Stateful Protocol Composition

Andreas V. Hess[1], Sebastian A. Mödersheim[1], and Achim D. Brucker[2]

[1] DTU Compute, Technical University of Denmark, Lyngby, Denmark
{avhe,samo}@dtu.dk
[2] The University of Sheffield, Sheffield, United Kingdom
a.brucker@sheffield.ac.uk

Abstract. We prove a parallel compositionality result for protocols with a shared mutable state, i.e., stateful protocols. For protocols satisfying certain compositionality conditions our result shows that verifying the component protocols in isolation is sufficient to prove security of their composition. Our main contribution is an extension of the compositionality paradigm to stateful protocols where participants maintain shared databases. Because of the generality of our result we also cover many forms of sequential composition as a special case of stateful parallel composition. Moreover, we support declassification of shared secrets. As a final contribution we prove the core of our result in Isabelle/HOL, providing a strong correctness guarantee of our proofs.

1 Introduction

The typical use of communication networks like the Internet is to run a wide variety of security protocols in parallel, for example TLS, IPSec, DNSSEC, and many others. While the security properties of many of these protocols have been analyzed in great detail, much less research has been devoted to their parallel composition. It is far from self-evident that the parallel composition of secure protocols is still secure, in fact one can systematically construct counter-examples. One such problem is if protocols have similar message structures of different meaning, so that an attacker may be able to abuse messages, or parts thereof, that he has learned in the context of one protocol, and use them in the context of another where the same structure has a different meaning. Thus, we have to exclude that the protocols in some sense "interfere" with each other. However, it is unreasonable to require that the developers of the different protocols have to work together and synchronize with each other. Similarly, we do not want to reason about the composition of several protocols as a whole, neither in manual nor automated verification. Instead, we want a set of sufficient conditions and a composition theorem of the form: every set of protocols that satisfies the conditions yields a secure composition, provided that each protocol is secure in isolation. The conditions should be realistic so that many existing protocols like TLS (without modifications) actually satisfy them, and they should be simple, in the sense that checking them is a static task that does not involve considering the reachable states.

J. Lopez and J. Zhou (Eds.): ESORICS 2018, pp. 1–20, 2018.

The main contribution of this paper is the extension of the compositionality paradigm to *stateful* protocols, where participants may maintain a database (e.g., a list of valid public keys) independent of sessions. Such databases do not necessarily grow monotonically during protocol execution—we allow, for instance, negative membership checks and deletion of elements from databases. Moreover, we allow for such databases to be *shared* between the protocols to be composed. For instance, in the example of public keys, there could be several different protocols for registering, certifying, and revoking keys that all work on the same public-key database. Since such a shared database can potentially be exploited by the intruder to trigger harmful interferences, an important part of our result is a clear coordination of the ways in which each protocol is allowed to access the database. This coordination is based on assumptions and guarantees on the transactions that involve the database. Moreover, this also allows us to support protocols with the declassification of long-term secrets (e.g., that the private key to a revoked public key may be learned by the intruder without breaking the security goals). The result is so general that it actually also covers many forms of *sequential composition* as a special case, since one can for instance model that one protocol inserts keys into a database of fresh session keys, and another protocol "consumes" and uses them.

The proof of the main result is by a reduction to a problem finding solutions for intruder constraints: given a satisfiable constraint representing an attack on the composition, we show that the projection of the constraints to the individual protocols are satisfiable. This particular tricky part of the proof has been formalized in the interactive theorem prover Isabelle/HOL. This formalization, along with all proofs, is available at:

`https://people.compute.dtu.dk/samo/composec.html`

An extended version of this paper that includes the pen-and-paper proofs and short explanations of the Isabelle proofs is also available at this website [15]. Last but not least, as already indicated in [17], the formulation of the problem over intruder constraints allows us to apply our result with a variety of protocol formalisms such as applied-π calculus and multi-set rewriting.

