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

Laouen Fernet[1], Sebastian Mödersheim[1] and Luca Viganò[2]
[1]Danmarks Tekniske Universitet, Kgs. Lyngby, Denmark
[2]King's College London, London, United Kingdom
{lpkf,samo}@dtu.dk, 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.

Keywords: Privacy, Formal Methods, Security Protocols, Automated Verification.

1 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 be alerted by a counter-example. 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 always 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 the definition of a decision procedure for the full notion of transaction processes defined by [2] for constructor/destructor theories. 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 [4]. This is similar to the bounds needed in tools like APTE [5], AKiSs [6], SPEC [7, 8] and DeepSec [9]. 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 Z3 [10] 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, there are tools like ProVerif [11] and Tamarin [12] that do handle unbounded sessions but that require the restriction to so-called *diff-*

equivalence [13, 14], which drastically limits the use of conditional branching in security protocols. 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 §2, we give a high-level overview of our decision procedure. In §3, we define how we symbolically represent messages sent by the intruder and how to solve constraints with the *lazy intruder* rules. In §4, we introduce the notions of nodes and symbolic states with their semantics. In §5, we explain how the intruder can perform experiments and make logical deductions relevant for privacy by comparing messages in their knowledge. In §6, we summarize how the different parts of the procedure are integrated. In §7, we discuss the prototype tool we have developed and its application to some examples. In §8, we conclude by discussing future work. The appendix contains additional technical details and the proofs of correctness.

2 Overview of the Procedure and Preliminaries

In this section, we give a high-level overview of our decision procedure. [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 any 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.

2.1 (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* [15], 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 in examples and define below a class of properties that our result supports. 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.

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

Figure 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 will omit the alphabet and write $\models$ for $\models_{\Sigma_0}$ unless explicitly written.

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 2.1 (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 $F_1\{\!|\, r_1 \,|\!\} \approx F_1\{\!|\, r_2 \,|\!\} \Leftrightarrow F_2\{\!|\, r_1 \,|\!\} \approx F_2\{\!|\, r_2 \,|\!\}$. Intuitively, this means that the intruder cannot *distinguish* F_1 and F_2, because any experiment they can make (i.e., comparing the outcome of two computations r_1 and r_2) either gives in both frames the same result or in both frames different results.

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 $\beta \equiv \alpha \wedge \mathit{concr} \sim \mathit{struct}$ (see [1] for details on formalizing frames in Herbrand logic), where *struct* contains only variables from α and $\mathit{concr} = \mathcal{I}(\mathit{struct})$ for one interpretation $\mathcal{I} \models \alpha$.

As an example, let $\alpha \equiv x_1, x_2 \in \{0, 1\}$, $\mathit{struct} = l_1 \mapsto \mathsf{h}(k, x_1).l_2 \mapsto \mathsf{h}(k, x_2)$ and $\mathit{concr} = l_1 \mapsto \mathsf{h}(k, 0).l_2 \mapsto \mathsf{h}(k, 1)$. Observe that there are four models $\mathcal{I} \models \alpha$, and in two of them $\mathit{concr} \sim \mathcal{I}(\mathit{struct})$ while $\mathit{concr} \not\sim \mathcal{I}(\mathit{struct})$ in the other two. The goal of the intruder is to rule out models that are not consistent with β. Note that β *requires* $\mathit{concr} \sim \mathit{struct}$: the intruder knows that *concr* is an instance of *struct* and thus any experiment must yield the same result under the actual model $\mathcal{I} \models \alpha$ such that $\mathit{concr} = \mathcal{I}(\mathit{struct})$. 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 $\mathcal{I}_1(\mathit{struct}) \sim \cdots \sim \mathcal{I}_n(\mathit{struct})$ (note that in such problems $\mathit{fv}(\alpha) = \mathit{fv}(\beta)$). [3] gives a more efficient procedure that

[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.

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.

2.2 (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 now some adaptations.

For our 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, 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 destructors and constructors 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 subterms without any key (but recall that inv is not a public function).

Definition 2.2 (Protocol specification)**.** *A* protocol specification *consists of*

- *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* $\mathsf{cell}(t) = C[t]$*; and*
- *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.*

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

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

where mode *is either* $\star$ *or* $\diamond$ *and* d *is a destructor. 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 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:*

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

Observe that privacy variables are introduced only by non-deterministic choices $\mathsf{mode}\ x \in D$. 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.

The constructs rcv, snd, ν, and 0 are standard, as well as reading from and writing to memory cells. For the formula ϕ in a condition or a release we have to make a restriction here for our decision procedure:

ϕ		*Formula*
::=	$t \doteq t$	*Comparison between messages*
|	$\neg\phi$	*Negation*
|	$\phi \wedge \phi$	*Conjunction*
|	$R(t,\ldots,t)$	*Relation*

where for releases the formulas may contain the meta-notation $\gamma(t)$ for a message t. The meta-notation $\gamma(\cdot)$ refers to the fact that in every state, the reality (i.e., the actual values of privacy variables) is defined by a ground interpretation. Thus, for instance, releasing $\star\ x \doteq \gamma(x)$ means allowing the intruder to learn the true value of x.

As syntactic sugar, we may write $\mathsf{let}\ X = t.P$ for the substitution of all occurrences of X in P by t. In a destructor application or conditional statement, we may omit the catch or else branch when it only contains the nil process. In a formula, the disjunction $\vee$ is standard sugar and $x \in \{c_1,\ldots,c_n\}$ is sugar for $x \doteq c_1 \vee \cdots \vee x \doteq c_n$. We may write P for $\nu.P$ or $P.0$. We define the application of a substitution σ to a process P by applying σ to all messages in P, except in the formulas released.[2]

[2]For example, if $\sigma = [x \mapsto c]$ and P contains the release $\star\ x \doteq \gamma(x)$, the formula must not be substituted because the payload should be about x and not become $c \doteq \gamma(c)$, which is equivalent to true and would miss the point of releases.

Requirements on transactions. The protocol specification must fulfill the following requirements: a bound variable cannot be instantiated a second time later; the only place destructors are allowed is in a destructor application with try; in different branches of a destructor application or conditional statement, the same privacy variables are chosen in the same order, from the same domains and with the same mode, and the ordering of receive steps is also the same; the transaction processes must be *closed*, which means that they have no free variables.

Moreover, it is considered a specification error if at runtime, in a formula $R(t_1, \ldots, t_n)$ in a condition there are symbols which are in $\Sigma \setminus \Sigma_0$ and variables not in $fv(\alpha)$. Thus, the specification can use symbols from the technical level in a relation *as long as* the evaluated terms use only symbols in Σ_0 and $fv(\alpha)$ (i.e., the payload level) when executing the protocol. The same requirement applies to all terms in a release step $\star\ \phi$, i.e., releasing technical information in the payload is not allowed. This can be detected during the symbolic execution and means that insufficient checks are made over the terms before the conditional statement or release.

Example 2.1. Suppose that a server sends an encrypted message to a client, containing either a nonce for a positive answer or simply a negative answer. This can be formalized as the following transaction:

$$
\begin{array}{l}
\star\ x \in \mathsf{Agent}.\ \star\ y \in \{\mathsf{yes}, \mathsf{no}\}. \\
\mathsf{if}\ y \doteq \mathsf{yes}\ \mathsf{then} \\
\quad \nu n, r.\mathsf{snd}(\mathsf{crypt}(\mathsf{pk}(x), \mathsf{pair}(\mathsf{yes}, n)), r) \\
\mathsf{else} \\
\quad \nu r.\mathsf{snd}(\mathsf{crypt}(\mathsf{pk}(x), \mathsf{no}, r))
\end{array}
$$

where yes, no and the values in the set Agent are public constants and pk is a public function from agent names to public keys. ◁

Semantics. We define the semantics of transaction processes along the lines of normal process calculi, but with adaptations for (α, β)-privacy. Note that transactions are *not* executed in parallel, so there are no race conditions between cell reads and writes and we do not need a locking mechanism. The intruder is connected to all send and receive steps: a frame collects all messages sent by honest agents, and for every receive, the intruder can choose a recipe over that frame.

Our semantics needs to generate (α, β)-pairs to reflect the privacy goals, including what the intruder can actually observe about the variables from nondeterministic choices. For instance, in case of a conditional statement, the intruder in general does not know whether the condition is true during execution and thus whether the then or the else branch is taken. However, if a message is sent only in one branch, or if the messages sent in each branch are distinguishable, the intruder learns whether the condition was true. This may or may not lead to a violation of privacy.

To formalize this, [2] gives a semantics that describes a symbolic execution performed by the intruder. A *possibility* is a triple $(P, \phi, \mathit{struct})$, where P is the transaction being executed, ϕ is the conditions under which this possibility was reached (initially true) and *struct* is the structural knowledge about the messages. When $P = \mathsf{if}\ \psi\ \mathsf{then}\ P_1\ \mathsf{else}\ P_2$, the symbolic execution replaces this with two new possibilities $(P_1, \phi \wedge \psi, \mathit{struct})$ and $(P_2, \phi \wedge \neg\psi, \mathit{struct})$. In this way, the symbolic execution can split into several possibilities due to conditionals, reflecting that the intruder does not know a priori which branch was taken, but keeps track of the conditions that would lead to each branch.

Suppose that after evaluating several conditional statements, the remaining process in each possibility is either a send step or 0. The intruder can observe whether a message was actually sent or not and can thus rule out all possibilities that do not comply with this. For instance, say there was a concrete message m and two possibilities $(\mathsf{snd}(m_1), \phi_1, \mathit{struct}_1)$ and $(\mathsf{snd}(m_2), \phi_2, \mathit{struct}_2)$. In this case, we pick a new label l and augment struct_1 with $l \mapsto m_1$ and struct_2 with $l \mapsto m_2$, and we have a global frame *concr* that we augment with $l \mapsto m$. Thus, *concr* records the concrete messages actually sent and the struct_i the respective structural knowledge that the intruder at this point in general cannot tell apart yet.

This gives rise to the following generalization of the message-analysis problem:

Definition 2.3 (Multi message-analysis problem (MMA))**.** *Let $\mathcal{P}$ a set of possibilities $\{(P_1, \phi_1, \mathit{struct}_1), \ldots, (P_n, \phi_n, \mathit{struct}_n)\}$ be , α and β_0 be Herbrand formulas, $i \in \{1, \ldots, n\}$, $\mathcal{I} \models \alpha \wedge \beta_0 \wedge \phi_i$ and $\mathit{concr} = \mathcal{I}(\mathit{struct}_i)$. Define*

$$\mathit{MMA}(\alpha, \beta_0, \mathcal{P}, \mathit{concr}) = \alpha \wedge \beta_0 \wedge \bigvee_{i=1}^{n} \phi_i \wedge \mathit{concr} \sim \mathit{struct}_i \ .$$

We call the MMA well-formed *iff the ϕ_i are mutually exclusive, i.e., $\models \neg(\phi_j \wedge \phi_k)$ for $j \neq k$, and $\alpha \wedge \beta_0 \models \bigvee_{j=1}^{n} \phi_j$.*

We will deal only with well-formed *MMA*s.

