A Compositionality Result for Alpha-Beta Privacy

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

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

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

***Abstract*—The security goals of a protocol are often studied with the protocol running in isolation. However, in reality, several protocols are running in parallel or are otherwise interacting with each other. It is thus desirable to obtain results for composing protocols securely. While there are many results for goals like secrecy and authentication, it is harder to achieve compositionality for privacy properties. We consider protocols in a symbolic typed model where privacy goals are expressed with alpha-beta privacy and can be verified as a reachability property. We support a large class of stateful protocols and algebraic theories, and we identify *composable protocols*, for which we show how to derive the security of a composed protocol from the security of its components.**

***Index Terms*—Privacy, Formal Methods, Security Protocols, Protocol Composition**

I. Introduction

The model of Dolev-Yao for security protocols, i.e., an intruder who can control the network and act as a protocol participant but who cannot break cryptography, has proved to be a very effective basis for automated verification approaches, e.g., ProVerif [1], Tamarin [2], or CPSA [3].

Using a communication medium like the Internet where a variety of protocols run in parallel, sharing a public-key infrastructure, begs the question of protocol composition: whether any attacks can arise even if each protocol in isolation is secure. It certainly does not make sense to verify the composition directly, not the least because any new protocol and any protocol update would require one to start verification from scratch. There are compositionality results that show: this composition is secure, if the messages of the component protocols are sufficiently disjoint [4], which can be achieved, e.g., using tags [5], [6], [7].

Beyond protocols that only share network and public-key infrastructure, there are some compositionality results that allow for some interaction between the component protocols, e.g., one protocol P_1 negotiates a key that is then used by another protocol P_2. There needs to be an interface between the components; for instance, P_1 gives guarantees of secrecy, authentication, and freshness of the negotiated keys, and P_2 may guarantee that it does not leak those keys to another party. This also allows one to verify that components can be replaced by ones that offer the same interface to the other protocol: for P_2 it does not matter how exactly P_1 achieves its goals and vice-versa.

For privacy-type goals like unlinkability (an outsider cannot tell if two protocol sessions were executed by the same or different participants, let alone know their identity), the standard Dolev-Yao model is not sufficient because it cannot directly model weak secrets like the name of a participant: the intruder may know the name of all participants, and the secret is rather which user has performed a particular action. It is standard to use equivalence notions here: we consider a pair of worlds and verify that the intruder cannot tell which world is the case. For instance, for unlinkability we have one world where some agent runs several protocol sessions and another world where different agents run one session each.

For normal trace properties, the compositionality argument consists in demonstrating that any attack trace against the composed protocol can be split into traces for the component protocols in isolation such that the goals of at least one component are violated. This reasoning is substantially more complicated when considering an equivalence notion of traces instead. Thus, there exist but a few compositionality results for privacy-type goals and they have rather restrictive assumptions; this is discussed in detail in the final section.

To overcome these difficulties, we employ for our compositionality result an alternative model of privacy: (α, β)-privacy. Here, a protocol is modeled as a set of reachable states where each state represents just one reality and contains a logical characterization of what the intruder knows about the state of the world. This allows for turning privacy into a reachability problem without losing the expressive power of equivalence-based notions. We can thus resort to the standard approach for compositional reasoning, i.e., showing that an attack trace against the composed protocol can be mapped into traces for the component protocols, and in fact obtain a general result:

- The component protocols do not need to be disjoint: they can have a set of shared secrets, and these secrets can also be declassified in the course of the protocol.
- A component protocol can be used as a subprotocol called by another protocol, e.g., to query a key server and proceed with the result from the key server.
- The component protocols can include long-term state (that lasts beyond a single session) and this long-term state can also be shared.

Moreover, the logical approach of (α, β)-privacy is fully supported for the specification of the goals; for instance, for unlinkability one simply specifies that the intruder is only allowed to know that each actor x_i is in the set Agent; it is thus an attack if the intruder can deduce $x_1 \doteq x_2$ for example. (This is more complicated when an intruder can be a participant, as discussed in the running example.)

In this paper, we restrict the cryptographic operators to constructor-destructor theories. Moreover we assume a typed model, where the intruder is only sending messages of the correct types. This makes the reasoning about attack traces much easier and typing results for similar protocol models show that this is without loss of attacks for well-tagged protocols [7], [8], [9], [10], [11], [12].

Our first contribution is to extend (α, β)-privacy to generalize the class of protocols that can be modeled, with new constructs useful for protocol composition. Our second and main contribution is the compositionality result itself: we identify requirements for *composable protocols*, which allow for the modular verification of privacy goals with an "assume-guarantee" approach, i.e., we verify one component with the interface of the other component.

The paper is organized as follows. In §II, we introduce (α, β)-privacy and protocol composition through an example based on Needham-Schroeder. In §III, we give the grammar of protocol specifications and define the supported algebraic theories. In §IV, we define the semantics of protocols as a symbolic execution performed by the intruder, giving rise to a state transition system. In §V, we define the class of *composable* protocols that can be composed securely. In §VI, we present our results and finally we conclude in §VII with the discussion of related work. All the proofs and intermediate results are provided in appendix, together with a larger example protocol.

II. Running Example

As a simple example we play a bit with the famous Needham-Schroeder-Lowe public-key protocol (NSL) [13], [14]. Privacy was not a concern in the original protocol, but we can easily add this as a requirement: when A contacts B, nobody else but A and B should learn who is communicating here. In fact, A and B may not know each other in advance and need to first get each other's public key from a key server who thus also learns their identities. In the original protocol, the key-exchange with the server is not encrypted, so that everybody can see that A is looking up B's public key and vice-versa, so we are going to encrypt this exchange. As an example for compositionality we actually split the protocol into two components: P_1 the three-way handshake between A and B and P_2 the lookup protocol for the key server, so we can verify handshake and key server protocol separately.

We give the specification of P_1 the handshake protocol in Fig. 1 (on the left); the notation is adapted from (α, β)-privacy [15] with some extensions for composition and procedure calls. (We give in the next sections the precise syntax and semantics.) We use the color blue to highlight the parts in a protocol that the other protocol must know about.

In the first component, we have two roles, Initiator and Responder, which each consist of a number of *transactions*. A transaction is a sequence of steps that will be executed atomically (without interleaving by other transactions). The transactions are separated by semicolons. This is relevant for privacy as we assume that the intruder knows the protocol and can observe when an agent reaches the end of a transaction in our semantics, because typically the agent is inactive and waiting for another input message.

The first transaction in the Initiator role is that we choose two *privacy variables* x_A and x_B from the sets Honest and Agent. These can be any finite sets (that we do not specify here) where $\mathsf{Honest} \subseteq \mathsf{Agent}$ and the intruder can control a number of dishonest agents in $\mathsf{Agent} \setminus \mathsf{Honest}$. The symbol $\star$ means here that the intruder is allowed to learn the domains of x_A and x_B. More precisely every state will contain a formula α, also called the *payload*, which contains all the information that was deliberately released to the intruder; in this case the conjunct $x_A \in \mathsf{Honest} \wedge x_B \in \mathsf{Agent}$. Note that the variables will be freshly renamed for every session, i.e., instantiation of the role. One may wonder why $x_A \in \mathsf{Honest}$ rather than Agent. This is because x_A would be the agent executing this role; since a dishonest agent may not follow the protocol, their behavior is covered by the intruder rules, and thus it is convenient to restrict the execution here to the honest x_A (while x_B may well be a dishonest agent). The first thing that x_A really does is to look up the public key of x_B. Whatever happens here will be the job of the other protocol P_2, so we just consider here that this results in a key PKB.

In its second transaction, x_A generates nonces N_A and R and sends out the first real message of the protocol: $\mathsf{crypt}(PKB, \mathsf{f}_1(N_A, x_A), R)$ where crypt stands for public key encryption, R is a randomization value to avoid deterministic encryption, and f_1 is abstracting the data format of the first message of the protocol. This is a slight generalization of the usual tagging schemes where the messages of different meaning contain a tag to tell them apart; rather formats like f_1 represent an arbitrary way to implement the message formats (e.g., XML or JSON) and we just assume that they are unambiguous and pairwise disjoint.

Note that in this way, the protocol would be violating privacy: if x_B is honest, then the intruder would learn this fact from the outgoing message, assuming the intruder knows the private keys of all dishonest agents. (This is the worst-case assumption behind Dolev-Yao that all dishonest agents work together and we can thus think of them as one single intruder.) Moreover, if the intruder is dishonest, then they learn at this point both the values of x_A and x_B. Both is hardly avoidable, and so we just specify as part of this transaction that we *release* this information, again to be stored in the formula α. Here $x_A \doteq \gamma(x_A)$ refers to a third formula γ that is present besides α and β in every state: it represents the truth, mapping every privacy variable to its true value. Thus if $\gamma(x_A) = \mathsf{a}$, then we release here the formula $x_A \doteq \mathsf{a}$. Initially, the nonce N_A is a shared secret of the two protocols, but in the dishonest x_B case, the intruder is now able to learn N_A since x_B is the intended recipient for it and the nonce has to be *declassified*, while in the honest x_B case it would count as a *leakage* if the

```
Role $Initiator$
$\star\ x_A \in \mathsf{Honest}$.
$\star\ x_B \in \mathsf{Agent}$.
$PKB := lookup(x_A, x_B)$
;
$\nu N_A, R.$
$\mathsf{snd}(\mathsf{crypt}(PKB, \mathsf{f}_1(N_A, x_A), R))$.
if $x_B \in \mathsf{Honest}$ then
    $\star\ x_B \in \mathsf{Honest}$
else
    $\star\ x_A \doteq \gamma(x_A) \wedge x_B \doteq \gamma(x_B)$.
    $\mathsf{snd}(N_A)$
;
$\mathsf{rcv}(\mathsf{crypt}(\mathsf{pk}(x_A), \mathsf{f}_2(N_A, N_B, x_B), \_))$.
$\nu R'.$
$\mathsf{snd}(\mathsf{crypt}(PKB, \mathsf{f}_3(N_B), R'))$
```

```
Role $Responder$
$\mathsf{rcv}(\mathsf{crypt}(\mathsf{pk}(B), \mathsf{f}_1(N_A, A), \_))$.
if $B \notin \mathsf{Honest}$ then
    stop
;
$PKA := lookup(B, A)$
;
$\nu N_B, R.$
$\mathsf{snd}(\mathsf{crypt}(PKA, \mathsf{f}_2(N_A, N_B, B), R))$.
if $A \notin \mathsf{Honest}$ then
    $\mathsf{snd}(N_B)$
;
$\mathsf{rcv}(\mathsf{crypt}(\mathsf{pk}(B), \mathsf{f}_3(N_B), \_))$
```

```
Procedure $lookup(A, B)$
$\nu N, R.$
$\mathsf{snd}(\mathsf{scrypt}(\mathsf{sk}(A, \mathsf{s}), \mathsf{req}(B, N), R))$
;
$\mathsf{rcv}(\mathsf{scrypt}(\mathsf{sk}(A, \mathsf{s}), \mathsf{resp}(B, PKB, N), \_))$.
$\mathsf{assert}(PKB \doteq \mathsf{pk}(B))$.
$\mathsf{return}(\mathsf{pk}(B))$

Role $Server$
$\mathsf{rcv}(\mathsf{scrypt}(\mathsf{sk}(A, \mathsf{s}), \mathsf{req}(B, N), \_))$.
$\nu R.$
$\mathsf{snd}(\mathsf{scrypt}(\mathsf{sk}(A, \mathsf{s}), \mathsf{resp}(B, \mathsf{pk}(B), N), R))$
```

Fig. 1. Specification based on the Needham-Schroeder-Lowe public-key protocol

intruder finds out N_A as it is then between two honest agents.

The third and final transaction consists of both receiving the answer from x_B and sending the reply. Here, the receive message is with pattern matching: x_A only accepts the incoming message if it is encrypted with the public key $\mathsf{pk}(x_A)$ of x_A and contains the format f_2 of the second step of the protocol, where the first item is nonce N_A that x_A created earlier and x_B is the same name as before. Moreover, the variable N_B is bound at this point to whatever (apparently) x_B has sent as their nonce. The semantics of this pattern matching is that the role will just abort when receiving a message that does not fit; this also prevents that the intruder can send several messages attempting an online-guessing attack. We will later discuss in more detail why this pattern matching is not a further restriction on top of the assumption of a typed model, and how this can be reduced to simpler concepts.

Let us turn now to the Responder role. Here, we start with a receive message that contains again a pattern with variables B, N_A and A (the underscore corresponds to variable that is not used); similarly as before, we stop if the input has not the right form, but the variables B and N_A and A can be arbitrary and this is their binding occurrence. Thus, curiously, B “learns” its name from this message. Since we want that this is only executed by an honest agent, we stop here if $B \notin$ Honest. The stop means that in this case the following transactions of this role are not going to be executed. (In fact, this is the same stop that occurs when the pattern of a receive is not matched.) Our semantics assumes that the intruder can observe such a stop, because the agent ceases to communicate. The rest of the responder role is similar to the initiator role.

Let us now describe the protocol P_2 (given in Fig. 1, on the right). Here the first role is the procedure *lookup*, i.e., this role cannot spontaneously start or be triggered by a received message, but rather this is started by an invocation from P_1, binding the agent names A and B. Here we assume that each honest agent has a shared key $\mathsf{sk}(A, \mathsf{s})$ with the trusted key server s. Note that s is a constant of type agent i.e., this represents a fixed honest server that cannot be played by the intruder. The first message is an encrypted request (message format req) for the public key of B, including a fresh nonce N. We leave it as an exercise for the reader to consider a variant of the protocol where we omit this random number and the randomization value R of the encryption to show that this induces privacy, authentication, and freshness problems.