The rest of the paper is organized as follows. Preliminaries are introduced in Section 2. In Section 3 we define stateful constraints and protocols. Afterwards we define protocol composition and introduce a keyserver protocol example in Section 4. We define our compositionality conditions and prove our main result in Section 5. Finally, we conclude in Section 6 and discuss related work.

2 Preliminaries

2.1 Terms and Substitutions

We model terms over a countable signature Σ of function symbols and a countably infinite set $\mathcal{V}$ of variable symbols. We do not fix here a particular set of cryptographic operators but rather parameterize our theory over arbitrary Σ.

A term is either a variable $x \in \mathcal{V}$ or a composed term of the form $f(t_1, \ldots, t_n)$ where $f \in \Sigma^n$ and t_i are terms and Σ^n denotes the symbols in Σ of *arity* n. The set of *constants* $\mathcal{C}$ is defined as Σ^0. The set of variables of a term t is denoted by $fv(t)$ and if $fv(t) = \emptyset$ then t is *ground*. Both of these notions are extended to sets of terms. By $\sqsubseteq$ we denote the *subterm* relation.

Substitutions are defined as functions from variables to terms. The domain of a substitution δ is denoted by $dom(\delta)$ and is defined as the set of variables that are not mapped to themselves by δ: $dom(\delta) \equiv \{x \in \mathcal{V} \mid \delta(x) \neq x\}$. The substitution image, $img(\delta)$, is then defined as the image of $dom(\delta)$ under δ: $img(\delta) \equiv \delta(dom(\delta))$. If the image of δ is ground then δ is said to be a *ground substitution*. Additionally, we define an *interpretation* to be a substitution that assigns a ground term to every variable: $\mathcal{I}$ is an interpretation iff $dom(\mathcal{I}) = \mathcal{V}$ and $img(\mathcal{I})$ is ground. We extend substitutions to functions on terms and set of terms as expected. For substitutions δ with finite domain we will usually use the common value mapping notation: $\delta = [x_1 \mapsto t_1, \ldots, x_n \mapsto t_n]$. Finally, a substitution δ is a *unifier of* terms t and t' iff $\delta(t) = \delta(t')$.

2.2 The Intruder Model

The intruder model follows the standard of Dolev and Yao, roughly, the intruder can encrypt and decrypt terms where he has the respective keys, but he cannot break the cryptography. This is often done by a set of rules specialized to the concrete cryptographic functions, but since our model is parameterized over an arbitrary set Σ, we also need to parameterize it over (a) a predicate $\mathtt{public}$ over Σ that says for each function whether it is available to the intruder and (b) a function Ana that takes a term t and returns a pair (K, T) of sets of terms. The meaning is: from the term t the intruder can obtain the terms T, provided that he knows all the "keys" in the set K. For instance if crypt is a public function symbol to represent asymmetric encryption and inv is a private function symbol (i.e., $\neg\mathtt{public}(\mathsf{inv})$) mapping public keys to the corresponding private key, then we may define $\mathsf{Ana}(\mathsf{crypt}(k, m)) = (\{\mathsf{inv}(k)\}, \{m\})$ for any terms k and m. Thus we can inductively define the relation $\vdash$, where $M \vdash t$ means that an intruder who knows the set of terms M can derive the message t as the least relation that includes M, is closed under composition with public functions and is closed under analysis with Ana as follows where $\Sigma^n_{pub} \equiv \{f \in \Sigma^n \mid \mathtt{public}(f)\}$:

Definition 1 (Intruder model).

$$\frac{}{M \vdash t}\ t \in M\ (Axiom), \qquad \frac{M \vdash t_1 \quad \cdots \quad M \vdash t_n}{M \vdash f(t_1, \ldots, t_n)}\ f \in \Sigma^n_{pub}\ (Compose),$$

$$\frac{M \vdash t \quad M \vdash k_1 \quad \cdots \quad M \vdash k_n}{M \vdash t_i}\ t_i \in T, K = \{k_1, \ldots, k_n\}\ (Decompose), \mathsf{Ana}(t) = (K, T),$$

Note that [16] in contrast considers only public function symbols; one can simulate however a private function symbol of arity n by a public function symbol of arity $n+1$ where the additional argument is used with a special constant that

is never given to the intruder; in this way all results can be lifted to a model with both private and public function symbols. For instance we can encode $\mathsf{inv} \in \Sigma^1$ in terms of a public symbol $\mathsf{inv}' \in \Sigma^2$ and a special secret constant $\mathsf{sec}_{\mathsf{inv}}$.