Again, a naive way to decide whether a given *MMA* satisfies (α, β)-privacy is first to check every model $\mathcal{I}$ of $\alpha \wedge \beta_0$ as follows. Each $\mathcal{I}$ determines a unique index i such that $\mathcal{I} \models \phi_i$. Then we check whether $\mathit{concr} \sim \mathcal{I}(\mathit{struct}_i)$. If not, the intruder can rule that model out and can augment β_0 with $\neg\mathcal{I}$.[3] The resulting *MMA* satisfies (α, β)-privacy iff (α, β_0)-privacy holds.

Our approach. We present a procedure that generally avoids the enumeration of all models to decide (α, β)-privacy and rather uses a symbolic algorithm to identify if the intruder can make any experiments on their knowledge that would rule out some models of $\alpha \wedge \beta_0$. This is based on the following insight: by construction of the transition system, whenever in a state S we have $\mathit{concr} = \mathcal{I}(\mathit{struct}_i)$ for some interpretation $\mathcal{I}$ of the privacy variables and some

[3]Since we are dealing in our fragment with privacy variables over finite domains, we can always express models as formulas.

$struct_i$, then also for *every* model $\mathcal{I}'$ of α, a state S' is reachable that is identical with S except that $concr = \mathcal{I}'(struct_i)$. We can thus lump all states together that only differ in $concr$ but have same α and $struct_1, \ldots, struct_n$. Since every $concr$ is statically equivalent to some $struct_i$, instead of deciding static equivalence our procedure checks that no intruder experiments would rule out any instantiation of variables that α allows.

The lazy intruder. The semantics has an infinite aspect that we have not yet discussed. In a set $\mathcal{P}$ of possibilities, when one process is receiving, all are receiving (following our requirements on transactions), and w.l.o.g. we can also assume they receive the same variable X, i.e., we have:

$$\{(\mathsf{rcv}(X).P_1, \phi_1, struct_1), \ldots, (\mathsf{rcv}(X).P_n, \phi_n, struct_n)\}$$

Now the semantics says the intruder can choose any recipe r that can be built on the domain of $concr$ (respectively, the $struct_i$: they all have the same domain). The concrete message $concr\{\!|\, r \,|\!\}$ is what the transaction processes receives (but that is irrelevant due to our symbolic approach), and the structural knowledge that the intruder has about this message depends on the possibility, namely in the ith possibility it is $struct_i\{\!|\, r \,|\!\}$, and we thus have the resulting possibilities:

$$\begin{aligned}\{&(P_1[X \mapsto struct_1\{\!|\, r \,|\!\}], \phi_1, struct_1), \ldots,\\ &(P_n[X \mapsto struct_n\{\!|\, r \,|\!\}], \phi_n, struct_n)\}\ .\end{aligned}$$

The problem is that there are in general infinitely many r 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* [16, 4, 17]: it is lazy in that it avoids, as long as possible, to instantiate the variables of receive steps like 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, where 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.

To keep track of all 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; note that the order of the entries is thus significant now. In general, the messages t and recipes r 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 particular message.

In general, a FLIC represents constraints, 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 techniques like the lazy intruder, similarly to other works doing constraint-based solving with frames such as [18, 9].

Since we deal with several possibilities in parallel, we will consequently have several FLICs in parallel, replacing the $struct_i$ in the concrete model. Each FLIC will have the same sequence of incoming labels and outgoing recipes. The intruder does not know in general which possibility is the case, and rather just knows how they constructed the message from their knowledge, i.e., the recipe, which may result in a different message in each possibility. Note that, if there are different numbers of messages sent in the possibilities, it is observable to the intruder and allows for ruling out some of the possibilities.

We thus give an approach that is symbolic in three regards: (1) we deal with privacy variables that represent the different non-deterministic choices, (2) we represent all states that differ only by this choice as one symbolic state, (3) we represent the intruder inputs symbolically with another kind of variables and constraints about what is available to the intruder at the time when the represented message was constructed. This gives us a decision procedure for (α, β)-privacy for a bounded number of transitions by honest agents, but without bounding the intruder. This is restricted to specifications that fall within the fragment we present here and algebraic models with constructor/destructor theories [11, 19, 9, 20, 14]. In the appendix we provide more details and discuss the algebraic theories we support.

3 FLICs: Framed Lazy Intruder Constraints

3.1 Defining Constraints

While a frame only lists the messages *received* by an intruder, a FLIC combines this with the messages *sent* by an intruder, where the order of the steps is significant: in each send step the intruder can only use messages received before.

Definition 3.1 (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 each recipe variable* R *occurs at most once. The terms* t *are messages 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.*

We define the message $\mathcal{A}\{\!|\, r \,|\!\}$ produced *by recipe* r *in FLIC* $\mathcal{A}$ *as follows:*

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

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

Example 3.1. Suppose that we are executing a transaction where agent a receives a value X, so we add $+R \mapsto X$ to the FLIC (where both R and X are fresh variables). Here we are lazy in the sense that we do not explore at this point what R (and thus X) might be, because any value would do. Now suppose the agent checks whether X can be decrypted with the private key $\mathsf{inv}(\mathsf{pk}(\mathsf{a}))$. This is the case iff the message the intruder chooses for X has the form $\mathsf{crypt}(\mathsf{pk}(\mathsf{a}), \cdot, \cdot)$. So, in this case, we *instantiate* X in this constraint with $\mathsf{crypt}(\mathsf{pk}(\mathsf{a}), Y, Z)$ for two fresh variables Y and Z, thus requiring that what R yields is indeed of this form. (The case where the intruder sends something that does not fit is considered separately, where $X \not\doteq \mathsf{crypt}(\mathsf{pk}(\mathsf{a}), \cdot, \cdot)$.) The constraint solving in §3.2 below then computes (a finite representation of) all solutions for R. ◁

Definition 3.2 (Semantics of FLICs)**.** *Let* $\mathcal{A}$ *be a FLIC such that* $vars(\mathcal{A}) = \emptyset$*. We say that* $\mathcal{A}$ *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.*

Example 3.2. Suppose that Alice has sent a signed message m to the intruder, and the constraint is to send some signed message to Bob. This is recorded in the following FLIC $\mathcal{A}$:

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

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. ◁

3.2 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 generated by the intruder. Thus the only place where destructors can occur are in the transactions using $\mathsf{try/catch}$. As a consequence, we can

work in the free algebra *for now* and with only destructor-free terms. We show later how the results of our decision procedure are lifted to the model where the intruder is allowed to apply destructors.

One particular case of FLICs is when the intruder only has to send intruder variables, so there are no more constraints on the messages sent and the intruder can choose any recipes they want. Such FLICs are thus always satisfiable.

Definition 3.3 (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.*

Solving constraints is done by transforming a non-simple FLIC into a simple FLIC. With the lazy intruder we can represent infinitely many models in a finite way, using substitutions instead of ground interpretations.

Definition 3.4. *Let σ be a substitution that does not contain recipe variables. We define the application of σ to a FLIC by applying σ to the messages in the FLIC: $\sigma(-l \mapsto t.\mathcal{A}) = -l \mapsto \sigma(t).\sigma(\mathcal{A})$ 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 3.5 (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 $R_1, \ldots, R_n$ be the fresh variables in r, i.e., $\{R_1, \ldots, R_n\} = rvars(r) \setminus rvars(\mathcal{A})$, where the recipe variables are taken in a fixed order (e.g., the order in which they first occur in r). Let $X_1, \ldots, X_n$ 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. $\lhd$

Table 1: 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, $t \notin \mathcal{V}_{intruder}$, and $\sigma' \neq \bot$
Composition	$(\rho, \mathcal{A}_1.+R \mapsto f(t_1, \ldots, t_n).\mathcal{A}_2, \sigma) \rightsquigarrow (\rho', \mathcal{A}_1.+R_1 \mapsto t_1.\cdots.+R_n \mapsto t_n.\mathcal{A}_2, \sigma)$ where the R_i are fresh recipe variables and $\rho' = [R \mapsto f(R_1, \ldots, R_n)]\rho$	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. 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 $\bot$ whenever no unifier exists. Slightly abusing notation, we consider a substitution $[x_1 \mapsto t_1, \ldots, x_n \mapsto t_n]$ as the formula $x_1 \doteq t_1 \wedge \cdots \wedge x_n \doteq t_n$ and $\bot$ as false. Moreover, in our fragment the privacy variables are always associated to some domains, where the formula $x \in D$ defines the set of values (public constants) that x can take. 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 König's lemma.

Definition 3.6 (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 choices of recipes and substitutions of privacy and intruder variables performed in the reduction steps so far. We assume that $dom(\rho) \cap rvars(\mathcal{A}) = \emptyset$ and $dom(\sigma) \cap vars(\mathcal{A}) = \emptyset$. The rules are defined in Table 1.*

We now describe what happens for each rule in Table 1.

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 if the message t to send is an intruder variable: the intruder is *lazy*.

Composition When the intruder has to send a composed message $f(t_1, \ldots, t_n)$, they can generate it themselves if f is public and they can generate the t_i. The

intruder thus chooses to compose the message t 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, say c, of that variable. In fact, this is a guessing attack as we let the privacy variables range over small domains of known 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 When the intruder has to send an intruder variable that they have already sent earlier, they can reuse the same recipe. While there may be several ways to generate the same message, we are lazy and just use the same recipe. The only interesting situation where we need to deal with several recipes for the same message is for the messages received by the intruder, so that they can make comparisons. This is not part of the lazy intruder rules but rather of the experiments explained in §5.

Definition 3.7 (Lazy intruder result)**.** *Let ρ be a choice of recipes, $\mathcal{A}$ be a FLIC and σ be a substitution such that $dom(\rho) \cap rvars(\mathcal{A}) = \emptyset$ and $dom(\sigma) \cap vars(\mathcal{A}) = \emptyset$. A* lazy intruder result of $(\rho, \mathcal{A}, \sigma)$ *is a triple $(\rho', \mathcal{A}', \sigma')$ such that $(\rho, \mathcal{A}, \sigma) \rightsquigarrow^* (\rho', \mathcal{A}', \sigma')$ and $\mathcal{A}'$ is simple. Let σ now be arbitrary, we define*

$$LI(\mathcal{A}, \sigma) = \{\rho' \mid (\varepsilon, \sigma(\mathcal{A}), \sigma) \rightsquigarrow^* (\rho', \mathcal{A}', \_) \text{ and } \mathcal{A}' \text{ is simple}\}$$

where ε is the identity substitution.