When the lookup receives the answer from the server, we have the assertion $PBK \doteq \mathsf{pk}(B)$. This reflects a protocol goal: it should never happen that the server returns another key $\mathsf{pk}(B)$ as answer to a request for public key of B. If such a mismatch should ever happen, then it would count as an attack; in this way we make this assertion part of the verification of the security of the lookup protocol. Finally, if the assertion has succeeded, the lookup returns the key it has received from the server. Since we can at this point be sure that $PBK \doteq \mathsf{pk}(B)$, we can simply return $\mathsf{pk}(B)$. Thus *lookup* is guaranteed to always give back the correct key, or fail with an attack before returning something.

A. The Composition

Now the composition is in some sense obvious: the call to lookup in the initiator and responder role shall be replaced with the lookup role under the respective instantiation of the formal parameters, and replacing PKB and PKA in the initiator and responder roles with the return values $\mathsf{pk}(B)$ and $\mathsf{pk}(A)$, respectively. The composed protocol consists then of this expanded initiator and responder roles of P_1 and the server role of P_2 (because the server is "called" by incoming messages, not a procedure call).

This gives the entire protocol as a monolith, and we rather want to verify the two protocols as components in isolation. However, in complete isolation, neither component really makes sense. This is because P_1 cannot really do anything without some form of mechanism to get the key of the intended recipient. While P_2 can work sort of independently, it really gets its actual goals from the context with P_1, namely that it is transporting the names and public keys of agents that are privacy variables chosen in P_1; we need this context to make clear that P_2 has the obligation to keep these values private.

We thus need to define an interface between the two protocols, and we do so by highlighting in each protocol which steps are relevant to the other protocol. This is the blue highlighting in the figures before. Basically this is saying: when verifying either protocol, one needs to consider at least the blue steps from the other protocol, because these are relevant things happening in the other protocol. Most notably this includes the non-deterministic choices like $x_A \in \mathsf{Honest}$, because here a privacy-critical information is introduced, as well as all released information that is released in α, like $x_A \doteq \gamma(x_A)\dots$. Procedure calls are replaced with the body of procedures so we do not highlight them here, but they are obviously relevant to the other protocol. Creation of fresh nonces is also highlighted, because it introduces fresh secrets that can in principle be used in either protocol.

All other kinds of steps are not necessarily highlighted: it depends on whether this step is deemed relevant for the other protocol. We will have below some requirements on the highlighting, but besides these requirements it is the choice of the modeler what to highlight. Naturally, we want as few highlighted steps as possible, because the more details can be hidden, the easier the verification task gets.

As can be seen in the specification, we have highlighted all the conditionals: in the Initiator, releases depend on the condition, in the first conditional of the Responder, it depends on the condition whether we proceed in the first place, and the second conditional is again containing a release. Of all the protocol messages we have only highlighted the first message of P_1. This is because the message is binding several of the variables in the responder role upon reception, in particular A and B that are again relevant for highlighted steps. Thus this binding occurrence is also relevant.

Highlighting a $\mathsf{snd}(t)$ step has another important meaning in our composition: it means that the message t is now *declassified*, i.e., the intruder may know it. We use declassification in the initiator role with $\mathsf{snd}(N_A)$ in the case x_B is dishonest: the intruder is just allowed to know N_A in this case, because one of the dishonest agents is the intended recipient of N_A.

We thus regard as a component protocol the individual steps of that component protocol plus all the highlighted steps of the other component protocol. In a nutshell, the compositionality result below is that an attack trace against the composed protocol can be transformed into an attack trace against one of the components. Thus it suffices to verify the security of the components to show the security of their composition—provided they satisfy the requirements of our compositionality theorem.

III. Specification

A. (α, β)-*Privacy*

We consider terms over an alphabet Σ, containing function and relation symbols with their arity, and interpret the terms in the quotient algebra modulo a congruence relation $\approx$. Functions can be either *public* or *private* (accessible/not accessible to the intruder); the set of public functions is $\Sigma_{pub} \subseteq \Sigma$. For our purpose, the congruence allows for constructors (e.g., encryption, hashing, pairing) and destructors (e.g., decryption, projection), where decryption failure yields a distinguished constant $\mathbb{ff}$; the set of constructors is $\Sigma_c \subseteq \Sigma$. Definition III.4 describes the precise class of algebraic theories that we support. Formulas (typically α, β, or ϕ) are in *Herbrand logic* [16], which is like standard First-Order Logic where the universe is said quotient algebra.

We use standard definitions like: $fv(\cdot)$ returning the free variables of a term or formula; ground terms (terms without any variable); linear terms (every variable occurs at most once); the interpretation $\mathcal{I}$ mapping all variables to the universe, and n-ary relations to a set of n-tuples of the universe; the models relation $\mathcal{I} \models \phi$ expressing that $\mathcal{I}$ is a satisfying interpretation for ϕ; $\equiv$ for logic equivalence of formulas (and for defining formulas).

The main idea of (α, β)-privacy is that every state of the world contains a formula α that represents what the intruder is allowed to know and the formula β represents what the intruder has actually observed. A state violates (α, β)-privacy iff some model of α can be ruled out by the intruder knowledge in β, i.e., the intruder has learned more than what is allowed. We use a sub-alphabet $\Sigma_0 \subset \Sigma$ containing the *payload symbols*, which are used to express the privacy goals. The complement $\Sigma \setminus \Sigma_0$ contains the *technical symbols*, which are used to represent the intruder knowledge (e.g., the cryptographic messages observed).

Definition III.1 ((α, β)-privacy [17])**.** *Given two formulas α over Σ_0 and β over Σ with $fv(\alpha) \subseteq fv(\beta)$, we say that (α, β)-*privacy holds *iff for every $\mathcal{I} \models \alpha$ there exists $\mathcal{I}' \models \beta$ such that $\mathcal{I}$ and $\mathcal{I}'$ agree on the variables in $fv(\alpha)$ and on the relation symbols in Σ_0.*

A *frame* is a notion that is commonly used to characterize the knowledge of the intruder: we have a set of distinguished constants called *labels* that do not occur in normal messages; a frame F maps such labels to messages. The *domain* of a frame is the set of the labels that the frames maps. The labels allow for describing intruder actions like encryption and decryption by a *recipe* r: a term built from labels and public functions. Here, $F(r)$ is the term that results by replacing all labels in r with the corresponding message from F; we ensure throughout the paper that $F(r)$ is only used when all labels in r occur in the domain of F. Whereas some works view frames as substitutions, for our purpose the order of the mappings is significant: we will use frames to record the messages observed by the intruder and their order.

We speak of an *experiment* for a frame F when the intruder checks whether two recipes r_1 and r_2 over the domain of F give the same result, i.e., whether $F(r_1) \approx F(r_2)$. We say two frames F_1 and F_2 are *statically equivalent*, written $F_1 \sim F_2$, iff there is no experiment to distinguish them, i.e., every experiment will either give the same result in both frames, or different results in both frames. (This implicitly requires that the frames have the same domain.)

B. Protocols

We distinguish two sorts of variables: the privacy variables, denoted with lowercase letters, are introduced in steps of the form $x \in D$ (where D is called the domain of x) and represent non-deterministic choices; the intruder variables, denoted with uppercase letters, represent messages that may depend on the intruder, e.g., messages received by honest agents.

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

- *a number of* roles *and* procedures *according to the syntax below;*
- *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 set* Γ_0 *of ground interpretations of the relations occurring in the roles and procedures.*

R		*Role or procedure*
$::=$	$P_l; R$	*Sequence*
$\mid$	P_l	*Transaction*
P_l		*Left process*
$::=$	$\star\ x \in D.P_l$	*Non-deterministic choice*
$\mid$	$\mathsf{rcv}(t).P_l$	*Receive*
$\mid$	$\mathsf{let}\ X = t.P_l$	*Let statement*
$\mid$	$X := \mathit{proc}(t, \ldots, t)$	*Procedure call*
$\mid$	P_c	*Center process*
P_c		*Center process*
$::=$	$X := \mathsf{cell}(t).P_c$	*Cell read*
$\mid$	$\mathsf{if}\ \phi\ \mathsf{then}\ P_c\ \mathsf{else}\ P_c$	*Conditional statement*
$\mid$	$\mathsf{let}\ X = t.P_c$	*Let statement*
$\mid$	stop	*Stop*
$\mid$	$\nu n_1, \ldots, n_k.P_r$	*Fresh constants*
P_r		*Right Process*
$::=$	$\mathsf{snd}(t).P_r$	*Send*
$\mid$	$\mathsf{cell}(t) := t.P_r$	*Cell write*
$\mid$	$\star\ \phi.P_r$	*Release*
$\mid$	$\mathsf{assert}(\phi).P_r$	*Assertion*
$\mid$	$\mathsf{let}\ X = t.P_r$	*Let statement*
$\mid$	$\mathsf{return}(t)$	*Return*
$\mid$	0	*Nil process*

where D is a finite set of public constants, t ranges over destructor-free messages (that do not contain $\mathbb{ff}$) and ϕ ranges over quantifier-free Herbrand logic formulas.

The non-deterministic choices, receives, let statements, procedure calls, cell reads and fresh constants are binding.

We require that every role is closed. Every procedure has a number of parameters, which are the free variables of the procedure's body, and every procedure call must give the appropriate number of arguments. Moreover, the last step of a procedure must be either a return *statement or* stop*. Finally, there must be no cycle in the graph of procedure calls.*

Definition III.3 (Syntactic sugar)**.** *For ease of notation, we allow the following in specifications:*

- *Trailing* 0 *and* $\mathsf{else}\ 0$ *can be omitted.*
- *Variables bound in a message received but otherwise never used can be written with a wildcard _ instead of a variable name.*
- *A step of a right process can be written in the left or center part with the meaning that this step is executed in every branch.*

Moreover, we will need for the semantics in Table I that the terms in messages received are linear with only fresh variables and no constants. In order to ensure this, we transform a message received, that may contain variables bound earlier or some constants, into a linear term with only fresh variables and no constants. We then insert a conditional statement to check that the fresh variables have the expected values.

Formally, for a message received of the form $R; P_1.\mathsf{rcv}(t).P_2.P_3$*, where R is a sequence of transactions, $P_2.P_3$ is a left process and P_3 is a center process, we obtain a linear term t' by replacing in t every variable already bound in R or P_1 and every constant with fresh intruder variables, and we replace* $\mathsf{rcv}(t).P_2.P_3$ *with* $\mathsf{rcv}(t').P_2.\mathsf{if}\ t \doteq t'\ \mathsf{then}\ P_3\ \mathsf{else}\ \mathsf{stop}$.

We have already presented informally most of the steps in processes when describing the example in §II. We explain below the rest of protocol specifications. The details of the semantics for processes are given in Definition IV.8.

The specification defines a set Γ_0 of interpretations for the relations that are used in the processes, and we consider that the intruder knows that in every concrete execution, the true interpretation is in Γ_0 (but they do not know a priori which one). Relations can model some context about the agents participating in the protocol. For instance, in our model of NSL, we could have a relation $\mathit{talk}(\cdot, \cdot)$ over agent names that represents whether some agent wants to talk to another one. Then the set of interpretations could contain, e.g., some interpretations where Alice (denoted a) wants to talk to Bob (denoted b), i.e., where $\mathit{talk}(\mathsf{a}, \mathsf{b})$ holds, and some other interpretations where it is not the case, modeling that the intruder does not know whether Alice actually wants to talk to Bob. We could then require that all interpretations agree on $\mathit{talk}(x, y)$ whenever x is dishonest, modeling that initially the intruder knows who dishonest agents want to talk to.

In a let statement, a fresh variable can be bound to a message, where the scope of the binding is the rest of the role or procedure (and not just the current transaction).

The memory cells make protocols stateful: transactions can read from or write to memory, so the execution of the protocol may depend on the state of the memory.

The class of algebraic theories that we support can be used to model common cryptographic operators such as symmetric and asymmetric encryption/decryption, pairing and projection (opening of pairs), digital signatures and opening/verification of the signatures etc. The behavior of the cryptographic operators is defined as a set of rewrite rules and gives rise to a congruence relation.

Definition III.4 (Algebraic theory). *A* constructor/destructor rule *is a rewrite rule of one of the following forms:*

- *Decryption:* $d(k, c(k', X_1, \ldots, X_n)) \to X_i$ *where* d *is a destructor,* c *is a constructor, the* X_j *are distinct variables,* $i \in \{1, \ldots, n\}$ *and* $fv(k) = fv(k')$.
- *Projection:* $d_i(c(X_1, \ldots, X_n)) \to X_i$ *where* $i \in \{1, \ldots, n\}$, d_i *is a public destructor called a* projector, c *is a constructor of arity* n *and the* X_j *are distinct variables. There must be such a rule for every* $i \in \{1, \ldots, n\}$ *and* c *is then called* transparent.
- *Private extraction:* $d(c(t_1, \ldots, t_n)) \to t_0$ *where* d *is a private destructor called a* private extractor, c *is a constructor and* t_0 *is a subterm of one of the* t_i.

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

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

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

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

IV. Symbolic Execution

The grammar of specifications allows for procedure calls, where one protocol is calling a process that may be specified by another protocol. Procedure calls can be used to define the interface between two protocols, e.g., one protocol obtains a key that is established by another protocol. The semantics of a procedure call is essentially to inline the body of the procedure. Formally, we define the replacement of procedure calls with the processes they represent as *procedure call expansion*.

Definition IV.1 (Procedure call expansion). *Let* R_1, R_2 *be sequences of transactions such that* R_1 *does not contain any procedure call and let* $P.X := proc(t_1, \ldots, t_n)$ *be a transaction, where* $proc$ *is a procedure with parameters* $X_1, \ldots, X_n$ *and body* R.

$$\begin{aligned} &expand(R_1; P.X := proc(t_1, \ldots, t_n); R_2) \\ &\quad = R_1; expand(P.\sigma(R'); R_2) \end{aligned}$$

where $\sigma = [X_1 \mapsto t_1, \ldots, X_n \mapsto t_n]$ *and* R' *is the same as* R *except that every* $\mathsf{return}(t)$ *is replaced with* $\mathsf{let}\ X = t.0$, *i.e., the return value is bound to variable* X.