Our results will not work with an arbitrary analysis function, so we make the following requirements on Ana:

1. $\mathsf{Ana}(x) = (\emptyset, \emptyset)$ for variables $x \in \mathcal{V}$,
2. $\mathsf{Ana}(f(t_1, \ldots, t_n)) = (K, T)$ implies $T \subseteq \{t_1, \ldots, t_n\}$, finite K, and $fv(K) \subseteq fv(f(t_1, \ldots, t_n))$,
3. $\mathsf{Ana}(f(t_1, \ldots, t_n)) = (K, T)$ implies $\mathsf{Ana}(\delta(f(t_1, \ldots, t_n))) = (\delta(K), \delta(T))$.

Note that Ana must be defined for arbitrary terms, including terms with variables (while the standard Dolev-Yao deduction is typically applied to ground terms). The three conditions regulate that Ana is also meaningful on symbolic terms. The first requirement says that we cannot analyze a variable. The second requirement says that the result of the analysis are *immediate* subterms of the term being analyzed, and the keys can be any finite set of terms, but built with only variables that occur in the term being analyzed. The third requirement says that analysis does not change its behavior when instantiating a term (that is not a variable).

Example 1. We model asymmetric encryption and signatures with the following Ana theory: $\mathsf{Ana}(\mathsf{crypt}(k, m)) = (\{\mathsf{inv}(k)\}, \{m\})$, $\mathsf{Ana}(\mathsf{sign}(k, m)) = (\emptyset, \{m\})$. We will also later use some transparent functions: $\mathsf{Ana}(\mathsf{pair}(t, t')) = (\emptyset, \{t, t'\})$ and $\mathsf{Ana}(\mathsf{update}(s, t, u, v)) = (\emptyset, \{s, t, u, v\})$. For all other terms t: $\mathsf{Ana}(t) = (\emptyset, \emptyset)$.

3 Stateful Protocols

We now introduce a strand-based protocol formalism for stateful protocols adapted from [17]. This formalism is compact and reduced to the key concepts needed here, while more complex formalisms like process calculi can easily be fitted similarly. The semantics is defined by a symbolic transition system where constraints are built-up during transitions. The models of the constraints then constitute the concrete protocol runs. We will use a typing result that shows that for a large class of protocols, it is without loss of attacks to restrict the constraints to well-typed models [17].

3.1 Stateful Symbolic Constraints

We use *intruder constraints* as a key concept for reasoning about protocol executions and attacks. This is in fact applicable with a variety of protocol verification formalisms, such as process calculi or multi-set rewrite rules. The idea is to define a *symbolic* transition system where the variables of sent and received messages of the original protocol formalism are not instantiated (only renamed as necessary) and formulate symbolic constraints on these variables: the intruder needs to be able to construct each message an honest agent receives from the messages the honest agents have sent up to that point. When equipping these constraints

also with equalities and inequalities, the set of all executions (and the attack predicates) of many formalisms like Applied π-calculus can be described by a set of constraints. An attack can then be defined by satisfiability of a constraint in which the intruder produces a secret. *Stateful constraints* can furthermore express queries and updates on databases. They are defined as finite sequences of *steps* and are built from the following grammar where t and t' ranges over terms and $\bar{x}$ over finite variable sequences $x_1, \ldots, x_n$:

$$\begin{aligned}\mathcal{A} ::= {} & \mathsf{send}(t).\mathcal{A} \mid \mathsf{receive}(t).\mathcal{A} \mid t \doteq t'.\mathcal{A} \mid (\forall \bar{x}.\ t \not\doteq t').\mathcal{A} \mid \\ & \mathsf{insert}(t,t').\mathcal{A} \mid \mathsf{delete}(t,t').\mathcal{A} \mid t \mathrel{\dot\in} t'.\mathcal{A} \mid (\forall \bar{x}.\ t \mathrel{\dot\notin} t').\mathcal{A} \mid 0\end{aligned}$$