In our procedure, we use the lazy intruder by starting with a simple FLIC $\mathcal{A}$ and a substitution σ such that $\sigma(\mathcal{A})$ is not simple, and then taking some choice of recipes from $LI(\mathcal{A}, \sigma)$.

Definition 3.8 (Representation of choice of recipes)**.** *Let $\mathcal{A}$ be a FLIC, $\mathcal{I} \models \mathcal{A}$, ρ_0 be a ground choice of recipes and ρ be a choice of recipes for $\mathcal{A}$. 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 $R \in rvars(\mathcal{A})$:*

- *If $\rho(R) \in rvars(\mathcal{A})$, then we have $\mathcal{I}(\mathcal{A})\{\!|\, \rho_0(R) \,|\!\} = \mathcal{I}(\mathcal{A})\{\!|\, \rho_0'(R) \,|\!\}$.*
- *If $\rho(R) \in dom(\mathcal{A})$, then we have $\rho_0(R) = \rho_0'(R)$.*
- *If $\rho(R)$ is a composed recipe, then we have $\rho_0(R) = f(r_1, \ldots, r_n)$ 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 every $i \in \{1, \ldots, n\}$.*

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. Part of the completeness proofs later is to show that, despite all these liberties, the lazy intruder still returns enough recipes to

do experiments that would lead to violations of (α, β)-privacy. The lazy intruder rules are sound, complete and terminating:

Theorem 3.1. *Let $\mathcal{A}$ be a FLIC, σ be a substitution, $\mathcal{I} \models \sigma(\mathcal{A})$ and ρ_0 be a ground choice of recipes. Then ρ_0 constructs $\mathcal{I}(\sigma(\mathcal{A}))$ iff there exists $\rho \in LI(\mathcal{A}, \sigma)$ such that ρ represents ρ_0 w.r.t. $\sigma(\mathcal{A})$ and $\mathcal{I}$. Moreover, $LI(\mathcal{A}, \sigma)$ is finite.*

4 The Symbolic States

Our approach explores a symbolic transition system, i.e., transitions on symbolic states, where each symbolic state represents a set, in general infinite, 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.

In fact, 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: First, it merges all those states that differ only in the concrete γ and thus the concrete frame *concr*, i.e., where the intruder has the same uncertainty. In fact, the symbolic states does not have *concr* anymore. Second, we 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.

Finally, we have the concept of a node which is a generalization of symbolic state. A node, generally written N, contains processes (one for each $struct_i$) 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 node is a symbolic state of the symbolic transition system. This in particular ensures that transactions can only be executed atomically.

Definition 4.1 (Node, symbolic state and ground state)**.** *A* node *is a tuple $(\alpha_0, \beta_0, \mathcal{P}, Checked)$ such that:*

- *α_0 is a Σ_0-formula, the* common payload*;*
- *β_0 is a Σ_0-formula, the* intruder reasoning *about which structure is possible and about the values of 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 follow form:

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

We say that a node is a symbolic state *iff all the processes in $\mathcal{P}$ are the nil process. A* ground state *is a symbolic state (where Checked is omitted) that does not contain any recipe and intruder variables, together with a Σ_0-formula γ called the* truth*, such that γ is true for exactly one model w.r.t. $fv(\alpha_0)$ and Σ_0, and $\gamma \wedge \beta_0$ is consistent.*

As syntactic sugar, we may write $N[e \leftarrow e']$ to denote the node identical to N except that e replaced with e'. For brevity, we may omit the nil process in the possibilities of a symbolic state and write $(\phi, \mathcal{A}, \mathcal{X}, \alpha, \delta)$ for $(0, \phi, \mathcal{A}, \mathcal{X}, \alpha, \delta)$.

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 following definition of applying a recipe substitution which is only possible when all the respective $\mathcal{X}_i$ are consistent with it:

Definition 4.2 (Choice of recipes for a node)**.** *Let $N = (\_, \_, \mathcal{P}, \mathit{Checked})$ be a node and ρ be a recipe substitution. We say that ρ is a* choice of recipes for *N 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 $\sigma^{\mathcal{A}}_{\rho}(\mathcal{X})$ is consistent, i.e., ρ does not contradict the disequalities attached to any FLIC. Moreover, we define*

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

When writing $\rho(N)$ in the following, we implicitly assume that all disequalities in N are satisfiable under ρ, and that $\rho(N)$ 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.

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.

To define the semantics we also 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 $\gamma(x) = c$, then $[x \doteq \gamma(x)]^\gamma = x \doteq c$.

Definition 4.3 (Semantics of symbolic states)**.** *The concrete instantiations of a symbolic state $\mathcal{S} = (\alpha_0, \beta_0, \mathcal{P}, \_)$ are given by*

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

The ground states of $\mathcal{S}$ are given by

$$\begin{aligned}[\![\mathcal{S}]\!] = \{&(\alpha, \mathit{MMA}(\alpha, \beta_0, \rho(\mathcal{P}), \mathit{concr}), \rho(\mathcal{P}), \gamma) \mid\\ &(\alpha, \beta_0, \rho, \gamma, \mathit{concr}) \in (\!|\mathcal{S}|\!)\}\end{aligned}$$

We say that a symbolic state $\mathcal{S}$ satisfies privacy *iff for every ground state $(\alpha, \beta, \_, \_) \in [\![\mathcal{S}]\!]$, (α, β)-privacy holds.*

Remark. Given a symbolic state $\mathcal{S} = (\_, \_, \mathcal{P}, \_)$ and a ground choice of recipes ρ for $\mathcal{S}$, when defining the multi message-analysis problems in the ground states in $[\![\mathcal{S}]\!]$, we only use the first components $P_i, \phi_i, \mathit{struct}_i$ of the possibilities in $\rho(\mathcal{P})$, i.e., the components $\mathcal{X}_i, \alpha_i, \delta_i$ are ignored because they are irrelevant for the definition of *MMA* (note that the α_i have already been given as part of the payload α).

Given a symbolic state $\mathcal{S} = (\alpha_0, \beta_0, \mathcal{P}, \_)$ and a possibility with formula ϕ_i, if there are no models of $\alpha_0 \wedge \beta_0 \wedge \gamma_0 \wedge \phi_i$, then the possibility can be removed from the set $\mathcal{P}$, since it cannot correspond to any 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 4.4. *Given a substitution σ, we define the predicate isPriv as follows: $\mathit{isPriv}(\sigma)$ iff $\mathit{dom}(\sigma) \subseteq \mathcal{V}_{\mathit{privacy}}$. Moreover, we define $\mathit{isPriv}(\bot)$ to be false.*[4]

We denote with $\underline{\sigma}$ the substitution of privacy variables but not *intruder variables, i.e., $\underline{\sigma}(x) = \sigma(x)$ if $x \in \mathcal{V}_{\mathit{privacy}}$ and $\underline{\sigma}(x) = x$ otherwise.*

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

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 §5, 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 4.5. *Let $N = (\alpha_0, \beta_0, \mathcal{P}, \_)$ be a node, where $\mathcal{P}$ is a set of possibilities $\{(\_, \phi_1, \mathcal{A}_1, \_, \_, \_), \ldots, (\_, \phi_n, \mathcal{A}_n, \_, \_, \_)\}$. Let r_1 and r_2 be two recipes, and $\sigma_i = mgu(\mathcal{A}_i\{\!|\, r_1 \,|\!\} \doteq \mathcal{A}_i\{\!|\, r_2 \,|\!\})$ for $i \in \{1, \ldots, n\}$.*

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

Given two recipes r_1 and r_2, the meaning is that:

- If $r_1 \sqsubset r_2$, then the two recipes produce the same message in every FLIC.
- If $r_1 \bowtie r_2$, then 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 4.1. Suppose that a node contains two possibilities where

$$
\begin{array}{ll}
\phi_1 \equiv y \doteq \mathsf{yes} & \mathcal{A}_1 = -l \mapsto \mathsf{crypt}(\mathsf{pk}(x), \mathsf{pair}(\mathsf{yes}, \mathsf{n})) \\
\phi_2 \equiv y \doteq \mathsf{no} \wedge x \not\doteq \mathsf{a} & \mathcal{A}_2 = -l \mapsto \mathsf{crypt}(\mathsf{pk}(x), \mathsf{no})
\end{array}
$$

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. ◁

We can now define well-formed nodes, where in particular what has been checked cannot distinguish the possibilities.

Definition 4.6. *Let $N = (\alpha_0, \beta_0, \mathcal{P}, Checked)$ be a node, where $\mathcal{P}$ is a set of possibilities $\{(\_, \phi_1, \mathcal{A}_1, \_, \alpha_1, \_), \ldots, (\_, \phi_n, \mathcal{A}_n, \_, \alpha_n, \_)\}$. We say that N 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 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* $(l, r) \in \mathit{Checked}$*, we have* $l \simeq r$*.*

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

The initially empty set *Checked* keeps track of which experiments the intruder has performed (cf. §5) and well-formedness requires that these experiments indeed no longer distinguish the possibilities. We now define a set of experiments $\mathit{Pairs}(\mathcal{S})$ 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 $(l, \rho(R))$:

Definition 4.7. *Let* $\mathcal{S} = (\_, \_, \mathcal{P}, \mathit{Checked})$ *be a symbolic state. The set of pairs of recipes to compare in* $\mathcal{S}$ *is*

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

Definition 4.8 (Normal symbolic state)**.** *We say that a symbolic state* $\mathcal{S}$ *is* normal *iff* $\mathit{Pairs}(\mathcal{S}) = \emptyset$*.*

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 4.1. *Let* $\mathcal{S} = (\alpha_0, \beta_0, \mathcal{P}, \_)$ *be a normal symbolic state, where* $\mathcal{P} = \{(\phi_1, \mathcal{A}_1, \_, \_, \_), \ldots, (\phi_n, \mathcal{A}_n, \_, \_, \_)\}$*. Let* $(\_, \_, \rho_0, \_, \mathit{concr}) \in (\!|\mathcal{S}|\!)$*. Let* $\theta \models \alpha_0 \wedge \beta_0 \wedge \phi_i$ *for some* $i \in \{1, \ldots, n\}$ *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 state is consistent, iff β_0 cannot lead to violations either:

Definition 4.9 (Consistent symbolic state)**.** *We say that a 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 A.4. 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 Z3 to check consistency more efficiently. ◁

Example 4.2. Let us consider again the situation where a server sends an encrypted message containing either a pair for a positive answer or just a negative answer. Let $\mathcal{S} = (\alpha_0, \beta_0, \mathcal{P}, \emptyset)$ be the symbolic state such that:

$$\begin{aligned}
\alpha_0 &\equiv x \in \mathsf{Agent} \wedge y \in \{\mathsf{yes}, \mathsf{no}\} \\
\beta_0 &\equiv y \doteq \mathsf{yes} \vee y \doteq \mathsf{no} \\
\mathcal{P} &= \{(y \doteq \mathsf{yes}, \mathcal{A}_1, \mathsf{true}, \mathsf{true}, 0), (y \doteq \mathsf{no}, \mathcal{A}_2, \mathsf{true}, \mathsf{true}, 0)\} \\
\mathcal{A}_1 &= -l \mapsto \mathsf{crypt}(\mathsf{pk}(x), \mathsf{pair}(\mathsf{yes}, \mathsf{n})) \\
\mathcal{A}_2 &= -l \mapsto \mathsf{crypt}(\mathsf{pk}(x), \mathsf{no})
\end{aligned}$$

and 0 denotes the empty memory. We have that $\mathcal{S}$ is consistent iff for every $\mathcal{I} \models x \in \mathsf{Agent} \wedge y \in \{\mathsf{yes}, \mathsf{no}\}$, also $\mathcal{I} \models y \doteq \mathsf{yes} \vee y \doteq \mathsf{no}$. 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 consider the states where the recipes produce different messages, we would have $y \doteq \mathsf{no} \wedge x \neq \mathsf{a}$ for the second possibility and the symbolic state would then not be consistent (same payload but the new β_0 rules out the model $[x \mapsto \mathsf{a}, y \mapsto \mathsf{no}]$). ◁

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.

Theorem 4.2. *Let $\mathcal{S}$ be a normal symbolic state. Then $\mathcal{S}$ satisfies privacy iff $\mathcal{S}$ is consistent.*

Thus, to verify whether a symbolic state satisfies privacy, our strategy is to first transform it into several normal symbolic states (this is done with our intruder experiments) and then verify consistency.

5 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 5.1 (Compose-checks)**.** *The relation $\rightarrowtail$ is a binary relation on symbolic states. Let $\mathcal{S} = (\_, \beta_0, \mathcal{P}, Checked)$ be a symbolic state, where $\mathcal{P} = \{(\phi_1, \mathcal{A}_1, \mathcal{X}_1, \alpha_1, \delta_1), \ldots, (\phi_n, \mathcal{A}_n, \mathcal{X}_n, \alpha_n, \delta_n)\}$.*

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 some 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 (i.e., $l \sqsubset\!\sqsupset r$) and in the other symbolic state that the messages are not equal (i.e., $l \bowtie r$).*

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

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

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

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. Some 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.*

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

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

Example 5.1. Let $\mathcal{S} = (\alpha_0, \beta_0, \mathcal{P}, \emptyset)$ be the symbolic state from Example 4.2.

$$\begin{aligned}
\alpha_0 &\equiv x \in \mathsf{Agent} \wedge y \in \{\mathsf{yes}, \mathsf{no}\} \\
\beta_0 &\equiv y \doteq \mathsf{yes} \vee y \doteq \mathsf{no} \\
\mathcal{P} &= \{(y \doteq \mathsf{yes}, \mathcal{A}_1, \mathsf{true}, \mathsf{true}, 0), (y \doteq \mathsf{no}, \mathcal{A}_2, \mathsf{true}, \mathsf{true}, 0)\} \\
\mathcal{A}_1 &= -l \mapsto \mathsf{crypt}(\mathsf{pk}(x), \mathsf{pair}(\mathsf{yes}, \mathsf{n})) \\
\mathcal{A}_2 &= -l \mapsto \mathsf{crypt}(\mathsf{pk}(x), \mathsf{no})
\end{aligned}$$

Here we do not have any intruder variables, so the FLICs are actually already structural frames and we will not have to consider choices of recipes. We have that $\mathcal{S}$ is not normal, because for instance $(l, \mathsf{crypt}(\mathsf{pk}(\mathsf{a}), \mathsf{no})) \in \mathit{Pairs}(\mathcal{S})$.

We can perform a compose-check, in this case by applying the privacy split rule. In $\mathcal{A}_1$ we have to unify $\mathsf{crypt}(\mathsf{pk}(x), \mathsf{pair}(\mathsf{yes}, \mathsf{n}))$ 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 $\mathit{Checked} = \{(l, \mathsf{crypt}(\mathsf{pk}(\mathsf{a}), \mathsf{no}))\}$.

$$\begin{aligned}
\mathcal{S}_1 \qquad & \beta_0 \equiv (y \doteq \mathsf{yes} \vee y \doteq \mathsf{no}) \wedge (y \doteq \mathsf{yes} \Rightarrow \mathsf{false}) \\
& \qquad \wedge (y \doteq \mathsf{no} \Rightarrow x \doteq \mathsf{a}) \\
& \mathcal{P} = \{(y \doteq \mathsf{no} \wedge x \doteq \mathsf{a}, \sigma(\mathcal{A}_2), \mathsf{true}, \mathsf{true}, 0)\} \\
\mathcal{S}_2 \qquad & \beta_0 \equiv (y \doteq \mathsf{yes} \vee y \doteq \mathsf{no}) \wedge (y \doteq \mathsf{yes} \Rightarrow \mathsf{true}) \\
& \qquad \wedge (y \doteq \mathsf{no} \Rightarrow x \not\doteq \mathsf{a}) \\
& \mathcal{P} = \{(y \doteq \mathsf{yes}, \mathcal{A}_1, \mathsf{true}, \mathsf{true}, 0), \\
& \qquad (y \doteq \mathsf{no} \wedge x \not\doteq \mathsf{a}, \mathcal{A}_2, \mathsf{true}, \mathsf{true}, 0)\}
\end{aligned}$$ ◁

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 5.1 (Compose-check correctness)**.** *Let $\mathcal{S}$ be a symbolic state, $(l, r) \in \mathit{Pairs}(\mathcal{S})$ and $\{\mathcal{S}_1, \ldots, \mathcal{S}_n\}$ be the symbolic states after one rule application given the pair (l, r). Then $[\![\mathcal{S}]\!] = \biguplus_{i \in \{1,\ldots,n\}} [\![\mathcal{S}_i]\!]$, 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.*

6 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.

[2] defines a transition system for ground states: in any reachable state, any transaction can be taken and this produces a countable set of successor states. This works by inserting the transaction process in every possibility of the state, and working off steps of the processes until all are nil. We adapt this process to the symbolic level where each transaction produces only finitely many successor states. Most of the adaption is in fact straightforward now and we here give just the highlights. The details are found in Appendix A.2.

When the process receives a message, the ground system branches over the infinitely many ground recipes the intruder can employ to provide a message — we instead put a (freshly renamed) intruder variable into each FLIC. We thus do not examine what the intruder could send until it actually matters.

For a condition $\mathsf{if}\ s \doteq t$, the outcome may depend on intruder variables. We solve the respective FLIC under the mgu of s and t, yielding a finite set of recipe substitutions ρ. We split now the current symbolic state into several ones: one for each ρ and one excluding the mgu. With this, we do just partition the set of represented states into separate symbolic states. Now we look at the condition again: it may now still depend on privacy variables, and if so, we split the respective possibility into one where the condition is true and one where it is false.

For try, when honest agents apply destructors, we can turn the condition into a destructor-free unification problem, e.g., $\mathsf{try}\ X \doteq \mathsf{dscrypt}(k,t)$ becomes $t \doteq \mathsf{scrypt}(k,X,R)$, because, if the unification problem is not satisfied, the try fails. This can now be handled similar to if-then-else, and thus honest agents never introduce destructors into the messages.

When all processes have reached 0, we normalize the obtained states, i.e., perform all intruder experiments, and check consistency. If it fails, privacy is violated.

This solves the question already for the case that the destructors are unavailable for the intruder. To handle destructors conveniently, we can use the existing machinery instead of painfully extending it 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 transaction, but rather we apply them to a reached symbolic state until destructors yield no further results and the state is *analyzed*. In an analyzed state, for every recipe there exists an equivalent destructor-free recipe and we thus obtain the correctness of our decision procedure:

Theorem 6.1 (Correctness)**.** *Given a protocol specification for (α,β)-privacy, a bound on the number of transitions and an algebraic theory allowed by Definition A.2, our decision procedure is sound, complete and terminating.*

7 Tool support

We have developed a prototype tool implementing our decision procedure (available as supplementary material). 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 A.2).

In case there is a privacy violation, the tool provides a counter-model proving that the privacy goals in some reachable 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 due to 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 protocol [21] and the OSK protocol [22], 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 [23]. 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 [24].

8 Future work

One restriction of our procedure is the algebraic equations and theories we support. In the future, we would like to extend the procedure to handle more general theories. In particular, we plan to include unification modulo theories like AC to support Diffie-Hellman exponentiation. Moreover, for voting protocols it would be nice to allow arithmetic expressions in the formulas for the payload (e.g., the tally may be a sum of votes) and the information gathered by the intruder. We believe the integration with SMT solvers is a promising direction, as we can benefit from builtin support of arithmetic.

Another objective for future work is to obtain a full-fledged tool that is user-friendly and can be interactive. For instance, we can add features to explore specific traces of the transition system.

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 2021*, ser. LNCS, vol. 13075. Springer, 2021, pp. 122–142.

[4] 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.

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

[6] 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.

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

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

[9] 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.

[10] L. de Moura and N. Bjørner, "Z3: An efficient SMT solver," in *TACAS 2008*, ser. LNCS, vol. 4963. Springer, 2008, pp. 337–340.

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

[12] 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.

[13] 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.

[14] 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.

[15] 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

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

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] 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.