Remark. Since the graph of procedure calls is acyclic, the expansion terminates. $\lhd$

The semantics of protocols is a symbolic execution performed by the intruder, giving rise to a state transition system. In every state, there is a formula α called payload that expresses the information released so far, a formula γ called truth that interprets every relation and privacy variable in one protocol execution (the intruder does not know γ a priori), and a set $\mathcal{P}$ of possibilities that correspond to all the branches in the processes that the intruder has not ruled out yet: the intruder has made some concrete observations, and may eliminate some possibilities if they can be distinguished from their observations.

Every state also contains a sequence ρ called choice of recipes that stores the recipes used by the intruder whenever the protocol was receiving a message, a boolean *flag* that tracks whether some assertion was broken, and a thread ID *TID* that keeps track of the interleaving of transactions: the stop step in a process is stopping the transactions coming from one particular role instance, i.e., one thread, and thus we have to remember which thread is affected by a stop.

Definition IV.2 (State). *A* state *is a tuple* $(\alpha, \gamma, \mathcal{P}, \rho, \mathit{flag}, \mathit{TID})$ *such that:*

- α *is a* Σ_0*-formula, the* payload*;*
- γ *is a* Σ_0*-formula, the* truth*;*
- $\mathcal{P}$ *is a set of* possibilities*, which are each of the form* $(R, \phi, \mathit{struct}, \delta)$*, where* R *is a sequence of processes,* ϕ *is a* Σ*-formula,* struct *is a frame and* δ *is a sequence of memory updates of the form* $\mathsf{cell}(s) := t$ *for messages* s *and* t*;*
- ρ *is a sequence of recipes, the* choice of recipes *made so far for every message sent by the intruder;*
- *flag is either true or false, where the flag set to true represents that some assertion did not hold;*
- *TID represents the thread ID of the current transaction executed.*

Definition IV.3 (Well-formed state). *Let* $\mathcal{P} = \{(R_1, \phi_1, \mathit{struct}_1, \delta_1), \ldots, (R_n, \phi_n, \mathit{struct}_n, \delta_n)\}$ *be the possibilities in a state* $S = (\alpha, \gamma, \mathcal{P}, \rho, \_, \_)$*. Then* S *is* well-formed *iff*

- $\alpha \models \bigvee_{\gamma_0 \in \Gamma_0} \gamma_0$*;*
- $\gamma \models \alpha \wedge \bigvee_{i=1}^{n} \phi_i$*;*
- *the* ϕ_i *are such that* $\models \neg(\phi_i \wedge \phi_j)$ *for* $i \neq j$ *and* $fv(\phi_i) \subseteq fv(\alpha)$*;*

- *the $struct_i$ are frames with the same labels occurring in the same order;*
- *the recipes in ρ are over the domain of the $struct_i$.*

S is a milestone *iff every sequence R_i starts with the nil process, and S is an* intermediate state *otherwise.*

In the rest of the paper, we only consider well-formed states (our semantics ensures that the reachable states of a protocol are well-formed).

In a given state, the privacy goals are expressed through the payload α. The intruder knowledge is expressed through a formula β, which is defined based on the possibilities in that state. The intruder has made concrete observations, recorded in a frame $concr$, they consider a set of possibilities containing $\phi_i, struct_i$ such that, if condition ϕ_i holds, then the concrete instances of $struct_i$ are statically equivalent with $concr$.[1]

Definition IV.4 (Intruder knowledge)**.** *Given a well-formed state $S = (\alpha, \gamma, \mathcal{P}, \_, \_, \_)$, let $concr = \gamma(struct_j)$ where $(\_, \phi_j, struct_j, \_) \in \mathcal{P}$ is the unique possibility such that $\gamma \models \phi_j$. The* intruder knowledge *in state S is defined as*

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

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

A. Typed Model

In this paper, we consider that a protocol specification contains type annotations so that every message in a protocol execution has a type. We have a set of atomic types, e.g., a type for nonces and a type for agent names, and composed types are built with atomic types and public constructors.

Definition IV.5 (Typing function)**.** *A* typing function *Γ is such that:*

- *$\Gamma(c)$ is atomic for every function $c \in \Sigma$ of arity 0.*
- *$\Gamma(f(t_1, \ldots, t_n)) = f(\Gamma(t_1), \ldots, \Gamma(t_n))$ for every constructor $f \in \Sigma$ of arity $n > 0$.*
- *$\Gamma(x)$ is a type (atomic or composed) for every variable $x \in \mathcal{V}$.*

Note we do not define the typing for destructors, as the point is that we will only consider destructor-free messages based on our congruence relation (but recipes chosen by the intruder may well use destructors).

Definition IV.6 (Well-typed substitution)**.** *A substitution σ is* well-typed *iff for every $x \in dom(\sigma)$, we have $\Gamma(x) = \Gamma(\sigma(x))$.*

We assume that in a state, for every label, the different frames map the label to messages of the same type. We are working in this paper in a typed model where we assume that the intruder only uses well-typed messages: whenever the possibilities are receiving a message $\mathsf{rcv}(t)$, the intruder chooses a recipe r such that $\Gamma(struct(r)) = \Gamma(t)$, where $struct$ is the frame in one of the possibilities (the type of the message produced by the recipe is the same in every frame because we assume that the different frames map, for a given label, to messages of the same type).

B. State Transition System

A protocol specification defines a set of roles, which are sequences of transactions, and a concrete protocol execution corresponds to an interleaving of such transactions. We attach a *thread ID* to every transaction and use the thread IDs in order to consider sequences with correct interleaving, and to express the semantics of stopping a role.

Definition IV.7 (Thread filtering)**.** *Let $P_1; \ldots; P_n$ be a sequence of transactions (after expansion), where every transaction is annotated with a* thread ID*. We define $filter_{=TID}(P_1; \ldots; P_n)$ as the largest subsequence that only contains transactions with thread ID equal to TID. Conversely, $filter_{\neq TID}(P_1; \ldots; P_n)$ is the largest subsequence that does not contain any transaction with thread ID equal to TID.*

The symbolic execution is defined through a set of rules that are transitions between states. Each transition evaluates one step of the processes.

Definition IV.8 (Semantics)**.** *The semantics of the symbolic execution is given in Table I. The changes to the state are colored in blue. The semantics defines a relation $\rightarrow$ on states. Let S, S' be two milestones. We define $\longrightarrow$ as the relation such that $S \longrightarrow S'$ iff $S \rightarrow^* S'$ and with the requirement that whenever the Eliminate rule is applicable, then it must be applied.*

Remark. The symbolic execution maintains some invariants, assuming that every possibility starts initially with the same sequence of transactions, with the same thread IDs:

- The non-deterministic choices happen at the same time in every possibility, because these steps occur before any branching.
- The receives also happen at the same time in every possibility, because these steps occur before any branching and also because we are only considering well-typed instantiations and messages received are linear with only fresh intruder variables and no constants, so in every frame, the matching problem has a solution.
- The thread ID of the transaction being executed is the same in every possibility and is also the thread ID in the current state, because the possibilities are synchronized when starting the next transaction in the sequence.
- There is always a single possibility with condition ϕ such that $\gamma \models \phi$ in the current state, which means that when every possibility is either stopping, sending a message or reaching a milestone, then exactly one of the rules Stop, Send and Milestone is applicable. If any possibility starts with a different step, then another rule must be applied.
- The messages in the frames can be considered destructor-free without loss of generality, even though they may

[1] The details of formalizing frames and their static equivalence in Herbrand logic are found in [18].

TABLE I
SEMANTICS OF THE SYMBOLIC EXECUTION

Choice	$(\alpha, \gamma, \{(\star\ x \in D.P_i; R_i, \phi_i, struct_i, \delta_i) \mid i \in \{1, \ldots, n\}\}, \rho, flag, TID)$ $\rightarrow (\alpha \wedge x \in D, \gamma \wedge x \doteq c, \{(P_i; R_i, \phi_i, struct_i, \delta_i) \mid i \in \{1, \ldots, n\}\}, \rho, flag, TID)$ for every $c \in D$
Receive	$(\alpha, \gamma, \{(\mathsf{rcv}(t).P_i; R_i, \phi_i, struct_i, \delta_i) \mid i \in \{1, \ldots, n\}\}, \rho, flag, TID)$ $\rightarrow (\alpha, \gamma, \{(\sigma_i(P_i; R_i), \phi_i, struct_i, \delta_i) \mid i \in \{1, \ldots, n\}\}, \rho.r, flag, TID)$ for every r over the domain of the $struct_i$, where σ_i is such that $\sigma_i(t) = struct_i(r)$
Let	$(\alpha, \gamma, \{(\mathsf{let}\ X = t.P; R, \phi, struct, \delta)\} \uplus \mathcal{P}, \rho, flag, TID)$ $\rightarrow (\alpha, \gamma, \{((P; R)[X \mapsto t], \phi, struct, \delta)\} \cup \mathcal{P}, \rho, flag, TID)$
Cell read	$(\alpha, \gamma, \{(X := \mathsf{cell}(s).P; R, \phi, struct, \delta)\} \uplus \mathcal{P}, \rho, flag, TID)$ $\rightarrow (\alpha, \gamma, \{(P'; R, \phi, struct, \delta)\} \cup \mathcal{P}, \rho, flag, TID)$ where the updates on cell in δ are $\mathsf{cell}(s_1) := t_1. \cdots .\mathsf{cell}(s_k) := t_k$, the initial value of cell is given with ground context $C[\cdot]$ and $P' = \mathsf{if}\ s \doteq s_1\ \mathsf{then\ let}\ X = t_1.P\ \mathsf{else} \ldots \mathsf{if}\ s \doteq s_k\ \mathsf{then\ let}\ X = t_k.P\ \mathsf{else\ let}\ X = C[s].P$
Cell write	$(\alpha, \gamma, \{(\mathsf{cell}(s) := t.P; R, \phi, struct, \delta)\} \uplus \mathcal{P}, \rho, flag, TID)$ $\rightarrow (\alpha, \gamma, \{(P; R, \phi, struct, \mathsf{cell}(s) := t.\delta)\} \cup \mathcal{P}, \rho, flag, TID)$
Conditional	$(\alpha, \gamma, \{((\mathsf{if}\ \psi\ \mathsf{then}\ P_1\ \mathsf{else}\ P_2); R, \phi, struct, \delta)\} \uplus \mathcal{P}, \rho, flag, TID)$ $\rightarrow (\alpha, \gamma, \{(P_1; R, \phi \wedge \psi, struct, \delta), (P_2; R, \phi \wedge \neg\psi, struct, \delta)\} \cup \mathcal{P}, \rho, flag, TID)$
Release	$(\alpha, \gamma, \{(\star\ \psi.P; R, \phi, struct, \delta)\} \uplus \mathcal{P}, \rho, flag, TID)$ $\rightarrow (\alpha', \gamma, \{(P; R, \phi, struct, \delta)\} \cup \mathcal{P}, \rho, flag, TID)$ where $\alpha' \equiv \alpha \wedge \psi$ if $\gamma \models \phi$ and $\alpha' \equiv \alpha$ otherwise
Assert	$(\alpha, \gamma, \{(\mathsf{assert}(\psi).P; R, \phi, struct, \delta)\} \uplus \mathcal{P}, \rho, flag, TID)$ $\rightarrow (\alpha, \gamma, \{(P; R, \phi, struct, \delta)\} \cup \mathcal{P}, \rho, flag', TID)$ where $flag' = \mathsf{true}$ if $\gamma \models \phi \wedge \neg\psi$ and $flag' = flag$ otherwise
Eliminate	$(\alpha, \gamma, \{(P; R, \phi, struct, \delta)\} \uplus \mathcal{P}, \rho, flag, TID)$ $\rightarrow (\alpha, \gamma, \mathcal{P}, \rho, flag, TID)$ if $\beta \models \neg\phi$, where β is the intruder knowledge in the predecessor state
Stop	$(\alpha, \gamma, \{(\mathsf{stop}; R_i, \phi_i, struct_i, \delta_i) \mid i \in \{1, \ldots, n\}\} \uplus \mathcal{P}, \rho, flag, TID)$ $\rightarrow (\alpha, \gamma, \{(0; filter_{\neq TID}(R_i), \phi_i, struct_i, \delta_i) \mid i \in \{1, \ldots, n\}\}, \rho, flag, TID)$ if $\gamma \models \bigvee_{i=1}^{n} \phi_i$ and every process in $\mathcal{P}$ starts with $\mathsf{snd}(\cdot)$ or 0
Send	$(\alpha, \gamma, \{(\mathsf{snd}(t_i).P_i; R_i, \phi_i, struct_i, \delta_i) \mid i \in \{1, \ldots, n\}\} \uplus \mathcal{P}, \rho, flag, TID)$ $\rightarrow (\alpha, \gamma, \{(P_i; R_i, \phi_i, struct_i.l \mapsto t_i, \delta_i) \mid i \in \{1, \ldots, n\}\}, \rho, flag, TID)$ if $\gamma \models \bigvee_{i=1}^{n} \phi_i$ and every process in $\mathcal{P}$ starts with stop or 0, where l is a fresh label
Milestone	$(\alpha, \gamma, \{(0; R_i, \phi_i, struct_i, \delta_i) \mid i \in \{1, \ldots, n\}\} \uplus \mathcal{P}, \rho, flag, TID)$ $\rightarrow (\alpha, \gamma, \{(0; R_i, \phi_i, struct_i, \delta_i) \mid i \in \{1, \ldots, n\}\}, \rho, flag, TID)$ if $\gamma \models \bigvee_{i=1}^{n} \phi_i$ and every process in $\mathcal{P}$ starts with stop or $\mathsf{snd}(\cdot)$
Next	$(\alpha, \gamma, \{(0; R_i, \phi_i, struct_i, \delta_i) \mid i \in \{1, \ldots, n\}\}, \rho, flag, TID)$ $\rightarrow (\alpha, \gamma, \{(R_i, \phi_i, struct_i, \delta_i) \mid i \in \{1, \ldots, n\}\}, \rho, flag, TID')$ where TID' is the thread ID of the first transaction in the sequences R_i

contain privacy variables. The intruder can always compare the outcome of a destructor with what happens in the concrete frame, since we allow destructors in recipes. The Eliminate rule allows for eliminating all frames that are not statically equivalent to the concrete frame observed by the intruder. Therefore, for every message received by a process, even if a recipe chosen by the intruder contains a destructor, either the destructor gives a subterm in every frame, or the constant $\mathtt{ff}$ in every frame. ◁

Definition IV.9 (Fresh role instance)**.** *Let R be a role. A* fresh instance *R' of R is obtained by*

1) *Renaming all variables apart in the role R and in the procedures it calls.*
2) *Expanding the role.*
3) *Removing every step $\nu n_1, \ldots, n_k$ by instantiating the n_i with fresh constants.*
4) *Giving every transaction in the resulting R' the same fresh thread ID.*

We now have all the necessary concepts to define the overall state transition system and the traces of a protocol. A trace consists of the sequence of transactions executed, the interpretation of all relations and privacy variables chosen in those transactions, and the recipes chosen for the messages sent by the intruder (i.e., messages received in the transactions). Recall that the intruder does not know a priori what is the true interpretation of relations: the protocol specifies a set Γ_0 of possible interpretations, and for the symbolic execution we need to fix this interpretation. Thus, for the set of reachable states we consider several initial states where for each, we fix some $\gamma_0 \in \Gamma_0$.