Instead of $\forall \bar{x}.\ t \not\doteq t'$ and $\forall \bar{x}.\ t \mathrel{\dot\notin} t'$ we may write $t \not\doteq t'$ and $t \mathrel{\dot\notin} t'$ whenever $\bar{x}$ is the empty sequence. We may also write $t \mathrel{\dot\notin} f(\_)$ for $f \in \Sigma^n$ as an abbreviation of $\forall x_1, \ldots, x_n.\ t \mathrel{\dot\notin} f(x_1, \ldots, x_n)$. The *bound variables* of a constraint $\mathcal{A}$ consists of its variable sequences while the remaining variables, $fv(\mathcal{A})$, are the *free variables*. Also, by $trms(\mathcal{A})$ we denote the set of terms occurring in $\mathcal{A}$ and the *set of set operations* of $\mathcal{A}$, called $setops(\mathcal{A})$, is defined as follows where $(\cdot,\cdot) \in \Sigma^2_{pub}$:

$$setops(\mathcal{A}) \equiv \{(t,s) \mid \mathsf{insert}(t,s) \text{ or } \mathsf{delete}(t,s) \text{ or } t \mathrel{\dot\in} s \text{ or } \forall \bar{x}.\ t \mathrel{\dot\notin} s \text{ occurs in } \mathcal{A}\}$$

For the semantics of constraints we first define a predicate $[\![M, D; \mathcal{A}]\!]\,\mathcal{I}$, where M is a ground set of terms (the intruder knowledge), D is a ground set of tuples (the state of the sets), $\mathcal{A}$ is a constraint, and $\mathcal{I}$ is an interpretation as follows:

$$\begin{aligned}
[\![M, D; 0]\!]\,\mathcal{I} &\text{ iff } \mathit{true}\\
[\![M, D; \mathsf{send}(t).\mathcal{A}]\!]\,\mathcal{I} &\text{ iff } M \vdash \mathcal{I}(t) \text{ and } [\![M, D; \mathcal{A}]\!]\,\mathcal{I}\\
[\![M, D; \mathsf{receive}(t).\mathcal{A}]\!]\,\mathcal{I} &\text{ iff } [\![\{\mathcal{I}(t)\} \cup M, D; \mathcal{A}]\!]\,\mathcal{I}\\
[\![M, D; t \doteq t'.\mathcal{A}]\!]\,\mathcal{I} &\text{ iff } \mathcal{I}(t) = \mathcal{I}(t') \text{ and } [\![M, D; \mathcal{A}]\!]\,\mathcal{I}\\
[\![M, D; (\forall \bar{x}.\ t \not\doteq t').\mathcal{A}]\!]\,\mathcal{I} &\text{ iff } [\![M, D; \mathcal{A}]\!]\,\mathcal{I} \text{ and } \mathcal{I}(\delta(t)) \neq \mathcal{I}(\delta(t'))\\
&\quad \text{for all ground substitutions } \delta \text{ with domain } \bar{x}\\
[\![M, D; \mathsf{insert}(t,s).\mathcal{A}]\!]\,\mathcal{I} &\text{ iff } [\![M, \{\mathcal{I}((t,s))\} \cup D; \mathcal{A}]\!]\,\mathcal{I}\\
[\![M, D; \mathsf{delete}(t,s).\mathcal{A}]\!]\,\mathcal{I} &\text{ iff } [\![M, D \setminus \{\mathcal{I}((t,s))\}; \mathcal{A}]\!]\,\mathcal{I}\\
[\![M, D; t \mathrel{\dot\in} s.\mathcal{A}]\!]\,\mathcal{I} &\text{ iff } \mathcal{I}((t,s)) \in D \text{ and } [\![M, D; \mathcal{A}]\!]\,\mathcal{I}\\
[\![M, D; (\forall \bar{x}.\ t \mathrel{\dot\notin} s).\mathcal{A}]\!]\,\mathcal{I} &\text{ iff } [\![M, D; \mathcal{A}]\!]\,\mathcal{I} \text{ and } \mathcal{I}(\delta((t,s))) \notin D\\
&\quad \text{for all ground substitutions } \delta \text{ with domain } \bar{x}
\end{aligned}$$