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

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

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

A Appendix

A.1 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.1. *Let ρ be a choice of recipes, $\mathcal{A}$ be a FLIC and σ be a substitution such that $dom(\rho) \cap rvars(\mathcal{A}) = \emptyset$ and $dom(\sigma) \cap vars(\mathcal{A}) = \emptyset$. 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.2 (Lazy intruder soundness)**.** *Let ρ be a choice of recipes, $\mathcal{A}$ be a FLIC, σ be a substitution, $\mathcal{I} \models \mathcal{A}$ and ρ_0 be a ground choice of recipes such that $dom(\rho) \cap rvars(\mathcal{A}) = \emptyset$ and $dom(\sigma) \cap vars(\mathcal{A}) = \emptyset$. 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 $\mathcal{I}$ and ρ_0' constructs $\mathcal{I}(\mathcal{A}')$. Then ρ_0 constructs $\mathcal{I}(\mathcal{A})$.*

Proof. 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 $R \notin dom(\rho')$. Since ρ_0' constructs $\mathcal{I}(\mathcal{A}')$, $\mathcal{I}(\mathcal{A}') = \mathcal{A}_1'.{+}R \mapsto t.\mathcal{A}_2'$ and $\mathcal{I}(\mathcal{A}_1')\{\!|\, \rho_0'(R) \,|\!\} = t$. Then $\mathcal{I}(\mathcal{A}_1)\{\!|\, \rho_0(R) \,|\!\} = t$.

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

- **Unification**: Let $l = \rho'(R)$. Since ρ' represents ρ_0 with ρ_0', $\rho_0(R) = \rho_0'(R) = l$. Since ρ_0' constructs $\mathcal{I}(\mathcal{A}')$, $\mathcal{I}(\mathcal{A}')\{\!|\, \rho_0'(R) \,|\!\} = t$. Then $\mathcal{I}(\mathcal{A}_1)\{\!|\, \rho_0(R) \,|\!\} = t$.

- **Composition**: Let $f(R_1, \ldots, R_n) = \rho'(R)$. Then $\mathcal{I}(\mathcal{A}') = \mathcal{A}'_1.{+}R_1 \mapsto t_1.\cdots.{+}R_n \mapsto t_n.\mathcal{A}'_2$ and $t = f(t_1, \ldots, t_n)$. Since ρ' represents ρ_0 with ρ'_0, $\rho_0(R) = f(r_1, \ldots, r_n)$ and $\rho'_0(R) = f(r'_1, \ldots, r'_n)$ such that $\mathcal{I}(\mathcal{A})\{\!|\, r_i \,|\!\} = \mathcal{I}(\mathcal{A})\{\!|\, r'_i \,|\!\}$ and $\rho'_0(R_i) = r'_i$ for $i \in \{1, \ldots, n\}$. Since ρ'_0 constructs $\mathcal{I}(\mathcal{A}')$, $\mathcal{I}(\mathcal{A}'_1)\{\!|\, \rho'_0(R_i) \,|\!\} = t_i$ for $i \in \{1, \ldots, n\}$. Then $\mathcal{I}(\mathcal{A}_1)\{\!|\, \rho_0(R) \,|\!\} = f(t_1, \ldots, t_n) = t$.

- **Guessing**: Then $t = \rho'(R)$ is a constant. Since ρ' represents ρ_0 with ρ'_0, $\rho_0(R) = \rho'_0(R) = t$. Then $\mathcal{I}(\mathcal{A})\{\!|\, \rho_0(R) \,|\!\} = t$.

- **Repetition**: Let $R' = \rho'(R)$. Then $\mathcal{I}(\mathcal{A}') = \mathcal{A}'_1.{+}R' \mapsto t.\mathcal{A}'_2$. Since ρ' represents ρ_0 with ρ'_0, $\mathcal{I}(\mathcal{A})\{\!|\, \rho_0(R) \,|\!\} = \mathcal{I}(\mathcal{A})\{\!|\, \rho'_0(R) \,|\!\}$ and $\rho'_0(R') = \rho'_0(R)$. Since ρ'_0 constructs $\mathcal{I}(\mathcal{A}')$, $\mathcal{I}(\mathcal{A}'_1)\{\!|\, \rho'_0(R') \,|\!\} = t$. Then $\mathcal{I}(\mathcal{A}_1)\{\!|\, \rho_0(R) \,|\!\} = t$. □

Lemma A.3 (Lazy intruder completeness)**.** *Let ρ be a choice of recipes, $\mathcal{A}$ be a non-simple FLIC, σ be a substitution, $\mathcal{I} \models \mathcal{A}$ and ρ_0 be a ground choice of recipes such that $dom(\rho) \cap rvars(\mathcal{A}) = \emptyset$, $dom(\sigma) \cap vars(\mathcal{A}) = \emptyset$ and ρ_0 constructs $\mathcal{I}(\mathcal{A})$. Then 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 $\mathcal{I}$ and ρ'_0 constructs $\mathcal{I}(\mathcal{A}')$.*

Proof. Let $+R \mapsto t \in \mathcal{A}$ denote the first non-simple constraint. First, we consider the case that $t \notin \mathcal{V}_{intruder}$ and $\rho_0(R)$ is a label l. Then, $\mathcal{A} = \mathcal{A}_1.{-}l \mapsto s.\mathcal{A}_2.{+}R \mapsto t.\mathcal{A}_3$ such that $\mathcal{I}(s) = \mathcal{I}(t)$. Therefore, **Unification** is applicable, producing $(\rho', \mathcal{A}', \sigma')$ such that $\rho' = [R \mapsto l]\rho$, $\mathcal{A}' = \sigma'(\mathcal{A}_1.\mathcal{A}_2)$ and $\sigma' = mgu(\sigma \wedge s \doteq t)$. Let $\rho'_0 = \rho_0$. 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 $\rho'(R) = \rho_0(R) = l$. Moreover, since ρ_0 constructs $\mathcal{I}(\mathcal{A})$, we have ρ'_0 constructs $\mathcal{I}(\mathcal{A}')$.

Next, we consider the case that $t = f(t_1, \ldots, t_n)$ and $\rho_0(R) = f(r_1, \ldots, r_n)$. Then, $\mathcal{A} = \mathcal{A}_1.{+}R \mapsto t.\mathcal{A}_2$. Therefore **Composition** is applicable, producing $(\rho', \mathcal{A}', \sigma')$ such that $\rho' = [R \mapsto f(R_1, \ldots, R_n)]\rho$, $\mathcal{A}' = \mathcal{A}_1.{+}R_1 \mapsto t_1.\cdots.{+}R_n \mapsto t_n.\mathcal{A}_2$ and $\sigma' = \sigma$, where the R_i are fresh recipe variables. Let $\rho'_0(R_i) = r_i$ for $i \in \{1, \ldots, n\}$ and $\rho'_0(R') = \rho_0(R')$ otherwise. 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 $\rho_0(R) = \rho'_0(R)$. Moreover, since ρ_0 constructs $\mathcal{I}(\mathcal{A})$, we have ρ'_0 constructs $\mathcal{I}(\mathcal{A}')$.

Next, we consider the case that $t \in \mathcal{V}_{privacy}$ and $\rho_0(R) \notin dom(\mathcal{A})$ (the case that $\rho_0(R)$ is a label is already handled above). Then, $\mathcal{A} = \mathcal{A}_1.{+}R \mapsto t.\mathcal{A}_2$, $\mathcal{I}(t) = c$ for some $c \in dom(t)$ and $\rho_0(R) = c$. Therefore, **Guessing** is applicable, producing $(\rho', \mathcal{A}', \sigma')$ such that $\rho' = [R \mapsto c]\rho$, $\mathcal{A}' = \sigma'(\mathcal{A}_1.\mathcal{A}_2)$ and $\sigma' = mgu(\sigma \wedge t \doteq c)$. Let $\rho'_0 = \rho_0$. 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 $\rho'(R) = \rho_0(R) = c$. Moreover, since ρ_0 constructs $\mathcal{I}(\mathcal{A})$, we have ρ'_0 constructs $\mathcal{I}(\mathcal{A}')$.

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 $(\rho', \mathcal{A}', \sigma')$ such that $\rho' = [R \mapsto R']\rho$, $\mathcal{A}' = \mathcal{A}_1.{+}R' \mapsto t.\mathcal{A}_2.\mathcal{A}_3$ and $\sigma' = \sigma$. Let $\rho'_0(R) =$

$\rho_0(R')$ and $\rho'_0(R'') = \rho_0(R'')$ otherwise. 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.4 (Lazy intruder correctness)**.** *Let ρ be a choice of recipes, $\mathcal{A}$ be a FLIC, σ be a substitution, $\mathcal{I} \models \mathcal{A}$ and ρ_0 be a ground choice of recipes such that $dom(\rho) \cap rvars(\mathcal{A}) = \emptyset$, $dom(\sigma) \cap vars(\mathcal{A}) = \emptyset$. 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 $\mathcal{I}$ and ρ'_0 constructs $\mathcal{I}(\mathcal{A}')$.*

Proof. By induction, using Lemmas A.2 and A.3. □

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 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.

Lemma A.5 (Lazy intruder termination)**.** *Let ρ be a choice of recipes, $\mathcal{A}$ be a FLIC and σ be a substitution such that $dom(\rho) \cap rvars(\mathcal{A}) = \emptyset$ and $dom(\sigma) \cap vars(\mathcal{A}) = \emptyset$. 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: $s = \sum_{+R \mapsto t \in \mathcal{A}} size(t)$.

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

- **Unification**: We may instantiate intruder variables so v would decrease, and if the mgu is the identity substitution 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 3.1. *Let $\mathcal{A}$ be a FLIC, σ be a substitution, $\mathcal{I} \models \sigma(\mathcal{A})$ and ρ_0 be a ground choice of recipes. Then ρ_0 constructs $\mathcal{I}(\sigma(\mathcal{A}))$ iff there exists $\rho \in LI(\mathcal{A}, \sigma)$ such that ρ represents ρ_0 w.r.t. $\sigma(\mathcal{A})$ and $\mathcal{I}$. Moreover, $LI(\mathcal{A}, \sigma)$ is finite.*

Proof. This follows directly from Lemmas A.4 and A.5. □

We now prove our results for normal symbolic states.

Lemma 4.1. *Let $\mathcal{S} = (\alpha_0, \beta_0, \mathcal{P}, \_)$ be a normal symbolic state, where $\mathcal{P} = \{(\phi_1, \mathcal{A}_1, \_, \_, \_), \ldots, (\phi_n, \mathcal{A}_n, \_, \_, \_)\}$. Let $(\_, \_, \rho_0, \_, concr) \in (\!|\mathcal{S}|\!)$. 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. 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 3.1, 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 $(l, \rho_0(\rho(R)))$ 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 $\rho(R) = l' \in dom(\mathcal{S})$. Since ρ represents ρ_0 and ρ'_0 is an instance of ρ, we have $\rho_0(R) = \rho'_0(R) = l'$. This is however impossible, because as mentioned above, $\mathcal{S}$ is normal so (l, l') is checked and cannot be a witness.

Finally we consider the case that $\rho(R)$ is a composed recipe. Then $\rho_0(R) = f(r_1, \ldots, r_n)$ 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, $(l, \rho_0(\rho(R)))$ has been checked and cannot be a witness. Thus, for (l, r) 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 $concr\{\!|\, l \,|\!\}$.