Definition IV.10 (State transition system)**.** *A sequence of transactions $P_1; \ldots; P_n$ is* valid *iff there exist role instances $R_1, \ldots, R_m$, with thread IDs $TID_1, \ldots, TID_m$, such that for every $j \in \{1, \ldots, m\}$, $filter_{=TID_j}(P_1; \ldots; P_n)$ is a prefix of*

R_j.

Let γ be a Σ_0-formula such that $\gamma \models \bigvee_{\gamma_0 \in \Gamma_0} \gamma_0$ and $P_1; \ldots; P_n$ be a valid sequence of transactions. Then the initial state w.r.t. γ and $P_1; \ldots; P_n$ *is*

$$\begin{aligned} &\mathit{init}(\gamma, P_1; \ldots; P_n) \\ &\quad = (\bigvee_{\gamma_0 \in \Gamma_0} \gamma_0, \gamma, \{(0; P_1; \ldots; P_n, \mathsf{true}, [], [])\}, [], \mathsf{false}, 0) \end{aligned}$$

where $[]$ denotes the empty frame, empty memory and empty choice of recipes.

A tuple $(P_1; \ldots; P_n, \gamma, \rho)$ is a trace *iff $P_1; \ldots; P_n$ is valid, $\gamma \models \bigvee_{\gamma_0 \in \Gamma_0} \gamma_0$ and there exists a milestone $S = (\_, \gamma, \_, \rho, \_, \_)$ such that $\mathit{init}(\gamma_0, P_1; \ldots; P_n) \longrightarrow S$, where $\gamma_0 \in \Gamma_0$ is the interpretation of relations such that $\gamma \models \gamma_0$. The milestone S is then called a* reachable state.

V. Composition and Composability

A. Composition

We consider a composed protocol as the composition of two smaller specifications (the definitions and results can be generalized to an arbitrary number of specifications, we use two here for convenience of notation). Our main result is that, for the class of *composable* protocols (see Definition V.4), an attack trace on the composed protocol can be projected to an attack on one smaller protocol. To achieve this, every protocol specifies its *interface* and our projection corresponds to an individual protocol composed with the interface of the other protocol.

In the following we adopt the marking convention from [11] where all protocol steps are marked as 1, 2, or $\star$ for individual steps of the component protocols P_1 and P_2 and the interface, respectively. We used highlighting for the interface in Fig. 1 to avoid confusion with the $\star$ of non-deterministic choices and releases.

Definition V.1 (Protocol composition)**.** *Let Spec_1 and Spec_2 be protocols. The* composition $\mathit{Spec}_1 \parallel \mathit{Spec}_2$ *is the protocol defined with:*

- *the union of the roles, procedures, memory cells[2] and algebraic theories from Spec_1 and Spec_2; and*
- *the set of interpretations Γ_0, where we assume that both protocols specify the same set.*

In the context of a composed protocol, we consider that some steps of the processes specified are marked with either a protocol-specific index or the symbol $\star$. Let $i \in \{1, 2\}$ and P be a transaction from Spec_i. In the specification of P, the steps for receives, let statements, procedure calls, conditional statements, cell reads or writes, sends and assertions are marked with either i or $\star$. However, the steps for non-deterministic choices, stops, fresh constants, releases and nil processes are not marked because they are always relevant for the composition.

During procedure call expansion, every statement let $X = t$ *that replaces a* $\mathsf{return}(t)$ *is marked with the same mark as the procedure call that is being expanded.*

In frames, every mapping has the mark of the corresponding $\mathsf{snd}(\cdot)$ *step.*

Our main result is that we can *project* an attack on a composed protocol to an attack on a smaller protocol. In the context of composition, every step in a process has a mark that says whether it is part of the protocol's interface, i.e., it must always be present in the projection, or it is protocol-specific and can be abstracted away in the projection.

Definition V.2 (Projection to one component)**.** *Let $i, j \in \{1, 2, \star\}$. The projection of a transaction (after expansion) is defined below.*

$$\begin{aligned}
(\star\ x \in D.P)|_j &= \star\ x \in D.P|_j \\
(i : \mathsf{rcv}(t).P)|_j &= \begin{cases} \mathsf{rcv}(t).P|_j & \text{if } i \in \{j, \star\} \\ P|_j & \text{otherwise} \end{cases} \\
(i : \mathsf{let}\ X = t.P)|_j &= \begin{cases} \mathsf{let}\ X = t.P|_j & \text{if } i \in \{j, \star\} \\ P|_j & \text{otherwise} \end{cases} \\
(i : X := \mathsf{cell}(s).P)|_j &= \begin{cases} X := \mathsf{cell}(s).P|_j & \text{if } i \in \{j, \star\} \\ P|_j & \text{otherwise} \end{cases} \\
(i : \mathsf{if}\ \phi\ \mathsf{then}\ P\ \mathsf{else}\ Q)|_j &= \begin{cases} \mathsf{if}\ \phi\ \mathsf{then}\ P|_j & \\ \quad \mathsf{else}\ Q|_j & \text{if } i \in \{j, \star\} \\ P|_j & \text{otherwise} \end{cases} \\
(\mathsf{stop})|_j &= \mathsf{stop} \\
(\nu n_1, \ldots, n_k.P)|_j &= \nu n_1, \ldots, n_k.P|_j \\
(i : \mathsf{snd}(t).P)|_j &= \begin{cases} \mathsf{snd}(t).P|_j & \text{if } i \in \{j, \star\} \\ P|_j & \text{otherwise} \end{cases} \\
(i : \mathsf{cell}(s) := t.P)|_j &= \begin{cases} \mathsf{cell}(s) := t.P|_j & \text{if } i \in \{j, \star\} \\ P|_j & \text{otherwise} \end{cases} \\
(\star\ \phi.P)|_j &= \star\ \phi.P|_j \\
(i : \mathsf{assert}(\phi).P)|_j &= \begin{cases} \mathsf{assert}(\phi).P|_j & \text{if } i \in \{j, \star\} \\ P|_j & \text{otherwise} \end{cases} \\
(0)|_j &= 0
\end{aligned}$$

The notion of projection is extended to sequences of transactions and protocols. For a frame F, we define $F|_i$ as the projection of the frame to mappings marked with i or $\star$.

B. Composability

In general, the composition of secure protocols is not necessarily secure. In this section, we identify a set of requirements that form the class of protocols called *composable*. These requirements control the interface between protocols and the messages they share.

As mentioned in previous sections, in this paper we only consider well-typed instantiations. In an execution of the protocol, the concrete messages observed are ground instances

[2] If there are shared memory cells, then both protocols must specify the same ground contexts for the initial values of these cells.

of the terms that occur in the specification. We define below the notion of (ground) sub-message patterns and we will express our requirements using these ground patterns.

Definition V.3 ((Ground) sub-message patterns)**.** *The set of* sub-message patterns*, $\mathit{SMP}(M)$, of a set of terms M is the least set closed under the following rules:*

1) *If $t \in M$, then $t \in \mathit{SMP}(M)$.*
2) *If $t \in \mathit{SMP}(M)$ and t' is a subterm of t, then $t' \in \mathit{SMP}(M)$.*
3) *If $t \in \mathit{SMP}(M)$ and σ is a well-typed substitution, then $\sigma(t) \in \mathit{SMP}(M)$.*
4) *If $t \in \mathit{SMP}(M)$, k and t' are terms such that for some destructor d we have $d(k,t) \rightarrow t'$ as an instance of the rewrite rule for d, then $k \in \mathit{SMP}(M)$.*

The set of ground sub-message patterns*, $\mathit{GSMP}(M)$, of M is $\{t \in \mathit{SMP}(M) \mid t \text{ is ground}\}$.*

Let $\mathcal{T}_{pub} = \mathcal{T}(\Sigma_c \cap \Sigma_{pub}, \emptyset)$ denote the set of destructor-free public ground terms. Let $\mathit{Spec} = \mathit{Spec}_1 \parallel \mathit{Spec}_2$ be a composed protocol and $\mathit{Secrets}$ be a set of ground terms disjoint from $\mathcal{T}_{pub}$. The set $\mathit{Secrets}$ represent shared messages that are specified as secrets, i.e., they should not be known by the intruder. Our result is parameterized over this set of secret terms. It can be convenient to have protocols that share terms, where the terms themselves are not secrets but contain secrets as subterms. For instance, some protocols may use the same public key certificates signed with a trusted private key. We support this by specifying such terms as secrets, and when messages are sent they can be marked as *declassified* so that the intruder is now allowed to learn those messages.

Let GSMP_i denote the ground sub-message patterns of the terms occurring in $\mathit{Spec}|_i$ for $i \in \{1,2\}$, and let $\mathit{GSMP}_\star = \mathit{GSMP}_1 \cap \mathit{GSMP}_2$. The intersection $\mathit{GSMP}_\star$ corresponds to all ground messages (and their subterms) that are shared by the protocols.

We now list our requirements on composed protocols and we provide an intuitive explanation for them just afterwards.

Definition V.4 (Composability)**.** *Spec is* composable *w.r.t. $\mathit{Secrets}$ iff*

1) *For every conditional statement* if ϕ then P else Q*, if the branching is not marked with $\star$, then $Q =$* stop*.*
2) *$\mathit{GSMP}_\star \subseteq \mathcal{T}_{pub} \cup \mathit{Secrets}$.*
3) *Every role in $\mathit{Spec}|_1$ and $\mathit{Spec}|_2$ is well-formed (in the sense of Definition III.2).*
4) *Every cell read or cell write is marked with $\star$ iff the memory cell is shared (i.e., occurs in both Spec_1 and Spec_2).*
5) *If a process calls a procedure from the other protocol, then the procedure call is marked with $\star$.*
6) *In every formula released ϕ, all the variables $fv(\phi)$ are privacy variables chosen earlier in that role or procedure.*
7) *In every transaction, if there are two branches that send the same number of messages, then the marks of the messages sent must match.*

1) If the branching is protocol-specific, we want the projection to abstract it away. Given an attack trace on the composed protocol, we will argue why it is not necessary for the intruder to reach the second branch given that it stops immediately, so that we only have to consider the projection of the first branch.
2) Messages shared by the two protocols must be either public or specified as secrets. We will have to verify that the components do not leak any secret, and if that is the case then we can show (also using Item 6) that the intruder never needs to use a protocol-specific message, say from protocol 1, when executing protocol 2.
3) When projecting to 1 or 2, the resulting (expanded) roles must be well-formed so that the semantics is well-defined for the projected protocols; in particular, the projected roles must be closed.
4) Shared memory cells are a way for the two protocols to interact, and in order for the projection to faithfully represent the behavior of the original role or procedure, all reads and writes must also be present in the projection.
5) Procedure calls are another way for the two protocols to interact, and the values returned by these calls must also be present in the projection.
6) We exclude releasing information on variables that are bound in messages received, procedure calls or cell reads, i.e., variables that are not necessarily privacy variables. This is a strong restriction that seems necessary to obtain an intermediate result, which in short means that whenever a protocol is receiving a message, we can consider that the intruder sends a message from that same protocol (and not the other one) without loss of generality. Intuitively, since releases are changing the information allowed about privacy variables, it makes sense that an agent only releases information about the choices they have made themselves and not arbitrary messages that may or may not contain privacy variables chosen in other transactions.
7) In every reachable state, the frames in the different possibilities are indistinguishable and we must further have that a given label has the same mark in every possibility. Intuitively, the intruder should know whether a message is protocol-specific (and then from which protocol it comes from) or shared (and then it is always present in the projections).

VI. The Compositionality Results

A. Compositionality on the Frame Level

In the rest of the paper, we consider a composable protocol Spec. Before concluding on the compositionality for the protocol itself, we present results on frames. We give here short and informal proof sketches and all detailed proofs and intermediate results are provided in appendix.