We then define that $\mathcal{I}$ is a *model* of $\mathcal{A}$, written $\mathcal{I} \models \mathcal{A}$, iff $[\![\emptyset, \emptyset; \mathcal{A}]\!]\,\mathcal{I}$.

A crucial requirement on constraints is that they are well-formed in the sense that every variable first occurs in a message the intruder sends, or in a positive check like $t \doteq t'$ or $t \mathrel{\dot\in} s$, and that the intruder knowledge monotonically grows over time. The latter condition is already built-in in our constraint notation, the former is expressed as follows: A constraint $\mathcal{A}$ is *well-formed w.r.t.* the set of variables X (or just *well-formed* if $X = \emptyset$) iff the free variables and the bound

variables of $\mathcal{A}$ are disjoint and $wf_X(\mathcal{A})$ holds where:

$$
\begin{array}{ll}
wf_X(0) & \text{iff } \mathit{true} \\
wf_X(\mathsf{receive}(t).\mathcal{A}) & \text{iff } fv(t) \subseteq X \text{ and } wf_X(\mathcal{A}) \\
wf_X(\mathsf{send}(t).\mathcal{A}) & \text{iff } wf_{X \cup fv(t)}(\mathcal{A}) \\
wf_X(t \doteq t'.\mathcal{A}) & \text{iff } fv(t') \subseteq X \text{ and } wf_{X \cup fv(t)}(\mathcal{A}) \\
wf_X(\mathsf{insert}(t,t').\mathcal{A}) & \text{iff } fv(t) \cup fv(t') \subseteq X \text{ and } wf_X(\mathcal{A}) \\
wf_X(\mathsf{delete}(t,t').\mathcal{A}) & \text{iff } fv(t) \cup fv(t') \subseteq X \text{ and } wf_X(\mathcal{A}) \\
wf_X(t \mathrel{\dot\in} t'.\mathcal{A}) & \text{iff } wf_{X \cup fv(t) \cup fv(t')}(\mathcal{A}) \\
wf_X(\mathfrak{a}.\mathcal{A}) & \text{iff } wf_X(\mathcal{A}) \text{ otherwise}
\end{array}
$$

Note that this allows to "introduce" variables in a send step, on the left-hand side of an equation, or in a positive set-membership check (and we will work only with well-formed constraints throughout the paper).

3.2 Typed Model

Our result is based on a typed model of protocols, i.e., where the intruder by definition cannot send ill-typed messages. [17] shows that this is not a restriction for a large class of so-called *type-flaw resistant* stateful protocols, since for every ill-typed attack also exists a well-typed one. This gives a sufficient condition for protocols to satisfy a prerequisite of our compositionality result. The definition of typed model is then as follows. Type expressions are terms built over the function symbols of Σ and a finite set $\mathfrak{T}_a$ of *atomic* types like Agent and Nonce. Further, we define a typing function Γ that assigns to every variable a type, to every constant an atomic type, and that is extended to composed terms as follows: $\Gamma(f(t_1,\ldots,t_n)) = f(\Gamma(t_1),\ldots,\Gamma(t_n))$ for every $f \in \Sigma^n \setminus \mathcal{C}$ and terms t_i. We also require that $\{c \in \mathcal{C} \mid \mathtt{public}(c), \Gamma(c) = \beta\}$ is infinite for each $\beta \in \mathfrak{T}_a$, thus giving the intruder access to an infinite supply of terms of each atomic type.