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. □

Theorem 4.2. *Let $\mathcal{S}$ be a normal symbolic state. Then $\mathcal{S}$ satisfies privacy iff $\mathcal{S}$ is consistent.*

Proof. Let $\mathcal{P} = \{(\phi_1, \mathcal{A}_1, \_, \alpha_1, \_), \ldots, (\phi_n, \mathcal{A}_n, \_, \alpha_n, \_)\}$ be the possibilities in $\mathcal{S}$. First we assume that $\mathcal{S}$ satisfies privacy and show that $\mathcal{S}$ is consistent. Let $(\alpha, \beta_0, \rho, \gamma, concr) \in (\!|\mathcal{S}|\!)$. Then $S = (\alpha, \beta, \rho(\mathcal{P}), \gamma) \in [\![\mathcal{S}]\!]$ is the corresponding ground state, where $\beta \equiv MMA(\alpha, \beta_0, \rho(\mathcal{P}), concr)$. Let $\mathcal{I} \models_{\Sigma_0} \alpha$. 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 $(\alpha, \beta_0\rho, \gamma, concr) \in (\!|\mathcal{S}|\!)$. Then $S = (\alpha, \beta, \rho(\mathcal{P}), \gamma) \in [\![\mathcal{S}]\!]$ is the corresponding ground state, where $\beta \equiv MMA(\alpha, \beta_0, \rho(\mathcal{P}), concr)$. Let $struct_i = \rho(\mathcal{A}_i)$ for $i \in \{1, \ldots, n\}$. Let $\mathcal{I} \models_{\Sigma_0} \alpha$. 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 $fv(\alpha_0)$ 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 4.1, $concr \sim \mathcal{I}'(struct_i)$ 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. □

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.6. *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. We have $\mathcal{A}' = \sigma''(\rho(\mathcal{A}))$. Since $\rho(\mathcal{A})$ and $\mathcal{A}'$ are simple, the application of ρ already substitutes the intruder variables in $dom(\sigma'')$, so we have $\mathcal{A}' = \underline{\sigma''}(\rho(\mathcal{A}))$. Then we have $\sigma''(\mathcal{A}'\{\!|\, \rho(r_1) \,|\!\}) = \underline{\sigma''}(\rho(\mathcal{A})\{\!|\, \rho(r_1) \,|\!\})$ and $\sigma''(\mathcal{A}'\{\!|\, \rho(r_2) \,|\!\}) = \underline{\sigma''}(\rho(\mathcal{A})\{\!|\, \rho(r_2) \,|\!\})$. Moreover, we have $\sigma''(\mathcal{A}'\{\!|\, \rho(r_1) \,|\!\}) = \sigma''(\mathcal{A}\{\!|\, r_1 \,|\!\})$ and $\sigma''(\mathcal{A}'\{\!|\, \rho(r_2) \,|\!\}) = \sigma''(\mathcal{A}\{\!|\, r_2 \,|\!\})$. Since $\sigma'' \models \sigma$, we have $\sigma''(\mathcal{A}\{\!|\, r_1 \,|\!\}) = \sigma''(\mathcal{A}\{\!|\, r_2 \,|\!\})$, which is the same as $\underline{\sigma''}(\rho(\mathcal{A})\{\!|\, \rho(r_1) \,|\!\}) = \underline{\sigma''}(\rho(\mathcal{A})\{\!|\, \rho(r_2) \,|\!\})$. Then $\underline{\sigma''}$ is an instance of σ' and thus $isPriv(\sigma')$. □

Theorem A.7 (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 $\mathcal{A}_1, \ldots, \mathcal{A}_n$ 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 $i \in \{1, \ldots, n\}$ (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 (except possibly by instantiation of privacy variables), 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 $i \in \{1, \ldots, n\}$ 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. If p stays the same, let

$j \in \{1, \ldots, n\}$. By Lemma A.6, 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. □

Theorem 5.1 (Compose-check correctness)**.** *Let $\mathcal{S}$ be a symbolic state, $(l, r) \in Pairs(\mathcal{S})$ and $\{\mathcal{S}_1, \ldots, \mathcal{S}_n\}$ be the symbolic states after one rule application given the pair (l, r). Then $[\![\mathcal{S}]\!] = \biguplus_{i \in \{1,\ldots,n\}} [\![\mathcal{S}_i]\!]$, 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 $\mathcal{P} = \{(\phi_1, \mathcal{A}_1, \mathcal{X}_1, \alpha_1, \delta_1), \ldots, (\phi_n, \mathcal{A}_n, \mathcal{X}_n, \alpha_n, \delta_n)\}$. First we consider the case that **Privacy split** is applicable. For every $i \in \{1, \ldots, n\}$, $isPriv(\sigma_i)$ 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 $(\alpha, \beta_0, \rho, \gamma, concr) \in (\!|\mathcal{S}|\!)$ for some $i \in \{1, \ldots, n\}$ and $struct_j = \rho(\mathcal{A}_j)$ for $j \in \{1, \ldots, n\}$. Then $(\alpha, \beta, \rho(\mathcal{P}), \gamma) \in [\![\mathcal{S}]\!]$ is the corresponding ground state, where $\beta \equiv MMA(\alpha, \beta_0, \rho(\mathcal{P}), concr)$.

- If $isPriv(\sigma_i)$ and $\gamma \models \sigma_i$: Then $(\alpha, \beta_0', \rho, \gamma, concr) \in (\!|\mathcal{S}_1|\!)$ and $(\alpha, \beta', \rho(\mathcal{P}'), \gamma) \in [\![\mathcal{S}_1]\!]$ is the corresponding ground state, where $\beta' \equiv MMA(\alpha, \beta_0', \rho(\mathcal{P}'), concr)$ and

$$\beta_0' \equiv \beta_0 \wedge \bigwedge_{j=1}^{n} \left(\phi_j \Rightarrow \begin{cases} \sigma_j & \text{if } isPriv(\sigma_j) \\ \mathsf{false} & \text{otherwise} \end{cases} \right)$$

$$\mathcal{P}' = \{(\phi_j \wedge \sigma_j, \sigma_j(\mathcal{A}_j), \sigma_j(\mathcal{X}_j), \alpha_j, \sigma_j(\delta_j)) \mid j \in \{1, \ldots, n\}, isPriv(\sigma_j)\}$$

  We want to show that $\beta \equiv \beta'$. Let $\mathcal{I} \models_\Sigma \beta$. There exists $j \in \{1, \ldots, n\}$ 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 $\mathcal{I} \models_\Sigma \phi_j \wedge \sigma_j \wedge concr \sim \sigma_j(struct_j))$, 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: Then $(\alpha, \beta_0', \rho, \gamma, concr) \in (\!|\mathcal{S}_2|\!)$ and $(\alpha, \beta', \rho(\mathcal{P}'), \gamma) \in [\![\mathcal{S}_2]\!]$ is

the corresponding ground state, where $\beta' \equiv MMA(\alpha, \beta'_0, \rho(\mathcal{P}'), concr)$ and

$$\beta'_0 \equiv \beta_0 \wedge \bigwedge_{j=1}^{n} \left(\phi_j \Rightarrow \begin{cases} \neg\sigma_j & \text{if } isPriv(\sigma_j) \\ \mathsf{true} & \text{otherwise} \end{cases} \right)$$

$$\begin{aligned} \mathcal{P}' = \{&(\phi_j \wedge \neg\sigma_j, \mathcal{A}_j, \mathcal{X}_j, \alpha_j, \delta_j) \mid j \in \{1, \ldots, n\}, \\ &isPriv(\sigma_j)\} \\ \cup \{&(\phi_j, \mathcal{A}_j, \mathcal{X}_j, \alpha_j, \delta_j) \mid j \in \{1, \ldots, n\}, \\ &\text{not } isPriv(\sigma_j)\} \end{aligned}$$

We want to show that $\beta \equiv \beta'$. Let $\mathcal{I} \models_\Sigma \beta$. There exists $j \in \{1, \ldots, n\}$ 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 $i \in \{1, \ldots, n\}$ such that not $isPriv(\sigma_i)$ and $LI(\sigma_i, \sigma_i) = \{\rho_1, \ldots, \rho_k\}$, 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 $j \in \{1, \ldots, k\}$, and $\mathcal{S}_0$ 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 an instance of ρ_j (note that the ρ_j are mutually exclusive); otherwise $S \in [\![\mathcal{S}_0]\!]$. Conversely, $\biguplus_{j \in \{0,\ldots,k\}} [\![\mathcal{S}_j]\!] \subseteq [\![\mathcal{S}]\!]$.

The termination follows from Theorem A.7. □

A.2 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 lumping of several ground states into symbolic states 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 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.

We now sketch the proof 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, present our version of the rules and argue why our representation is correct. We have made some adaptations for the possibilities, in particular:

- On the ground level, there is a sequence δ, global to the ground state, of conditional updates of the form $\mathsf{cell}(s) := t$ if ϕ. On the symbolic level, we have sequences δ_i without conditions, attached to each possibility.
- On the symbolic level, there is a partial payload α_i attached to each possibility, because which possibility is marked is only defined in the semantics of symbolic states so the full payload cannot be fixed before.

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 node. This is done with normalization and evaluation rules that work out the steps of the processes. Once all the processes are nil, we have reached a symbolic state. The rules thus generate a transition system representing all the reachable states of the protocol, after a number of transactions.

Definition A.1 (Initial node of a transaction)**.** *Let $\mathcal{S} = (\_, \_, \mathcal{P}, \_)$ be a 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 node of P w.r.t. $\mathcal{S}$ and σ *is*

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

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

There are two kinds of rules: the normalization rules and the evaluation rules. We define the rules by focusing on the set of possibilities $\mathcal{P}$, which contains the transaction to execute. The rules update the current node $N = (\alpha_0, \beta_0, \mathcal{P}, \_)$ by updating $\mathcal{P}$, and the rest of the node is not changed unless explicitly stated. We use the symbol $\uplus$ to denote the disjoint union of sets.

A.2.1 Normalization Rules

The normalization rules are concerned with cell reads, cell writes, destructor applications, conditional statements and releases. For simplicity, we ignore the redundancy rules defined in [2] at this point: the redundant possibilities do not change the semantics of symbolic states, and we already said that they are eliminated in the procedure.