Definition VI.1. *A ground frame F is* well-formed *iff for every mapping $i : l \mapsto t$ in F, we have $i \in \{1,2,\star\}$ and $t \in \mathit{GSMP}_i$.*

The set of declassified terms *of F is declassified$(F) = \{t \mid \exists r.\ F|_\star(r) \approx t\}$. We say that F* leaks a secret from *Secrets iff there exist $t \in Secrets \setminus declassified(F)$, $i \in \{1,2\}$ and r such that $F|_i(r) \approx t$. The frame F is* leakage-free *iff F does not leak any secrets.*

Remark. We define that a secret is leaked when it is leaked in one projection of the frame instead of the full frame. This is crucial, because leakage-freeness is a requirement of the composition; due to our construction it is only necessary to check that the component protocols do not leak (rather than having to check their composition for leakage). $\triangleleft$

We introduce the notion of *homogeneous* recipes to describe recipes that can be used in a projection of the composed protocol, i.e., recipes that are not using protocol-specific messages from two different protocols.

Definition VI.2. *Let F be a ground frame. A recipe r over $dom(F)$ is* homogeneous *iff there exists $i \in \{1,2\}$ such that every label in r is marked with i or $\star$. A pair of recipes (r_1, r_2) is homogeneous iff there exists $i \in \{1,2\}$ such that every label in r_1 and r_2 is marked with i or $\star$.*

The results in this section are formulated for ground frames, and we will later use them with frames from a reachable state. Thus we only consider frames that have the same domain, with the same marks for the labels.

Definition VI.3. *Two ground frames F_1 and F_2 are* comparable *iff they have the same domain and for every label, they agree on its mark.*

The first result is that, if a frame does not leak any secret, then for every message occurring in the protocol execution, we can obtain a homogeneous recipe to produce that message.

Lemma VI.1. *Let F be a leakage-free frame, $i \in \{1,2,\star\}$ and r be a recipe such that $F(r) \in GSMP_i$. Then there exists a recipe r' such that $F(r) \approx F(r')$ and every label in r' is marked with i or $\star$.*

Proof sketch. If r is not homogeneous, then there is a subterm of $F(r)$ that is in $GSMP_\star$ and thus this subterm must be either public or a secret. If it is a secret, then it must be declassified since the frame is leakage-free. Then the protocol-specific label producing the subterm can be replaced with either a public term or a recipe containing only labels marked with $\star$ (for the declassified case). $\triangleleft$

The second result is that, if both frames do not leak any secret and are not statically equivalent, then we can find a witness against static equivalence that is homogeneous. Recall that the intruder knowledge in a state is defined through static equivalence between the frames that the intruder considers possible in that state. Thus the lemma is useful to prove that a violation of privacy in a reachable state of the composed protocol can be mapped to a violation of privacy in a state of a projection of the protocol.

Theorem VI.1. *Let F_1, F_2 be leakage-free comparable frames. If for every $i \in \{1,2\}$, $F_1|_i \sim F_2|_i$, then $F_1 \sim F_2$.*

Proof sketch. We proceed by contraposition and assume that $F_1 \not\sim F_2$, so there exists a witness against static equivalence. Then we apply a reduction strategy that successively replaces some labels in the witness with homogeneous recipes until the witness is homogeneous. During these reduction steps, we may find another simpler homogeneous witness, e.g., if two recipes for key terms suffice to distinguish the frames without needing to apply decryption. The main argument in the reduction is that if a recipe is not homogeneous, then there is a term that is shared between protocols and since the frames are leakage-free, we can find a homogeneous recipe for that term. In the end we get a homogeneous witness, which means that we can project the frames such that the witness is still well-defined (i.e., using only labels available in the projected frames). $\triangleleft$

B. Compositionality on the State Level

Definition VI.4 (Leakage-free state and protocol)**.** *A state S is* leakage-free *iff the concrete frame concr in that state is leakage-free. A protocol is* leakage-free *iff every reachable state is leakage-free.*

The properties that we are verifying for a protocol are *reachability* properties, even privacy, so a protocol has an attack iff some reachable state has an attack.

Definition VI.5 (Attack state and attack trace)**.** *A milestone S is an* attack state *iff at least one of the following is true:*

- *The flag in S is set to true.*
- *S is not leakage-free.*
- *S does not satisfy privacy.*

Let $(P_1; \ldots; P_n, \gamma, \rho)$ be a trace and $S_0, \ldots, S_n$ be the milestones such that for every $i \in \{1, \ldots, n\}$, executing P_i, starting from S_{i-1}, leads to S_i (following the truth γ and using the recipes in ρ). $(P_1; \ldots; P_n, \gamma, \rho)$ is an attack trace *iff*

- *For every $i \in \{0, \ldots, n-1\}$, S_i is not an attack state.*
- *S_n is an attack state.*

Finally we can state our main result: if no projection of the protocol has an attack, then also the composed protocol has no attack.

Theorem VI.2. *If for every $i \in \{1,2\}$, $Spec|_i$ has no attack, then Spec has no attack.*

Proof sketch. We proceed by contraposition and assume that there exists an attack trace on the composed protocol. Our goal is to prove the existence of an attack trace on a projection of the protocol, i.e., to show that there is an attack on one protocol composed with the interface of the other protocol.

The first step is to show that we can assume the intruder chooses homogeneous recipes without loss of generality, i.e., whenever a protocol is receiving a message, then the intruder uses labels from that same protocol (or declassified messages). Since we only consider well-typed instantiations, we can use our result of Lemma VI.1 to show that whenever a process is receiving a message, then the intruder can use a homogeneous recipe to produce that message. This step is actually quite difficult because we need to show that the change to homogeneous recipe does not significantly alter the reached state.

As a second step, we consider transactions that went into an else branch for a conditional that is not marked $\star$, i.e., that is protocol specific. Here the problem is that such a branching is not possible in the projection to the other protocol, but we can use our requirement that the else branch for protocol-specific conditionals must be stop. Thus, the transaction can only have the effect to show that the respective condition is false. So either that gives an attack on privacy, or we can remove this transaction from the trace.

Finally, we make a case distinction on the kind of attacks: if some assertion was broken or some secret was leaked, then the projected attack trace also leads to a broken assertion or secret leaked. If the attack is instead a violation of privacy, i.e., there is a model of privacy variables allowed by the payload α that the intruder can actually rule out given their knowledge, then we show that either we find a "simpler" attack that is just a secret leaked or we also have a violation of privacy from the projected trace. $\lhd$

VII. Related and Future Work

There is a number of works that show compositionality for standard trace-based properties in the symbolic (Dolev-Yao-style) model [19], [20], [21], [22], [11]. The closest to ours is the work of Hess et al. [11] which shows a compositionality result for stateful protocols. Here the state is represented by a family of sets of messages, e.g., a set of key registered as valid at a server. These sets can be shared between the component protocols. This is similar to our work where we have instead a family of memory cells which can only hold one message and also in our case these can be shared between the components. This is because previous work on (α, β)-privacy [15] uses such memory cells, but we want to investigate in future work whether we can also support sets. From [11] we adopted the idea that the protocols do not need to be completely disjoint but can share a set of messages that either are public or initially secret.

As already mentioned there are very few works on symbolic compositionality for privacy-type goals. The standard model for privacy goals is based on a notion of indistinguishability, i.e., the intruder cannot tell two processes apart. In a nutshell, for every trace that one process can exhibit, one has to show that the other process can have the a similar trace in the sense that the intruder frames are indistinguishable. This is particularly challenging in case of conditionals, because the intruder may in general not be able to tell whether the condition was true and thus the then or the else branch was the case. Note that in (α, β)-privacy this is handled by maintaining a number of possibilities $\mathcal{P}$: the rule Conditional in Table I says that a possibility with a conditional statement is split into two possibilities for the positive and negative branch. Moreover, the rule Eliminate says that the intruder can then discard possibilities that are not compatible with observations, and keeps all that are. As this is a major difficulty in approaches for unbounded session verification of trace equivalence, a restriction is often considered: bi-processes and diff-equivalence [23], [24]. Here, the two processes that should be indistinguishable for the intruder differ only in the concrete terms which come in two variants, and one requires that all conditionals are satisfied for both or for neither variant, so that it is always either the then or the else case. This allows to easily turn the problem into a trace problem that is much easier to deal with.

The most advanced work on compositionality for trace-equivalence that we are aware of [21] also uses the restriction to bi-processes and diff-equivalence. (They do also have a result for the standard trace properties that does not need this restriction, of course.) They show results for both the parallel case and a sequential case where one protocol generates a key and another protocol uses it. Our compositionality result is significantly more general than this: we do not need the restriction to bi-processes, i.e., we consider the standard model of (α, β)-privacy where all branches of conditionals are possible and maintained. For what concerns different types of composition, the generality of our result blurs the boundaries between concepts like parallel and sequential composition: in our parallel composition the components can actually communicate with each other, this just has to be part of the interface, either via memory cells and via shared messages (and declassificaton). Communication via memory cells actually allows for sequential composition. Moreover, we can employ components as subprotocols that can be invoked by other components.

We believe that the reason we can make such a generalization with complicated but still manageable proofs is due to the technically more simple concepts of (α, β)-privacy: we deal with a simple reachability problem where each state carries enough information to model the intruder reasoning, namely which models of the relations and privacy variables are still compatible with all observations (and messages sent by the intruder). This allows us to show that any trace of the composed system, when projected to the steps of one component and the shared ($\star$-marked) steps, still works with that component alone; thus the security of every component implies the security of the composition.

There are however also some aspects where [21] is more general. First, they allow for one component to use Diffie-Hellman (however it cannot be shared between components), which is valuable for key-exchange protocols. We plan to investigate as future work how to support more algebraic properties. Moreover, we require a typed model while the requirements of [21] seem less restrictive—except for Diffie-Hellman where they essentially have a strictly typed model.

One may wonder how fair the comparison between privacy in trace-equivalence models and (α, β)-privacy actually is. [15] gives an argument how trace-equivalence properties can be translated into (α, β)-privacy problems and vice-versa. Nonetheless there are several substantial differences in the models. (α, β)-privacy assumes that the intruder can always see which transaction is executed, and may be just unclear about the concrete values like privacy variables, and which branch of a conditional is taken. In contrast, the trace-equivalence approaches are focused on a trace of messages that the intruder sent or received, thus the intruder is a priori unable to tell which position in the considered process has produced a particular output, and where a particular input was received. Thus, the intruder gets a little more information in (α, β)-privacy than typically in the model in other approaches; this can actually be often justified in practice since the intruder can know which inputs and outputs belong to the same session, and they are a substantial simplification for automated and compositional reasoning.

References

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

[2] 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.

[3] S. Doghmi, J. Guttman, and F. J. Thayer, "Searching for shapes in cryptographic protocols," in *TACAS 2007*, ser. LNCS, vol. 4424. Springer, 2007, pp. 523–537.

[4] J. Guttman and F. J. Thayer, "Protocol independence through disjoint encryption," in *CSFW 2000*. IEEE, 2000, pp. 24–34.

[5] V. Cortier and S. Delaune, "Safely composing security protocols," *Form Methods Syst Des*, vol. 34, no. 1, pp. 1–36, 2009.

[6] S. Ciobâca and V. Cortier, "Protocol composition for arbitrary primitives," in *CSF 2010*. IEEE, 2010, pp. 322–336.

[7] M. Arapinis and M. Duflot, "Bounding messages for free in security protocols – extension to various security properties," *Inf Comput*, vol. 239, pp. 182–215, 2014.

[8] A. Hess and S. Mödersheim, "Formalizing and proving a typing result for security protocols in Isabelle/HOL," in *CSF 2017*. IEEE, 2017, pp. 451–463.

[9] ——, "A typing result for stateful protocols," in *CSF 2018*. IEEE, 2018, pp. 374–388.

[10] R. Chrétien, V. Cortier, A. Dallon, and S. Delaune, "Typing messages for free in security protocols," *ACM Trans. Comput. Logic*, vol. 21, no. 1, pp. 1–52, 2020.

[11] A. Hess, S. Mödersheim, and A. Brucker, "Stateful protocol composition in Isabelle/HOL," *ACM Trans. Priv. Secur.*, vol. 26, no. 3, pp. 1–36, 2023.

[12] L. Fernet, S. Mödersheim, and L. Viganò, "A typing result for alpha-beta privacy," DTU Compute; KCL Informatics, Tech. Rep., 2024. [Online]. Available: https://people.compute.dtu.dk/samo

[13] R. Needham and M. Schroeder, "Using encryption for authentication in large networks of computers," *Commun. ACM*, vol. 21, no. 12, pp. 993–999, 1978.

[14] G. Lowe, "An attack on the needham-schroeder public-key authentication protocol," *Inf Process Lett*, vol. 56, no. 3, pp. 131–133, 1995.

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

[16] 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

[17] L. Fernet, S. Mödersheim, and L. Viganò, "A decision procedure for alpha-beta privacy for a bounded number of transitions," in *CSF 2024 (to appear)*. IEEE, 2024, extended version at https://people.compute.dtu.dk/samo/alphabeta.

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

[19] J. Guttman, "Cryptographic protocol composition via the authentication tests," in *FOSSACS 2009*, ser. LNCS, vol. 5504. Springer, 2009, pp. 303–317.

[20] C. Chevalier, S. Delaune, S. Kremer, and M. Ryan, "Composition of password-based protocols," *Form Methods Syst Des*, vol. 43, no. 3, pp. 369–413, 2013.

[21] M. Arapinis, V. Cheval, and S. Delaune, "Composing security protocols: From confidentiality to privacy," in *POST 2015*, ser. LNCS, vol. 9036. Springer, 2015, pp. 324–343.

[22] V. Cheval, V. Cortier, and B. Warinschi, "Secure composition of PKIs with public key protocols," in *CSF 2017*. IEEE, 2017, pp. 144–158.

[23] 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.

[24] 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.