The sufficient condition for a protocol to satisfy the typing result is now based on the following notions. A substitution δ is *well-typed* iff $\Gamma(x) = \Gamma(\delta(x))$ for all $x \in \mathcal{V}$. Given a set of messages that occur in a protocol we define the following set of sub-message patterns, intuitively the ones that may occur during constraint reduction:

Definition 2 (Sub-message patterns). *The* sub-message patterns $SMP(M)$ *for a set of messages M is defined as the least set satisfying the following rules:*

1. $M \subseteq SMP(M)$.
2. *If* $t \in SMP(M)$ *and* $t' \sqsubseteq t$ *then* $t' \in SMP(M)$.
3. *If* $t \in SMP(M)$ *and* δ *is a well-typed substitution then* $\delta(t) \in SMP(M)$.
4. *If* $t \in SMP(M)$ *and* $\mathsf{Ana}(t) = (K, T)$ *then* $K \subseteq SMP(M)$.

The sufficient condition for the typing result is now that non-variable sub-message patterns have no unifier unless they have the same type:

Definition 3 (Type-flaw resistance). *We say a set M of messages is* type-flaw resistant *iff* $\forall t, t' \in SMP(M) \setminus \mathcal{V}.\ (\exists \delta.\ \delta(t) = \delta(t')) \longrightarrow \Gamma(t) = \Gamma(t')$. *We may also apply the notion of type-flaw resistance to a constraint $\mathcal{A}$ to mean that:*

– *trms*$(\mathcal{A}) \cup$ *setops*$(\mathcal{A})$ *is type-flaw resistant,*
– *if* t *and* t' *are unifiable then* $\Gamma(t) = \Gamma(t')$*, for all* $t \doteq t'$ *occurring in* $\mathcal{A}$*,*
– $\Gamma(fv(t) \cup fv(t')) \subseteq \mathfrak{T}_a$ *for all* $\mathsf{insert}(t, t')$ *and* $\mathsf{delete}(t, t')$ *occurring in* $\mathcal{A}$*, and*
– $\Gamma((fv(t) \cup fv(t')) \setminus \bar{x}) \subseteq \mathfrak{T}_a$ *for all* $\forall \bar{x}.\ t \not\doteq t'$ *and* $\forall \bar{x}.\ t \not\dot{\in} t'$ *occurring in* $\mathcal{A}$*.*

We have formalized in Isabelle/HOL the following typing result theorem, which shows that for type-flaw resistant protocols it is safe to check satisfiability of constraints within the typed model [17]:

Theorem 1 ([17]) *If* $\mathcal{A}$ *is a well-formed, type-flaw resistant constraint, and if* $\mathcal{I} \models \mathcal{A}$*, then there exists a well-typed interpretation* $\mathcal{I}_\tau$ *such that* $\mathcal{I}_\tau \models \mathcal{A}$*.*

3.3 Protocol Semantics

Protocols are defined as sets $\mathcal{P} = \{R_1, \ldots\}$ of *transaction rules* of the form: $R_i = \forall x_1 \in T_1, \ldots, x_n \in T_n.\ \mathsf{new}\ y_1, \ldots, y_m.\ S$ where S is a *transaction strand*, i.e., of the form $\mathsf{receive}(t_1).\cdots.\mathsf{receive}(t_k).\phi_1 \cdots \phi_{k'}.\mathsf{send}(t'_1).\cdots.\mathsf{send}(t'_{k''})$ where

$$\phi ::= t \doteq t' \mid \forall \bar{x}.\ t \not\doteq t' \mid t \dot{\in} t' \mid \forall \bar{x}.\ t \not\dot{\in} t' \mid \mathsf{insert}(t, t') \mid \mathsf{delete}(t, t')$$

The prefix $\forall x_1 \in T_1, \ldots, x_n \in T_n$ denotes that the transaction strand S is applicable for instantiations σ of the x_i variables where $\sigma(x_i) \in T_i$. The construct $\mathsf{new}\ y, \ldots, y_m$ represents that the occurrences of the variables y_i in the transaction strand S will be instantiated with fresh terms. We extend *trms*$(\cdot)$ and *setops*$(\cdot)$ to transactions strands, rules, and protocols as expected.