Cell read On the ground level, the memory δ contains the sequence $\mathsf{cell}(s_1) := t_1 \text{ if } \phi_1 . \cdots . \mathsf{cell}(s_k) := t_k \text{ if } \phi_k$ for the given cell, and the initial value is given with ground context $C[\cdot]$.

$$\begin{aligned}
&\{(x := \mathsf{cell}(s).P, \phi, \mathit{struct})\} \uplus \mathcal{P} \\
&\longrightarrow \{(\mathsf{if}\ s \doteq s_1 \wedge \phi_1\ \mathsf{then}\ \mathsf{let}\ x = t_1.P\ \mathsf{else} \\
&\qquad \cdots \\
&\qquad \mathsf{if}\ s \doteq s_k \wedge \phi_k\ \mathsf{then}\ \mathsf{let}\ x = t_k.P\ \mathsf{else} \\
&\qquad \mathsf{let}\ x = C[s].P, \phi, \mathit{struct})\} \cup \mathcal{P}
\end{aligned}$$

On the symbolic level, 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 $C[\cdot]$.

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

We do not have the conditions in the memory updates anymore. The variable X is assigned a value from a memory cell, and the cell read step is transformed into conditional statements so that the correct memory cell is used in the assignment to X. Using the δ_i instead of δ does not change the semantics, because the ϕ_i of the possibilities are mutually exclusive, so a memory update done in one possibility is not relevant to other possibilities.

Cell write On the ground level, the memory δ becomes $\mathsf{cell}(s) := t \text{ if } \phi.\delta$, where the condition of the memory update indicates which possibility is writing in memory. Note that it is important to *prepend* so that when we do a cell read, the most recent state is used first in a conditional, effectively overwriting the previous memory state.

$$\{(\mathsf{cell}(s) := t.P, \phi, \mathit{struct})\} \uplus \mathcal{P} \longrightarrow \{(P, \phi, \mathit{struct})\} \cup \mathcal{P}$$

On the symbolic level, the rule is the same but we update the memory δ attached to the possibility.

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

Again, we do not have the conditions ϕ_i in the memory updates but this does not change the semantics.

Destructor application On the ground level, try/catch is syntactic sugar wrapping the application of a verifier around the binding of the result from the destructor application. On the symbolic level, we have the destructor applications with try/catch and we handle destructor applications in a specific way because of our assumptions that this is the only place in a specification where destructors are allowed. We explain the correctness of that model in Appendix A.3.

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 algebraic equation $d(k, c(k', X_1, \ldots, X_n)) \approx X_i$ (for some $i \in \{1, \ldots, n\}$), and assuming the variables in k, k' and the X_j have been renamed with fresh intruder variables. To resolve the destructor application $X \doteq d(t_1, t_2)$, we compute the unifier $\sigma = mgu(t_2 \doteq c(k', X_1, \ldots, X_n) \wedge t_1 \doteq k \wedge X \doteq X_i)$. The meaning is that t_2 must be of the form $c(k', X_1, \ldots, X_n)$ 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 for a transparent function, then there are no terms t_1 and k but all 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. There are three cases.

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

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

- $\sigma \neq \bot$ and $\sigma(\mathcal{A})$ is simple: Then σ does not really introduce constraints, it can only substitute privacy variables and rename intruder variables with fresh names (otherwise there would be non-simple constraints). Thus we split into two possibilities, where we use $\underline{\sigma}$ so that the formulas like ϕ only contain privacy variables and the renamed intruder variables do not occur (but they are substituted in the process).

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

- $\sigma \neq \bot$ and $\sigma(\mathcal{A})$ is not simple: Then we use the lazy intruder rules to solve the constraints. There is one transition for every $\rho \in LI(\mathcal{A}, \sigma)$ returned by the lazy intruder rules, where the try/catch remains but ρ is applied to the whole node. When this possibility is normalized again, the destructor application will lead to a unifier making the FLIC simple, because the intruder variables are substituted when applying ρ. There may still be

intruder variables in the unifier but then they are only renamed and not really introducing constraints. The possibility will be split on privacy variables according to the previous case.

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

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

where $\bar{Y} = \mathit{ivars}(\sigma) \setminus \mathit{ivars}(\mathcal{A})$, i.e., the intruder variables that are not occurring in the FLIC are universally quantified when excluding the unifier.

Example A.1. Suppose that there is a possibility with the process $\mathsf{try}\ Y \doteq \mathsf{dcrypt}(\mathsf{inv}(\mathsf{pk}(\mathsf{a})), X)\ \mathsf{in}\ P_1\ \mathsf{catch}\ P_2$ and the FLIC $\mathcal{A} = -l \mapsto \mathsf{pk}(\mathsf{a}).+R \mapsto X$. Then we use the equation $\mathsf{dcrypt}(\mathsf{inv}(X_1), \mathsf{crypt}(X_1, X_2, X_3)) \approx X_2$ and we get $\sigma = [X \mapsto \mathsf{crypt}(\mathsf{pk}(\mathsf{a}), Y, X_3), X_1 \mapsto \mathsf{pk}(\mathsf{a}), X_2 \mapsto Y]$. We have for instance $\rho = [R \mapsto \mathsf{crypt}(l, R_2, R_3)] \in LI(\mathcal{A}, \sigma)$, so the intruder considers one node where they have chosen ρ and the process starts with $\mathsf{try}\ Y \doteq \mathsf{dcrypt}(\mathsf{inv}(\mathsf{pk}(\mathsf{a}), \mathsf{crypt}(\mathsf{pk}(\mathsf{a}), X_2', X_3'))$, which will lead to a unifier keeping the FLIC simple at the next normalization. Moreover, there is a node where we remember the disequality $\forall X_1, X_2, X_3, Y.\ X \not\doteq \mathsf{crypt}(\mathsf{pk}(\mathsf{a}), Y, X_3) \vee X_1 \not\doteq \mathsf{pk}(\mathsf{a}) \vee X_2 \not\doteq Y$, which is actually equivalent to $\forall X_3, Y.\ X \not\doteq \mathsf{crypt}(\mathsf{pk}(\mathsf{a}), Y, X_3)$. ◁

Conditional statement On the ground level, one possibility is split into two. 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.

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

On the symbolic level, there are two base cases: when the condition is a relation $R(t_1, \ldots, t_n)$ and when the condition is an equality $s \doteq t$. For an arbitrary formula in our grammar (defined in §2), 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.

$$\{(\mathsf{if}\ R(t_1, \ldots, t_n)\ \mathsf{then}\ P_1\ \mathsf{else}\ P_2, \phi, \mathcal{A}, \mathcal{X}, \alpha, \delta)\} \uplus \mathcal{P}$$
$$\Longrightarrow \{(P_1, \phi \wedge R(t_1, \ldots, t_n), \mathcal{A}, \mathcal{X}, \alpha, \delta),$$
$$(P_2, \phi \wedge \neg R(t_1, \ldots, t_n), \mathcal{A}, \mathcal{X}, \alpha, \delta)\} \cup \mathcal{P}$$

  Recall that all the t_i must be terms using only symbols from Σ_0 and $\mathit{fv}(\alpha_0)$ a that point, otherwise we consider it a specification error.

- If the condition is an equality: We first compute the unifier $\sigma = \mathit{mgu}(s \doteq t)$ 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 node, where $\mathcal{A} = +R_1 \mapsto X.+R_2 \mapsto Y$. Then $\sigma = [X \mapsto Y]$ and the only intruder result is $(\rho, +R_1 \mapsto Y, \sigma')$ where $\rho = [R_2 \mapsto R_1]$ and $\sigma' = [X \mapsto Y]$. The intruder considers one node where they have chosen ρ (all the FLICs and the rest of the node is updated accordingly), and one node 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. In the possibility where the condition holds, we may instantiate privacy variables, but that is just a consequence of the intruder's knowledge so it does not change the semantics. 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.

Note that there are several transitions on the symbolic level, while on the ground level there is just one transition. However, 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.

Release One 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.

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

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.

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

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.

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 in γ (with $\mathsf{mode} = \diamond$) 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.

A.2.2 Evaluation Rules

The evaluation rules are concerned with non-deterministic choices of variables, receiving and sending messages and terminating processes. The evaluation rules can only be applied if no normalization rule is applicable.

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 that the variable can take, where the truth formula γ is updated accordingly.

$$\begin{aligned}&\{(\mathsf{mode}\ x \in D.P_1, \phi_1, \mathit{struct}_1), \ldots,\\ &\ (\mathsf{mode}\ x \in D.P_n, \phi_n, \mathit{struct}_n)\}\\ &\longrightarrow \{(P_1, \phi_1, \mathit{struct}_1), \ldots, (P_n, \phi_n, \mathit{struct}_n)\}\end{aligned}$$

for every $c \in D$, and $\beta \leftarrow \beta \wedge x \in D$, $\gamma \leftarrow \gamma \wedge x \doteq c$ and if $\mathsf{mode} = \star$ also $\alpha \leftarrow \alpha \wedge x \in D$

On the symbolic level, we have a single transition and we only update α_0 or β_0 (which are used later to define the full α and β) with the formula $x \in D$.

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

where the rest of the node is updated as follows:

$$\begin{aligned}&\alpha_0 \leftarrow \alpha_0 \wedge x \in D && \text{if } \mathsf{mode} = \star\\ &\beta_0 \leftarrow \beta_0 \wedge x \in D && \text{if } \mathsf{mode} = \diamond\end{aligned}$$

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).

Receive All possibilities have this receive step at the same time. 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.

$$\begin{aligned}&\{\underline{(\mathsf{rcv}(x).P_1, \phi_1, \mathit{struct}_1)}, \ldots, (\mathsf{rcv}(x).P_n, \phi_n, \mathit{struct}_n)\}\\ &\longrightarrow \{\underline{(P_1[x \mapsto \mathit{struct}_1\{\!|\, r \,|\!}], \phi_1, \mathit{struct}_1)}, \ldots,\\ &\qquad (P_n[x \mapsto \mathit{struct}_n\{\!|\, r \,|\!}], \phi_n, \mathit{struct}_n)\}\end{aligned}$$

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.

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

where R is a fresh recipe variable.

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).

Send On the ground level, the intruder observes that a message is sent, so they can rule out all possibilities that have terminated: the formula β is updated to include the concrete message observed and to rule out possibilities not sending.

$$\begin{aligned}&\{\underline{(\mathsf{snd}(t_1).P_1, \phi_1, \mathit{struct}_1)}, \ldots, (\mathsf{snd}(t_k).P_k, \phi_k, \mathit{struct}_k)\} \uplus \mathcal{P}\\ &\longrightarrow \{\underline{(P_1, \phi_1, \mathit{struct}_1.l \mapsto t_1)}, \ldots, (P_k, \phi_k, \mathit{struct}_k.l \mapsto t_k)\}\end{aligned}$$

where every process in $\mathcal{P}$ is the nil process and

$$\begin{aligned}\beta \leftarrow &\beta \wedge \bigvee_{i=1}^{k} \phi_i \wedge \mathit{concr}[l] \doteq \gamma(t_1)\\ &\wedge \exists i \in \{1, \ldots, k\}.\ \bigvee_{j=1}^{k} i \doteq j \wedge \mathit{struct}[l] \doteq t_j \wedge \phi_j\end{aligned}$$

On the symbolic level, we update β_0 to include the disjunction of which possibilities send a message, add a mapping in the FLICs and remove the possibilities that are not sending.