Appendix

A. Proofs

1) Compositionality on the Frame Level:

Definition A.1. *A* shorthand $i : l \leftarrow r$ *consists of a label l associated with a recipe r and marked with i. We extend the notion of recipes so that they can use labels from shorthands: a recipe containing label l from a shorthand $l \leftarrow r$ produces the same message as the recipe where l is replaced with r.*

Let F be a ground frame. A frame F' is an extension of F with shorthands *iff $dom(F) \subseteq dom(F')$ and for every label $l \in dom(F') \setminus dom(F)$ such that $F' = F_1.i : l \leftarrow r.F_2$, we have $labels(r) \subseteq dom(F_1)$, $F_1(r) \in GSMP_i$ and every label in r is marked with i or $\star$.*

We extend the notion of comparable frames as well: two frames are comparable iff they have the same domain, the same shorthands and for every label, the frames agree on its mark.

The notion of shorthands does not impact the static equivalence between frames.

Lemma A.1. *Let F_1, F_2 be ground frames and let F_1', F_2' be extensions of the respective frames with the same shorthands. Then $F_1 \sim F_2$ iff $F_1' \sim F_2'$.*

Proof. Case $F_1 \not\sim F_2$: Then there exists a witness (r_1, r_2) that distinguishes the frames, which is then also a witness that distinguishes F_1' and F_2'.

Case $F_1' \not\sim F_2'$: Then there exists a witness (r_1, r_2) that distinguishes the frames. Any label l that comes from a shorthand $l \leftarrow r$ can be replaced with r while preserving the fact that the pair of recipes if a witness. Then there exists a witness that distinguishes F_1 and F_2. □

In the rest of this section, we only consider ground frames.

Lemma A.2. *Let F_1, F_2 be leakage-free comparable frames, $i \in \{1, 2, \star\}$ and r be a destructor-free recipe such that $F_1(r), F_2(r) \in GSMP_i$. Then at least one of the following is true:*

1) *There exist a destructor-free homogeneous recipe r' and frames F_1', F_2' extensions of F_1, F_2 with the same shorthands such that $F_1(r) \approx F_1'(r')$ and $F_2(r) \approx F_2'(r')$.*
2) *There exists a homogeneous witness against static equivalence of F_1 and F_2.*

Proof. Assume that $i = 1$ and that there is a label l in r marked with 2 (the cases that $i = 2$ or $i = \star$ are handled similarly). Let $t_1 = F_1(l)$ and $t_2 = F_2(l)$. Since l is marked with 2, $t_1, t_2 \in \mathit{GSMP}_2$. Since r is destructor-free, t_1 is a subterm of $F_1(r)$ and thus $t_1 \in \mathcal{T}_{pub} \cup \mathit{Secrets}$.

- Case $t_1 \in \mathcal{T}_{pub}$: If $t_1 = t_2$, then we continue with the recipe $r[l \mapsto t_1]$. Otherwise, (l, t_1) is a witness and thus Item 2 is true, since this pair of recipes is homogeneous.
- Case $t_1 \in \mathit{Secrets}$: Since F_1 is leakage-free and $F_1|_2(l) = t_1$, $t_1 \notin \mathit{Secrets} \setminus \mathit{declassified}(F_1)$. Thus $t_1 \in \mathit{declassified}(F_1)$, which means by definition of $\mathit{declassified}(F_1)$ that there exists a recipe r' such that $F_1(r') \approx t_1$ and all labels in r' are marked with $\star$. If (l, r') is a witness, then Item 2 is true since this pair of recipes is homogeneous. Otherwise, we add the shorthand $\star : l' \leftarrow r'$ in both frames. Then we have $F_1(l') \approx t_1$ and $F_2(l') \approx t_2$, so we continue with the recipe $r[l \mapsto l']$.

By repeating these steps, we can successively replace all labels in the recipe r with equivalent labels (possibly using shorthands) so that the recipe becomes homogeneous, or we may find a homogeneous witness during some step. □

Lemma A.3. *Let F be a leakage-free frame, $i \in \{1, 2, \star\}$ and r be a destructor-free recipe such that $F(r) \in \mathit{GSMP}_i$. Then there exists a recipe r' such that $F(r) \approx F(r')$ and every label in r' is marked with i or $\star$.*

Proof. By Lemma A.2, when considering twice the same frame, there exist an extension F' of F with shorthands and a destructor-free homogeneous recipe r'' such that $F(r) \approx F'(r'')$. Then we obtain the recipe r' from r'' by replacing any label from a shorthand with the recipe it is assigned to (and removing the shorthands preserves homogeneity, since the recipe in a shorthand is homogeneous w.r.t. the mark of the shorthand). □

Lemma VI.1. *Let F be a leakage-free frame, $i \in \{1, 2, \star\}$ and r be a recipe such that $F(r) \in \mathit{GSMP}_i$. Then there exists a recipe r' such that $F(r) \approx F(r')$ and every label in r' is marked with i or $\star$.*

Proof. If r is destructor-free, then the lemma holds by Lemma A.3. Otherwise, we consider an inner-most destructor in r. We have $r = r_0[d(r_k, r_t)]$ where r_k and r_t are destructor-free and $r_0[\cdot]$ is a recipe context. (A recipe context is a recipe with a hole, and the hole is filled when the context is applied to a recipe.) We have the rewrite rule $d(k, c(k', X_1, \ldots, X_n)) \to X_j$, where $j \in \{1, \ldots, n\}$. If $F(d(r_k, r_t)) \approx \mathtt{ff}$, then we continue with the recipe $r_0[\mathtt{ff}]$. Otherwise:

- Case $r_t = c(r'_k, r'_1, \ldots, r'_n)$: The decryption succeeds, so we continue with the recipe $r_0[r'_j]$.
- Case $r_t = l \in \mathit{dom}(F)$: The decryption succeeds. By definition of GSMP, we have $F(r_k) \in \mathit{GSMP}_i$. By Lemma A.3, there exists a homogeneous recipe r'_k such that $F(r_k) \approx F(r'_k)$. Then we add the shorthand $l' \leftarrow d(r'_k, l)$ with the same mark as for l. Note that every label in $d(r'_k, l)$ has the same mark as l or $\star$, so the shorthand is well-defined. Then we continue with the recipe $r_0[l']$.

By repeating these steps, we can successively replace all destructor applications in the recipe with equivalent labels (possibly using shorthands) so that we can get an extension F' of F with shorthands and a destructor-free recipe r_1 such that $F(r) \approx F'(r_1)$. By Lemma A.3, there exists a recipe r_2 such that $F'(r_1) \approx F'(r_2)$ and every label in r_2 is marked with i or $\star$. Then we obtain the recipe r' from r_2 by replacing any label from a shorthand with the recipe it is assigned to. □

Lemma A.4. *Let F_1, F_2 be leakage-free comparable frames. If there exists a destructor-free witness against static equivalence of F_1 and F_2, then there exists a homogeneous witness.*

Proof. For recipe r_1, r_2, let $\mathit{labels}(r_1, r_2)$ denote the set of labels occurring in r_1 or r_2 that are *not* marked with $\star$. Given a pair of recipes (r_1, r_2), we define the weight of the pair as:

$$w_{F_1,F_2}(r_1, r_2) = \begin{cases} (\#\mathit{labels}(r_1, r_2), \mathit{size}(F_1(r_1))) \\ \quad \text{if } F_1(r_1) \approx F_1(r_2) \text{ and } F_2(r_1) \not\approx F_2(r_2) \\ (\#\mathit{labels}(r_1, r_2), \mathit{size}(F_2(r_1))) \\ \quad \text{if } F_1(r_1) \not\approx F_1(r_2) \text{ and } F_2(r_1) \approx F_2(r_2) \end{cases}$$

and $w_{F_1,F_2}(r_1, r_2)$ is undefined otherwise (i.e., the weight is only defined for witnesses).

Assume that there exists a destructor-free witness (r_1, r_2) against static equivalence of F_1 and F_2. If both recipes are composed, then they must have the same constructor at the top-level, because they produce the same term in exactly one frame: $r_1 = c(r_1^1, \ldots, r_1^n)$ and $r_2 = c(r_2^1, \ldots, r_2^n)$. Then at least one of the (r_1^i, r_2^i) must already be a witness. We can thus move to a smaller witness: if $\mathit{labels}(r_1^1, r_2^1) \subsetneq \mathit{labels}(r_1, r_2)$, then the first component of the weight decreases; otherwise, the second component of the weight decreases (the recipes produce a strict subterm of the original message). This is well-founded: at some point, at least one of the recipes is no longer composed and thus a label. Therefore there exists a witness (l, r) such that l is a label and r is destructor-free. Assume that:

- $F_1(l) \approx F_1(r)$ and $F_2(l) \not\approx F_2(r)$.
- l is marked with 1 and r contains a label l' marked with 2.

Note that these assumptions are without loss of generality, because we can just rename the frames and the case that l is marked with 2 or $\star$ is handled in a similar way as below. Since l is marked with 1, $F_1(l), F_2(l) \in \mathit{GSMP}_1$. Let $t = F_1(l')$. Since l' is marked with 2, $t \in \mathit{GSMP}_2$. Since r is destructor-free, t is a subterm of $F_1(l)$ and thus $t \in \mathcal{T}_{pub} \cup \mathit{Secrets}$.

- Case $t \in \mathcal{T}_{pub}$: If (l', t) is a witness, then it is a homogeneous one and the lemma holds. Otherwise, we continue with the witness $(l, r[l' \mapsto t])$. This witness is smaller because we are removing a label not marked with $\star$, so the first component of the weight decreases.
- Case $t \in \mathit{Secrets}$: Since F_1 is leakage-free and $F_1|_2(l') = t$, $t \notin \mathit{Secrets} \setminus \mathit{declassified}(F_1)$. Thus

$t \in \mathit{declassified}(F_1)$, which means by definition of $\mathit{declassified}(F_1)$ that there exists a recipe r' such that $F_1(r') \approx t$ and all labels in r' are marked with $\star$. If (l', r') is a witness, then it is a homogeneous one and the lemma holds. Otherwise, we add the shorthand $\star : l'' \leftarrow r'$ in both frames. Then we have, $F_1(l') \approx F_1(l'')$ and $F_2(l') \approx F_2(l'')$, so continue with the witness $(l, r[l' \mapsto l''])$. This witness is smaller because we are removing a label not marked with $\star$ and the label we introduce is a shorthand marked with $\star$, so the first component of the weight decreases.

By repeating these steps, we can successively replace all labels in the recipe r with equivalent labels (possibly using shorthands) so that the witness becomes homogeneous, and at every step we decrease the weight of the witness, or we may find a homogeneous witness during some step. □

Lemma A.5. *Let F_1, F_2 be leakage-free comparable frames. If there exists a witness against static equivalence of F_1 and F_2, then there exists a homogeneous witness.*

Proof. Assume that there exists a witness (r_1, r_2) against static equivalence of F_1 and F_2. If the witness is destructor-free, then the lemma holds by Lemma A.4. Otherwise, we assume w.l.o.g. that r_1 is not destructor-free. We consider an inner-most destructor in r_1. We have $r_1 = r[d(r_k, r_t)]$ where r_k and r_t are destructor-free. We have the rewrite rule $d(k, c(k', X_1, \dots, X_n)) \to X_j$, where $j \in \{1, \dots, n\}$, and either $k = k'$ or there exists some public function f such that $k \approx f(k')$ or $k' \approx f(k)$. If $F_1(d(r_k, r_t)) \approx \mathtt{ff}$ and $F_2(d(r_k, r_t)) \approx \mathtt{ff}$, then $(r[\mathtt{ff}], r_2)$ is a witness. Otherwise:

- Case $r_t = c(r'_k, r'_1, \dots, r'_n)$:
  - Case $k = k'$: If (r_k, r'_k) is a witness, then by Lemma A.4 there exists a homogeneous witness and the lemma holds. Otherwise, the decryption succeeds in both frames so we continue with the witness $(r[r'_j], r_2)$.
  - Case $k \approx f(k')$: If $(r_k, f(r'_k))$ is a witness, then we continue with that simpler witness (it produces a strict subterm of the original witness). Otherwise, the decryption succeeds in both frames so we continue with the witness $(r[r'_j], r_2)$.
  - Case $k' \approx f(k)$: Similar to the previous case.
- Case $r_t = l \in \mathit{dom}(F_1)$: By Lemma A.2, we may find a homogeneous witness so that the lemma holds, or there exist a destructor-free homogeneous recipe r'_k and frames F'_1, F'_2 extensions of F_1, F_2 with the same shorthands such that $F_1(r_k) \approx F'_1(r'_k)$ and $F_2(r_k) \approx F'_2(r'_k)$. If $(d(r_k, l), \mathtt{ff})$ is a witness, then $(d(r'_k, l), \mathtt{ff})$ is also a witness and since it is homogeneous, the lemma holds. Otherwise, the decryption succeeds in both frames and we add the shorthand $i : l' \leftarrow d(r'_k, l)$ where i is the same mark as for l. Then we continue with the witness $(r[l'], r_2)$.