We define a transition relation $\Rightarrow^{\bullet}_{\mathcal{P}}$ for protocol $\mathcal{P}$ where states are constraints and the initial state is the empty constraint 0. First we define the *dual* of a transaction strand S, written *dual*(S), as "swapping" the direction of the sent and received messages of S: *dual*$(\mathsf{send}(t).S) = \mathsf{receive}(t).$*dual*$(S)$, *dual*$(\mathsf{receive}(t).S) = \mathsf{send}(t).$*dual*$(S)$, and otherwise *dual*$(\mathfrak{s}.S) = \mathfrak{s}.$*dual*$(S)$. The transition $\mathcal{A} \Rightarrow^{\bullet}_{\mathcal{P}} \mathcal{A}.$*dual*$(\alpha(\sigma(S)))$ is then applicable if these conditions are met:

1. $(\forall x_1 \in T_1, \ldots, x_n \in T_n.\ \mathsf{new}\ y_1, \ldots, y_m.\ S) \in \mathcal{P}$,
2. $dom(\sigma) = \{x_1, \ldots, x_n, y_1, \ldots, y_m\}$,
3. $\sigma(x_i) \in T_i$ for all $i \in \{1, \ldots, n\}$,
4. $\sigma(y_i)$ is a fresh ground term of type $\Gamma(y_i)$ for all $i \in \{1, \ldots, m\}$, and
5. α is a variable-renaming of the variables of $\sigma(S)$ where α is well-typed and the variables in $img(\alpha)$ do not occur in $\sigma(S)$.

Hence transaction rules are processed atomically, and converted into constraints, during transitions. Note that each transaction rule can be executed arbitrarily often and so we support an unbounded number of "sessions". For instance, the transaction rule $\forall A \in \mathsf{Hon}.\ \mathsf{new}\ PK.\ \mathsf{insert}(PK, \mathsf{ring}(A))$ models that each honest agent $a \in \mathsf{Hon}$ can insert one fresh key into its keyring $\mathsf{ring}(a)$ during each application of the transaction rule. This rule can be executed any number of times with any agent $a \in \mathsf{Hon}$ and a fresh value for PK each time.

We say that a constraint $\mathcal{A}$ is *reachable* in protocol $\mathcal{P}$ if $0 \Rightarrow^{\bullet\star}_{\mathcal{P}} \mathcal{A}$ where $\Rightarrow^{\bullet\star}_{\mathcal{P}}$ denotes the transitive reflexive closure of $\Rightarrow^{\bullet}_{\mathcal{P}}$. We need to ensure that these

constraints are well-formed and we will therefore always assume the following sufficient requirement on the protocols $\mathcal{P}$ that we work with: for any transaction strand S occurring in any rule $\forall x_1 \in T_1, \ldots, x_n \in T_n.\ \mathsf{new}\ y_1, \ldots, y_m.\ S$ of $\mathcal{P}$ the constraint $\mathit{dual}(S)$ is well-formed w.r.t. the variables $\{x_1, \ldots, x_n, y_1, \ldots, y_m\}$. In other words, the variables of S must first occur in either a $\mathsf{receive}$ step, a positive check ($\doteq$, $\dot{\in}$), or be part of $\{x_1, \ldots, x_n, y_1, \ldots, y_m\}$.

