}_i]\!]$ by Theorem VI.1. By Lemma VI.3 these symbolic states in $\mathcal{S}_1, \ldots, \mathcal{S}_n$ are also normal w.r.t. recipes with destructors.

Thus, starting at a normal analyzed symbolic state $\mathcal{S}$ and given a transaction process P, our procedure computes a finite set of normal analyzed symbolic states that represent exactly those states that can be reached on the ground level with P from any state represented by $\mathcal{S}$. Thus, by repeatedly applying this procedure, we obtain a correct finite representation of all states reachable from $\mathcal{S}$ after a given number of transactions. $\square$

C. Decidability of Our Fragment of Herbrand logic

We support the fragment such that:

- The alphabet Σ_0 is finite (in particular, there are finitely many constants).
- The equivalence class $[t]_E$ of every Σ_0-term t is computable (and thus finite).
- Every variable x (both bound and unbound) must range over a fixed domain of constants, $dom(x) \subseteq \Sigma_0^0$.

Before giving a decision procedure, we first need some definitions. Given the Herbrand universe U induced by Σ_0 and given α, we define the *relevant part of U for α* as follows:

$$U_0^\alpha = \{[\sigma(t_i)]_E \mid R(t_1, \ldots, t_n) \text{ occurs in } \alpha \text{ and} \\ \text{for all } x \in fv(t_1, \ldots, t_n), \sigma(x) \in dom(x)\}$$

We say that θ is an *interpretation representation* (w.r.t. α) iff θ maps every $x \in fv(\alpha)$ to some element of $dom(x)$ and every n-ary relation symbol R to a subset of $(U_0^\alpha)^n$. We say that θ *represents* interpretation $\mathcal{I}$ iff $\theta(x) = \mathcal{I}(x)$ for every $x \in fv(\alpha)$ and $\vec{t} \in \theta(R)$ iff $\vec{t} \in \mathcal{I}(R)$ for every n-ary relation symbol R and $\vec{t} \in (U_0^\alpha)^n$.

We now describe an algorithm that, given α, returns the set of all interpretation representations that represent a model of α (which implies a decision procedure for the model relation). We first compute all interpretation representations for α. This is finite since there are only finitely many variables and they have finite domains; moreover, U_0^α is finite, since finitely many relations $R(t_1, \ldots, t_n)$ are used, their variables can range over finitely many values, and the equivalence classes of every term is finite. Thus there are finitely many possible interpretations of every R over U_0^α. For a given interpretation representation θ, we can check the model relation with α as follows:

$$\begin{array}{rll}
\theta \models s \doteq t & \text{iff} & [\theta(s)]_E = [\theta(t)]_E \\
\theta \models R(t_1, \ldots, t_n) & \text{iff} & ([\theta(t_1)]_E, \ldots, [\theta(t_n)]_E) \in \theta(R) \\
\theta \models \phi \land \psi & \text{iff} & \theta \models \phi \text{ and } \theta \models \psi \\
\theta \models \neg\phi & \text{iff} & \text{not } \theta \models \phi \\
\theta \models \exists x.\ \phi & \text{iff} & \text{there exists } c \in dom(x) \text{ such that} \\
& & \theta[x \mapsto c] \models \phi
\end{array}$$