By repeating these steps, we can successively replace all destructor applications in the witness with equivalent labels (possibly using shorthands) so that the witness becomes homogeneous, or we may find a homogeneous witness during some step. □

Theorem VI.1. *Let F_1, F_2 be leakage-free comparable frames. If for every $i \in \{1, 2\}$, $F_1|_i \sim F_2|_i$, then $F_1 \sim F_2$.*

Proof. We proceed by contraposition. Assume that $F_1 \not\sim F_2$. By Lemma A.5, there exists a homogeneous witness (r_1, r_2) against static equivalence of F_1 and F_2 and thus there exists $i \in \{1, 2\}$ such that all labels in r_1 and r_2 are marked with i or $\star$. Then (r_1, r_2) is a witness against static equivalence of $F_1|_i$ and $F_2|_i$. □

2) Equivalence Between States:

Definition A.2. *Let $S = (\alpha, \gamma, \mathcal{P}, \_, \mathit{flag}, \mathit{TID})$ where $\mathcal{P} = \{(R_1, \phi_1, \mathit{struct}_1, \delta_1), \dots, (R_n, \phi_n, \mathit{struct}_n, \delta_n)\}$ and let $S' = (\alpha', \gamma', \mathcal{P}', \_, \mathit{flag}', \mathit{TID}')$ where $\mathcal{P}' = \{(R'_1, \phi'_1, \mathit{struct}'_1, \delta'_1), \dots, (R'_n, \phi'_n, \mathit{struct}'_n, \delta'_n)\}$.*

S and S' are equivalent *iff*

- *$\alpha \equiv \alpha'$;*
- *$\gamma \equiv \gamma'$;*
- *$\mathit{flag} = \mathit{flag}'$;*
- *$\mathit{TID} = \mathit{TID}'$;*
- *For every $i \in \{1, \dots, n\}$, $\beta(S) \wedge \phi_i \equiv \beta(S') \wedge \phi'_i$ and there exists a substitution σ_i such that $\sigma_i(R_i) = \sigma_i(R'_i)$, $\beta(S) \wedge \phi_i \models \sigma_i$, $\sigma_i(\mathit{struct}_i) = \sigma_i(\mathit{struct}'_i)$ and $\sigma_i(\delta_i) = \sigma_i(\delta'_i)$.*

The notation $\mathit{mgu}(s \doteq t)$ returns the most general unifier between s and t. As a slight abuse of notation, we consider a substitution $[X_1 \mapsto t_1, \dots, X_n \mapsto t_n]$ as the conjunction of equalities $X_1 \doteq t_1 \wedge \dots \wedge X_n \doteq t_n$ and thus we also use $\mathit{mgu}(\sigma \wedge \tau)$ to denote the mgu that unifies all equalities in σ and τ.

Lemma A.6. *The relation in Definition A.2 is an equivalence relation.*

Proof. Straightforward, it suffices to expand the definitions. □

Two equivalent states represent the same intruder knowledge.

Lemma A.7. *Let S, S' be equivalent states. Then $\beta(S) \equiv \beta(S')$.*

Proof. We only show $\beta(S) \models \beta(S')$, the other direction follows by symmetry. Let $\mathcal{I} \models \beta(S)$. Then there exists a possibility with condition ϕ in S such that $\mathcal{I} \models \phi$. Since S and S' are equivalent, there exists a possibility with condition ϕ' in S' such that $\beta(S) \wedge \phi \equiv \beta(S') \wedge \phi'$. Then $\mathcal{I} \models \beta(S') \wedge \phi'$ and thus $\mathcal{I} \models \beta(S')$. □

Lemma A.8. *Let S_1, S'_1 be equivalent states. Then for every state S_2 such that $S_1 \to S_2$, there exists a state S'_2 such that $S'_1 \to S'_2$ and S_2 and S'_2 are equivalent.*

Proof. By definition of equivalence, S_1 and S'_1 have the same payload α, truth formula γ, assertion flag flag and thread ID

TID. Moreover, for every possibility $(R_i, \phi_i, \mathit{struct}_i, \delta_i)$ in S_1, there is a corresponding possibility $(R'_i, \phi'_i, \mathit{struct}'_i, \delta'_i)$ in S'_1 and a substitution σ_i such that $\sigma_i(R_i) = \sigma_i(R'_i)$, $\beta(S_1) \wedge \phi_i \models \sigma_i$, $\sigma_i(\mathit{struct}_i) = \sigma_i(\mathit{struct}'_i)$ and $\sigma_i(\delta_i) = \sigma_i(\delta'_i)$. Let S_2 be a state such that $S_1 \to S_2$. We proceed by case distinction of the rule applied in the transition $S_1 \to S_2$. Note that for every step in a process in S_1, there is a corresponding step in a process in S'_1, where only the instantiations of privacy variables may differ. Thus there exists a state S'_2 such that $S'_1 \to S'_2$. We show that every transition preserves the equivalence.

- Choice: The truth formula is extended in the same way in S_2 and S'_2.
- Receive: For the message received $\mathsf{rcv}(t)$, the intruder uses a recipe r, then the variables bound in the linear term t are substituted in the rest of the process according to what the recipe r produces in the respective frame. The messages $\mathit{struct}_i(r)$ and $\mathit{struct}'_i(r)$ may be different, therefore the bound variables may be substituted differently in the rest of the process, but we have that $\sigma_i(\mathit{struct}_i(r)) = \sigma_i(\mathit{struct}'_i(r))$. Thus the messages occurring in the rest of the process are still related by the substitutions σ_i in S_2, S'_2.
- Let: For a step $\mathsf{let}\ X = t$ in S_1, there is a corresponding step $\mathsf{let}\ X = t'$ in S'_1 and a substitution σ_i such that $\sigma_i(t) = \sigma_i(t')$. The variable X may be substituted differently but the messages in the rest of the process are still related by the σ_i in S_2, S'_2.
- Cell read: For the possibility with memory δ_i that is doing a cell read in S_1, there is a corresponding possibility with memory δ'_i that is also doing a cell read and a substitution σ_i such that $\sigma_i(\delta_i) = \sigma_i(\delta'_i)$. The cell read introduces conditional statements with the memory updates and the variable bound to the cell read is substituted in each branch according to the respective memory update. The values read from δ_i and δ'_i may be different, therefore the bound variable may be substituted differently in the rest of the process, but the messages occurring in the rest of the process are still related by the σ_i in S_2, S'_2.
- Cell write: For a step $\mathsf{cell}(s) := t$ in S_1, there is a corresponding step $\mathsf{cell}(s') := t'$ in S'_1 and a substitution σ_i such that $\sigma_i(s) = \sigma_i(s')$ and $\sigma_i(t) = \sigma_i(t')$. The sequences of memory updates are still related by the σ_i in S_2, S'_2.
- Conditional: For a statement branching on condition ψ in the possibility with condition ϕ_i in S_1, there is a corresponding branching on condition ψ' in a possibility with condition ϕ'_i in S'_1 and a substitution σ_i such that $\psi \wedge \sigma_i \equiv \psi' \wedge \sigma_i$ and $\beta(S_1) \wedge \phi_i \equiv \beta(S'_1) \wedge \phi'_i$. The possibility can be split in two in each state. Then we have that $\gamma \models \phi_i \wedge \psi$ iff $\gamma \models \phi'_i \wedge \psi'$, and $\gamma \models \psi_i \wedge \neg\psi$ iff $\gamma \models \phi'_i \wedge \neg\psi'$.
- Release: For a step $\star\ \psi$ in the possibility with condition ϕ_i in S_1 such that $\gamma \models \phi_i$, there is the same step $\star\ \psi$ in a possibility with condition ϕ'_i in S'_1 such that $\gamma \models \phi'_i$. Thus the new payload in S_2 and S'_2 is the same. Note that we have the same formula ψ in both cases because the variables in ψ can only be privacy variables chosen in previous transactions following Definition V.4. If we did not make this restriction, then the formulas released would be related by the σ_i but the payload might be different and thus we would not obtain equivalent states.
- Assert: For a step $\mathsf{assert}(\psi)$ in the possibility with condition ϕ_i in S_1 such that $\gamma \models \phi_i$, there is a corresponding step $\mathsf{assert}(\psi')$ in a possibility with condition ϕ'_i in S'_1 such that $\gamma \models \phi'_i$ and a substitution σ_i such that $\psi \wedge \sigma_i \equiv \psi' \wedge \sigma_i$. Moreover, $\gamma \models \sigma_i$ because $\beta(S_1) \wedge \phi \models \sigma_i$, γ is consistent with $\beta(S_1)$ and $\gamma \models \phi$. Thus $\gamma \models \psi$ iff $\gamma \models \psi'$ and the assertion flag is the same in S_2 and S'_2.
- Stop or Milestone: For every possibility that starts with stop in S_1, there is a corresponding possibility that also starts with stop in S'_1 (and similarly for 0 instead of stop). Then corresponding possibilities are discarded in the same way in the transitions to S_2, S'_2.
- Send: For every step $\mathsf{snd}(t)$ in S_1, there is a corresponding step $\mathsf{snd}(t')$ in S' and a substitution σ_i such that $\sigma_i(t) = \sigma_i(t')$. Thus the frames in S_2 and S'_2 are still related by the σ_i in S_2, S'_2. Moreover, for every possibility in S_1 that is discarded (because it starts with stop or 0) in the transition to S_2, there is a possibility in S'_1 that is also discarded in the transition to S'_2.
- Eliminate: For a possibility with condition ϕ_i in S_1 such that $\beta(S_1) \not\models \phi_i$, there is a possibility with condition ϕ'_i in S'_1 such that $\beta(S'_1) \not\models \phi'_i$.
- Next: The same thread ID is set in S_2 and S'_2. □

3) Compositionality on the State Level: Given a state S, we denote with $\mathit{concr}(S)$ the concrete frame in that state.

Lemma A.9. *Let $(P_1; \ldots; P_n, \gamma, \rho)$ be an attack trace and $S_0, \ldots, S_n$ be the milestones such that for every $j \in \{1, \ldots, n\}$, executing P_j, starting from S_{j-1}, leads to S_j (following the truth γ and using the recipes in ρ).*

Then there exists ρ' such that $(P_1; \ldots; P_n, \gamma, \rho')$ is an attack trace leading to milestones $S'_0, \ldots, S'_n$ where S_j and S'_j are equivalent (for $j \in \{0, \ldots, n\}$) and for every step $i : \mathsf{rcv}(t)$ during the execution of the transactions, where $i \in \{1, 2, \star\}$, every label in the recipe given by ρ' to produce t is marked with i or $\star$.

Proof. Let $j \in \{1, \ldots, n\}$ and $\mathit{concr} = \mathit{concr}(S_{j-1})$. Consider a step $i : \mathsf{rcv}(t)$ ($i \in \{1, 2, \star\}$) in the transaction P_j, let r be the recipe given by ρ for this message received and $t' = \mathit{concr}(r)$. We have $t' \in \mathit{GSMP}_i$, because we consider only well-typed instantiations and by definition GSMP_i contains all the well-typed instances of t. By Lemma VI.1, there exists a homogeneous recipe r' such that $\mathit{concr}(r') \approx t'$ and every label in r' is marked with i or $\star$. Thus we can consider ρ' that is the same as ρ except that it uses r' instead of r for this message received.

Thus we know that the intruder is able to use a homogeneous recipe to produce the same message. However, we now need to argue that using a homogeneous r' instead of r is correct w.r.t. the symbolic execution. Indeed, the two recipes produce the same message in concr, i.e., the concrete messages are equal, but changing recipes can make a difference for the instantiation of privacy variables. Let $(\_, \gamma, \mathcal{P}, \_, \_, \_) = S_{j-1}$ where $\mathcal{P} = \{\underline{(0; R_1, \phi_1, \mathit{struct}_1, \delta_1)}, \ldots, (0; R_m, \phi_m, \mathit{struct}_m, \delta_m)\}$. The underlined possibility is what really is the case, i.e., $\gamma \models \phi_1$ and $\mathit{concr} = \gamma(\mathit{struct}_1)$. For every $k \in \{1, \ldots, m\}$, the intruder knows that $\beta(S) \wedge \phi_k \models \mathit{concr} \sim \mathit{struct}_k$. Since r and r' produce the same in concr, the intruder also knows that $\beta(S) \wedge \phi_k \models \mathit{struct}_k(r) \doteq \mathit{struct}_k(r')$. Let $\sigma_k = \mathit{mgu}(\mathit{struct}_k(r) \doteq \mathit{struct}_k(r'))$. The state S'_{j-1} obtained from S_{j-1} by applying the $\sigma_1, \ldots, \sigma_m$ to the respective possibilities is equivalent to S_{j-1}. Thus by Lemma A.8, the states reached when using a homogeneous r' instead of r leads are equivalent.

This argument holds for every message received in the transaction P_j and for every transaction in the trace. □

Every possibility that the intruder has not ruled out corresponds to some concrete execution.

Lemma A.10. *Let $S = (\alpha, \gamma, \mathcal{P}, \rho, \_, \_)$ be a state reached with trace $(P_1; \ldots; P_n, \gamma, \rho)$. Then for every possibility $(\_, \phi, \mathit{struct}, \_) \in \mathcal{P}$ and interpretation $\gamma' \models \beta(S) \wedge \phi$, $(P_1; \ldots; P_n, \gamma', \rho)$ is a trace and it leads to a state S' such that $\mathit{concr}(S') = \gamma'(\mathit{struct})$.*

Proof. We consider the transitions, in the symbolic execution, that depend on the truth formula.

- Release: The formulas released in the possibility underlined by γ' are added to the payload α' of S' while the releases in other possibilities are ignored. Thus the payload in S and S' may be different, but the possibilities contain the same frames in both states.
- Stop, Send or Milestone: Since the possibility with ϕ, struct remains in S, i.e., it was not ruled out by the intruder, its process was stopping, sending or reaching a milestone at the same time as the possibility underlined by γ, so the same transitions can be taken when considering the truth γ'.
- Assert: The assertions in the possibility underlined by γ' are checked while the assertions in other possibilities are ignored. Thus the assertion flag in S and S' may be different, but the possibilities contain the same frames in both states.
- Eliminate: The intruder knowledge in the intermediate states to reach S and S' may be different, so the eliminated possibilities may be different, but the intruder did not rule out the possibility with ϕ, struct in S, so also in S' there is a possibility with the same frame.