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

where l is a fresh label, every process in $\mathcal{P}$ is the nil process and the rest of the node is updated as follows: $\beta_0 \leftarrow \beta_0 \wedge \bigvee_{i=1}^{k} \phi_i$.

The formula for the concrete message and the existence of a corresponding structure is included in our definition of multi message-analysis problems, in the semantics of the symbolic states.

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.

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

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

On the symbolic level, we update β_0 to include the disjunction of which possibilities terminate.

$$\{(0, \phi_1, \mathcal{A}_1, \mathcal{X}_1, \alpha_1, \delta_1), \ldots, (0, \phi_k, \mathcal{A}_k, \mathcal{X}_k, \alpha_k, \delta_k)\} \uplus \mathcal{P}$$
$$\Longrightarrow \{(0, \phi_1, \mathcal{A}_1, \mathcal{X}_1, \alpha_1, \delta_1), \ldots, (0, \phi_k, \mathcal{A}_k, \mathcal{X}_k, \alpha_k, \delta_k)\}$$

where every process in $\mathcal{P}$ starts with a send step and the rest of the node 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 the normalization rules or require that all possibilities start with the same step (non-deterministic choices and receives), the processes that do not send are actually terminating (nil process). Thus, on the symbolic level both rules are in general applicable at the same time.

A.2.3 Correctness

Our normalization and evaluation rules are relations on nodes. They correspond to internal transitions for the symbolic execution of transactions, which is distinct from the overall transition system with symbolic states, where the transactions are atomic. We define $\underset{P}{\Longrightarrow}$ to be the relation between an initial node of a transaction P and a symbolic state, using the relation $\Longrightarrow$ of normalization and evaluation rules until all processes in $\mathcal{P}$ have terminated and a symbolic state is reached. Then we can define $\mathcal{S}_{\underset{P}{\Longrightarrow}}$ the set of symbolic 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 $S_{\underset{P}{\longrightarrow}}$ to be the set of ground states after the execution of one atomic transaction starting from a ground state S (transitions from [2]):

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

Our rules are correct w.r.t. the transitions that can happen on the ground level.

Proposition A.8 (Reachability correctness)**.** *Let $\mathcal{S}$ be a 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 $[\![\mathcal{S}_{\underset{P}{\Longrightarrow}}]\!] = [\![\mathcal{S}]\!]_{\underset{P}{\longrightarrow}}$.

A.3 Lifting to the Full Algebraic Model

So far, we have considered a model where the intruder does not have access to the destructors, which implies the considerable simplification that we can operate entirely within the free algebra (except for the statements with try/catch, where honest agents apply destructors).

A.3.1 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 [23] (found also in our examples `basicHash.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 A.2. *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: a constructor* c *of arity* n *and destructors* $d_1, \ldots, d_n$ *with the property* $d_i(c(x_1, \ldots, x_n)) \to x_i$ *for* $i \in \{1, \ldots, n\}$. *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 *can occur in at most one of the three categories (decryption, transparency, and private extractors).*

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

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

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* $k \approx f(k')$ *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 for a second we would omit 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 $\mathsf{dcrypt}(l_1, \mathsf{crypt}(l_2, r, r)) \approx \mathtt{ff}$ 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 in the proof below to show that the intruder cannot learn anything new from decrypting terms that they have encrypted themselves. ◁

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[5] in t and replacing by $\mathtt{ff}$ if no rewrite rule is applicable, and repeating this until all destructors are eliminated.

[5]Here "inner-most" means that no proper subterm has a destructor. This reduction strategy may be called "call by value" and this is necessary as the following example shows:

A.3.2 Destructor Oracles

So far, the intruder can only use constructors and labels in recipes, but not destructors, just as if they were not public functions. It is obvious that the entire machinery is an order of magnitude more complicated if one adds destructors, namely for sending messages to honest agents, analyzing their answers and conducting experiments. Given the quite involved development up to this point, we are sure that the reader is relieved to read that we go for a much simpler path. 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 following transaction

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

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:

$$\begin{aligned}&\mathsf{rcv}(Y).\mathsf{try}\ Z_1 \doteq d_1(Y)\ \mathsf{in}\ \ \ldots\ \ \mathsf{try}\ Z_n \doteq d_n(Y)\ \mathsf{in}\\ &\mathsf{snd}(Z_1).\cdots.\mathsf{snd}(Z_n)\end{aligned}$$

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:

$\mathsf{dscrypt}(\mathsf{dscrypt}(k, c), \mathsf{scrypt}(\mathsf{dscrypt}(k, d), s, r)) \approx s$ and it is not equivalent to $\mathtt{ff}$ (which an "outer-most" or "call by name" strategy would produce), because $\mathsf{scrypt}(\mathsf{dscrypt}(k, d), s, r) \approx \mathsf{scrypt}(\mathtt{ff}, s, r) \approx \mathsf{scrypt}(\mathsf{scrypt}(k, c), s, r)$ and thus the outer-most destructor must result in s according to Definition A.2. Also observe that at most one rewrite rule can be applied to an inner-most destructor subterm of t since E is convergent.

Definition A.3 (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 for $\star$ 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.

Definition A.4 (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 $\mathcal{S}'_1, \ldots, \mathcal{S}'_k$ (there is at least one, so $k \geq 1$) 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 it is currently not decipherable. If it was already marked $\checkmark$, 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 $c(k', t_1, \ldots, t_n)$ and 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 $\checkmark$, 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). 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.*

Lemma A.9. *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 FLICs 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 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 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 transaction, when we evaluate the try-statements, we split each possibility into two further cases (the one where the try succeeds, and where it fails), but from the send-statements only one version 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 $\checkmark$ (and similar 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 $\checkmark$ 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 $\checkmark$, we do not necessarily have a reduction, but new l'-term 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 with König'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. □

In analyzed states, the intruder does not need any destructors anymore:

Lemma A.10. *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 $\mathcal{A}\{\!|\, r \,|\!\} \approx \mathcal{A}\{\!|\, r' \,|\!\}$ in every FLIC $\mathcal{A}$ of S.*

Proof. Note also that this proof works on a ground state S which does not contain intruder variables anymore (but still privacy variables). Thus, the FLICs now 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 FLIC, then the result of the destructor must be the i-th subterm (for some $i \in \{1, \ldots, n\}$) of r_2 in every FLIC, i.e., $\mathcal{A}\{\!|\, r_d \,|\!\} = \mathcal{A}\{\!|\, r'_i \,|\!\}$ for every FLIC $\mathcal{A}$, and we can thus replace r_d by r'_i.

(b) If r_d yields $\mathtt{ff}$ in all FLICs, i.e. $\mathcal{A}\{\!|\, r_d \,|\!\} = \mathtt{ff}$ in every FLIC $\mathcal{A}$, we can just replace r_d by $\mathtt{ff}$.

(c) If r_d yields $\mathtt{ff}$ in some FLIC $\mathcal{A}_1$ and does not yield $\mathtt{ff}$ in another FLIC $\mathcal{A}_2$, it means that comparing r_d with $\mathtt{ff}$ is an intruder experiment that distinguishes the FLICs. We show that this contradicts the fact that $\mathcal{S}$ is analyzed and normal. The only reason that $\mathcal{A}_1$ and $\mathcal{A}_2$ give different results is that the encryption and decryption key do not match in $\mathcal{A}_1$ but do match in $\mathcal{A}_2$. Recall that in a decryption rule with decryption key k and encryption key k', we require that either $k \approx f(k')$ or $k' \approx f(k)$ for some public function f. We only prove the case $k \approx f(k')$, the other case is analogous. In $\mathcal{A}_2$, 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 FLICs 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 FLIC is marked $\checkmark$, 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 FLICs or failed in all FLICs. 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 FLICs. This would contradict that $\mathcal{S}$ is normal because r_1 and l_2 have no destructors. Thus, $\mathcal{A}\{\!|\, r_1 \,|\!\} = \mathcal{A}\{\!|\, l_2 \,|\!\}$ in every FLIC $\mathcal{A}$, and thus $\mathcal{A}\{\!|\, r_d \,|\!\} = \mathcal{A}\{\!|\, l_1 \,|\!\}$ and we can replace r_d by l_1.

(b) Case l maps in all FLICs to terms that have been marked $\checkmark$ 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 FLIC $\mathcal{A}$,

l maps to a term $c(s_1, \ldots, s_m)$ where c was composed by the intruder, i.e., there are destructor-free recipes $r'_1, \ldots, r'_m$ that produce s_i in $\mathcal{A}$, thus $\mathcal{A}\{\!|\, l \,|\!\} = \mathcal{A}\{\!|\, c(r'_1, \ldots, r'_m) \,|\!\}$. As these recipes are all destructor-free, this is an experiment that must work in all FLICs (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.

□

Note that we defined that a state is normal with respect to intruder experiments performed with destructor-free recipes. We now consider intruder experiments that may contain destructors:

Lemma A.11. *Let $\mathcal{S}$ be an analyzed state and normal w.r.t. destructor-free recipes. 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 A.10, 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. □

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

Theorem 6.1 (Correctness)**.** *Given a protocol specification for (α, β)-privacy, a bound on the number of transitions and an algebraic theory allowed by Definition A.2, our decision procedure is sound, complete and terminating.*

Proof. This is essentially lifting Proposition A.8 to the case where the intruder has access to destructors (except private destructors, 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 regards states 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 A.11, it is also normal w.r.t. arbitrary recipes. In the model where destructors are private, by Proposition A.8, 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, of course.) 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 A.10. Thus, $[\![\mathcal{S}]\!]_{\xrightarrow[P]{}}$ 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 $\mathcal{S}' \in \mathcal{S}_{\underset{P}{\Longrightarrow}}$, we obtain finitely many analyzed states $\mathbb{S}$ such that $[\![\mathcal{S}_{\underset{P}{\Longrightarrow}}]\!] = \bigcup_{\mathcal{S}'' \in \mathbb{S}} [\![\mathcal{S}'']\!]$ by Lemma A.9. By Lemma A.11 these symbolic states in $\mathbb{S}$ 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. □

A.4 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:

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