The other transitions in the symbolic execution do not depend on the truth formula, thus when executing the transactions following truth γ', there is a possibility with frame struct considered by the intruder and it is actually the underlined one, so $\mathit{concr}(S') = \gamma'(\mathit{struct})$. Note that by well-formedness of states, we have $\alpha' \models \bigvee_{\gamma_0 \in \Gamma_0} \gamma_0$, where α' is the payload in S'. Thus even if γ and γ' do not agree on the relations in Σ_0, there exists some $\gamma_0 \in \Gamma_0$ such that $\gamma' \models \gamma_0$ and thus $(P_1; \ldots; P_n, \gamma', \rho)$ is a trace and S' is a reachable state. □

Theorem VI.2. *If for every $i \in \{1, 2\}$, $\mathit{Spec}|_i$ has no attack, then Spec has no attack.*

Proof. We proceed by contraposition. We assume that the composed protocol Spec has an attack and we show that one of the projections to a component, i.e., $\mathit{Spec}|_1$ or $\mathit{Spec}|_2$, also has an attack. Let $(P_1; \ldots; P_n, \gamma, \rho)$ be an attack trace. The symbolic execution of the transactions, following recipes in ρ and truth formula γ, defines the milestones $S_0, \ldots, S_n$ such that for every $j \in \{1, \ldots, n\}$, executing P_j, starting from S_{j-1}, leads to S_j. Recall that by definition of attack traces, S_n is an attack state but all the milestones $S_0, \ldots, S_{n-1}$ do not have any attack.

By Lemma A.9, we can assume w.l.o.g. that in the attack trace, only homogeneous recipes are used. Now, consider the symbolic execution of a conditional statement $\mathsf{if}\ \phi\ \mathsf{then}\ P\ \mathsf{else}\ Q$ occurring in one of the $P_1, \ldots, P_{n-1}$ such that:

- The conditional statement is marked with a protocol-specific index, i.e., 1 or 2 but not $\star$.
- The underlined possibility, i.e., what really happened concretely, went into branch Q.

Since the branching is protocol-specific, we know that $Q = \mathsf{stop}$ (following Definition V.4), and the execution of stop is observable by the intruder. Moreover, a transaction cannot have any effects (e.g., writing to memory, sending a message or releasing a formula) before reaching stop because this step is in the center part of processes. Since reaching this stop step did not result in a privacy violation (as we are considering transactions before the last one), we can simplify the trace by removing the execution of this transaction and the successive ones that the stop would have filtered out. Thus, w.l.o.g. we can assume that whenever there is a protocol-specific branching occurring in the attack trace before the last transaction, the underlined possibility went into the first branch.

Let $j \in \{1, \ldots, n\}$. We will go over the different steps that can occur in P_j, and argue that for the appropriate projection, the steps are either present or have been soundly abstracted.

We know that the last transaction is the one that introduces an attack. Due to procedure call expansion, this transaction may contain steps from both protocols, i.e., some steps marked with 1 and other steps marked with 2: it may happen that the left part of the process has been specified by, say, protocol 1, then during the procedure call expansion a center process specified by protocol 2 was inserted. However, given a transaction, we can always uniquely identify the protocol that specifies the center process in that transaction (because procedure calls can only happen in the left part of the process). Let i be the index of the protocol that specifies the center process in the final transaction P_n. We show the existence of an attack by looking at the execution of the transactions $P_1|_i; \dots; P_n|_i$.

- Non-deterministic choice, release or nil process: The step remains in $P_j|_i$ because it is always present in any projection.
- Receive: If the message received is marked with i or $\star$, then it remains in $P_j|_i$ and we know that the intruder uses a recipe with labels marked with i or $\star$. Otherwise, since the role containing $P_j|_i$ is closed, the variables bound in that message are not used in the projection so the step can be skipped.
- Let statement or cell read: If the step is marked with i or $\star$, then it remains in $P_j|_i$. Otherwise, since the role containing $P_j|_i$ is closed, the variable bound by this step is not used in the projection so the step can be skipped.
- Cell write: If the cell write is marked with i or $\star$, then it remains in $P_j|_i$. Otherwise, the memory cell does not occur in $\mathit{Spec}|_i$ so the step can be skipped.
- Conditional statement: If the branching is marked with i or $\star$, then it remains in $P_j|_i$. Otherwise, we know that the underlined possibility went into the first branch (as justified above), so executing $P_j|_i$ (where this branching does not occur if $j < n$) preserves the attack.
- Assertion: If the assertion is marked with i or $\star$, then it remains in $P_j|_i$. Otherwise, the assertion was not present in the underlined possibility or was true (since the transactions before the last one do not lead to any attack) so the step can be skipped.
- Stop: This step is never reached in the transactions before the last one (as justified above). For the last transaction P_n, the stop remains in $P_n|_i$ because it is always present in any projection.
- Send: If the message sent is marked with i or $\star$, then it remains in $P_j|_i$ and the message is added to the frames with the same mark. Otherwise, the step can be skipped since in the projected transactions, the intruder only uses labels marked with i or $\star$.

Let ρ' be the same as ρ except that we remove the recipes corresponding to receive steps skipped when projecting to i.

We now consider different cases of attacks:

- If the flag in S_n is set to true: Then executing the trace $(P_1|_i; \dots; P_n|_i, \gamma, \rho')$ leads to a state with the same assertion that does not hold. Thus $\mathit{Spec}|_i$ has an attack.
- If the flag in S_n is set to false and S_n is not leakage-free: Then there exist $t \in \mathit{Secrets} \setminus \mathit{declassified}(\mathit{concr}(S_n))$, $i' \in \{1,2\}$ and r such that $\mathit{concr}(S_n)|_{i'}(r) \approx t$. Since it is the last transaction that leads to the attack, we have $i' = i$. Then executing the trace $(P_1|_i; \dots; P_n|_i, \gamma, \rho')$ leads to a state leaking the same secret t. Thus $\mathit{Spec}|_i$ is not leakage-free.
- Otherwise, i.e., the flag in S_n is set to false, S_n is leakage-free and does not satisfy privacy: Let $(\alpha, \gamma, \mathcal{P}, \rho, \mathsf{false}, \_) = S_n$ where $\mathcal{P} = \{(\_, \phi_1, \mathit{struct}_1, \_), \dots, (\_, \phi_m, \mathit{struct}_m, \_)\}$. Then we have that (α, β)-privacy does not hold, so there exists a model $\mathcal{I}$ such that $\mathcal{I} \models \alpha$ and $\mathcal{I} \not\models \beta$. Executing the trace $(P_1|_i; \dots; P_n|_i, \gamma, \rho')$ leads to a state $S' = (\alpha, \gamma, \mathcal{P}', \rho', \mathsf{false}, \_)$.
  - If there exists $j \in \{1, \dots, m\}$ such that $\mathcal{I} \models \phi_j$: Then $\mathcal{I} \not\models \mathit{concr}(S_n) \sim \mathit{struct}_j$, so $\mathit{concr}(S_n) \not\sim \mathcal{I}(\mathit{struct}_j)$. Since S_n is leakage-free, we know that concr is leakage-free. However, to apply our results on frames we need to have both frames leakage-free.
    - If $\mathcal{I}(\mathit{struct}_j)$ is leakage-free: Then by Theorem VI.1, there exists $i' \in \{1,2\}$ such that $\mathit{concr}(S_n)|_{i'} \not\sim \mathcal{I}(\mathit{struct}_j)|_{i'}$. Since it is the last transaction that leads to the attack, we have $i' = i$. Moreover, there exists a possibility $(\_, \phi'_j, \mathit{struct}'_j, \_) \in \mathcal{P}'$ such that $\phi_j \models \phi'_j$ and $\mathit{struct}_j|_i = \mathit{struct}'_j$. Note also that $\mathit{concr}(S') = \mathit{concr}(S_n)|_i$. Then $\mathcal{I} \models \alpha \wedge \phi'$ and $\mathit{concr}(S') \not\sim \mathcal{I}(\mathit{struct}')$, so S' does not satisfy privacy. Thus $\mathit{Spec}|_i$ does not satisfy privacy.
    - Otherwise: Then by Lemma A.10, there is a reachable state where the concrete frame is $\mathcal{I}(\mathit{struct}_j)$, i.e., there is a reachable state that leaks a secret and we can show (as done in a previous case) that one component of the protocol has an attack.
  - Otherwise, i.e., $\mathcal{I} \not\models \bigvee_{j=1}^{m} \phi_j$: The same branching occurs in the full trace $(P_1; \dots; P_n, \gamma, \rho)$ and the projected trace $(P_1|_i; \dots; P_n|_i, \gamma, \rho')$, except for protocol-specific branching that has been abstracted, but in this case the intruder observed in the full trace which branch was taken (as justified earlier). Moreover, if there are branches in the transactions $P_1, \dots, P_{n-1}$ that are distinguishable before the projection but not after (e.g., due to one branch sending a protocol-specific message), then the intruder can eliminate one of the branches in the projected trace because observing which branch was taken did not violate privacy, so even if the condition depended on privacy variables, then it was allowed by the payload. Thus $\beta(S') \models \bigvee_{j=1}^{m} \phi_j$. Then $\mathcal{I} \models \alpha$ and $\mathcal{I} \not\models \beta(S')$, so S' does not satisfy privacy and thus $\mathit{Spec}|_i$ does not satisfy privacy. □

B. Example of Composition with Simplified TLS

Our results are formalized with two protocols, but we can similarly handle the composition of three or more protocols: we could generalize all notions with a set of indices instead of just $\{1,2\}$, and we would then consider one protocol composed with the projection of all other protocols.

As a further example, we model the establishment of a fresh key between some agent and a trusted server. In Fig. 2, we model a simplified version of TLS, where the server is always the same trusted server s, and we also assume that agents already know the public key of that server so we do not model certificates. This protocol could for instance be used inside *lookup*, where the agent that makes the request is calling *tls_client* to get a fresh key instead of a long-term shared secret.

```
Procedure $tls\_client(C : \mathsf{agent})$
$\nu N_C : \mathsf{nonce}, PMS : \mathsf{pms}, R : \mathsf{nonce}.$
$\mathsf{snd}(N_C).$
$\mathsf{snd}(\mathsf{ox}(PMS)).$
$\mathsf{snd}(\mathsf{crypt}(\mathsf{pk}(\mathsf{s}), \mathsf{g}_1(PMS, N_C), R))$
;
$\mathsf{rcv}(N_S : \mathsf{nonce}).$
$\mathsf{rcv}(\mathsf{scrypt}(\mathsf{kdf}(PMS, N_C, N_S), \mathsf{g}_2(N_S), \_ : \mathsf{nonce})).$
$\mathsf{return}(\mathsf{kdf}(PMS, N_C, N_S))$
```

```
Procedure $tls\_server()$
$\mathsf{rcv}(N_C : \mathsf{nonce}).$
$\mathsf{rcv}(\mathsf{ox}(PMS : \mathsf{pms})).$
$\mathsf{rcv}(\mathsf{crypt}(\mathsf{pk}(\mathsf{s}), \mathsf{g}_1(PMS, N_C), \_ : \mathsf{nonce})).$
$\nu N_S : \mathsf{nonce}, R : \mathsf{nonce}.$
$\mathsf{snd}(N_S).$
$\mathsf{snd}(\mathsf{scrypt}(\mathsf{kdf}(PMS, N_C, N_S), \mathsf{g}_2(N_S), R)).$
$\mathsf{return}(\mathsf{kdf}(PMS, N_C, N_S))$
```

Fig. 2. Specification of the TLS0 protocol

Note that we highlighted the reception of server random N_S because we need to bind it before returning the value for the fresh key. In the projection, the intruder could freely choose any nonce, e.g., some value that the server does not know, and this does not matter because in the projection the server does not actually need to be executed: as in the previous model of the lookup, the correct public key is directly returned without actual communication with the server. Additional "trick": we use a private function ox so that PMS is bound also in the projections of the TLS0 procedures. Since the function is private, PMS is still a secret.

We further develop the model of TLS to make the server identity a parameter, where inside the lookup the argument is a fixed trusted server. See Fig. 3.

Here we do not assume that the client knows the public key of the server, but we assume that they know the public key of a trusted certificate authority so that they can check the certificate sent by the server.

```
Procedure $\mathit{tls\_client}(C : \mathsf{agent}, S : \mathsf{agent})$
$\nu N_C$ : nonce.
$\mathsf{snd}(N_C)$
;
$\mathsf{rcv}(\mathsf{g}_0(N_C, \mathsf{sign}(\mathsf{inv}(\mathsf{pk}(\mathsf{ca})), \mathsf{cert}(S, PKS)))).$
$\nu PMS$ : pms, $R$ : nonce.
$\mathsf{snd}(\mathsf{ox}(PMS)).$
$\mathsf{snd}(\mathsf{crypt}(PKS, \mathsf{g}_1(PMS, N_C), R))$
;
$\mathsf{rcv}(N_S : \mathsf{nonce}).$
$\mathsf{rcv}(\mathsf{scrypt}(\mathsf{kdf}(PMS, N_C, N_S), \mathsf{g}_2(N_S), \_ : \mathsf{nonce})).$
$\mathsf{return}(\mathsf{kdf}(PMS, N_C, N_S))$
```

```
Procedure $\mathit{tls\_server}(S : \mathsf{agent})$
$\mathsf{rcv}(N_C : \mathsf{nonce}).$
$\mathsf{snd}(\mathsf{g}_0(N_C, \mathsf{sign}(\mathsf{inv}(\mathsf{pk}(\mathsf{ca})), \mathsf{cert}(S, \mathsf{pk}(S))))).$
;
$\mathsf{rcv}(\mathsf{ox}(PMS)).$
$\mathsf{rcv}(\mathsf{crypt}(\mathsf{pk}(S), \mathsf{g}_1(PMS : \mathsf{pms}, N_C), \_ : \mathsf{nonce})).$
$\nu N_S$ : nonce, $R$ : nonce.
$\mathsf{snd}(N_S).$
$\mathsf{snd}(\mathsf{scrypt}(\mathsf{kdf}(PMS, N_C, N_S), \mathsf{g}_2(N_S), R)).$
$\mathsf{return}(\mathsf{kdf}(PMS, N_C, N_S))$
```

Fig. 3. Specification of the simplified TLS protocol