To model goal violations of a protocol $\mathcal{P}$ we first fix a special constant unique to $\mathcal{P}$, e.g., $\mathsf{attack}_{\mathcal{P}}$. Secondly, we add the rule $\mathsf{receive}(\mathsf{attack}_{\mathcal{P}})$ to $\mathcal{P}$ that we use as a signal for when an attack has occurred. The protocol then has a (well-typed) attack if there exists a (well-typed) satisfiable reachable constraint of the form $\mathcal{A}.\mathsf{send}(\mathsf{attack}_{\mathcal{P}})$. A protocol with no attacks is *secure*.

With sets we can model events, e.g., asserting an event e amounts to inserting e into a distinguished set of events while checking whether e has previously occurred (or not) corresponds to a positive (respectively negative) set-membership check. We therefore support all security properties expressible in the geometric fragment [1]. This covers many standard reachability goals such as authentication; it seems that any significantly richer fragment of first-order logic would be incompatible with our result. We do not currently support privacy-type properties, i.e., where goal violations occur if the observable behavior of protocols can be distinguished.

4 Composition and a Running Example

The core definition of this paper is rather simple: we define the *parallel composition* $\mathcal{P}_1 \parallel \mathcal{P}_2$ of protocols $\mathcal{P}_1$ and $\mathcal{P}_2$ as their union: $\mathcal{P}_1 \parallel \mathcal{P}_2 \equiv \mathcal{P}_1 \cup \mathcal{P}_2$. Protocols $\mathcal{P}_1$ and $\mathcal{P}_2$ are also referred to as the *component protocols* of the composition $\mathcal{P}_1 \parallel \mathcal{P}_2$. For such a composed protocol the reachable constraints in $\mathcal{P}_1 \parallel \mathcal{P}_2$ will in general contain steps originating from both component protocols. To keep track of where a step in a constraint originated we assign to each step a *label* $\ell \in \{1, 2, \star\}$. The steps that are exclusive to the first component are marked with 1 while the steps exclusive to the second are marked with 2. In addition to the protocol-specific labels we also have a special label $\star$ that we explain later.

Let $\mathcal{A}$ be a constraint with labels and $\ell \in \{1, 2, \star\}$, we define $\mathcal{A}|_\ell$ to be the projection of $\mathcal{A}$ to the steps labeled ℓ or $\star$ (so the $\star$-steps are kept in every a projection). We extend projections to transaction rules and protocols as expected. We may also write $\mathcal{P}^\star$ instead of $\mathcal{P}|_\star$.

4.1 A Keyserver Example

As a running example, Fig. 1 and Fig. 2 define two keyserver protocols that share the same databases of valid public keys registered at the keyserver. In a nutshell, the first protocol $\mathcal{P}_{ks,1} = \{R_1^1, \ldots, R_1^{10}\}$ allows users to register public keys out of band and to update an existing key with a new one (revoking the old key in the process), while the second protocol $\mathcal{P}_{ks,2} = \{R_2^1, \ldots, R_2^{10}\}$ uses a different mechanism to register new public keys.

$\boxed{1} \equiv 1\colon \mathsf{receive}(\mathsf{sign}(\mathsf{inv}(PK), \mathsf{pair}(A, NPK)))).\ \star\colon PK \mathrel{\dot\in} \mathsf{valid}(A, S).$
$\star\colon NPK \mathrel{\dot\notin} \mathsf{valid}(\_).\ 1\colon NPK \mathrel{\dot\notin} \mathsf{revoked}(\_)$

R_1^1	$\forall A \in \mathsf{Hon}, S \in \mathsf{Ser}.$ $1\colon \mathsf{receive}(\mathsf{inv}(PK)).\ \star\colon PK \mathrel{\dot\in} \mathsf{valid}(A, S).\ 1\colon \mathsf{send}(\mathsf{attack}_1)$
R_1^2	$\forall A \in \mathsf{Hon}, S \in \mathsf{Ser}.$ $\boxed{1}.\ \star\colon NPK \mathrel{\dot\notin} \mathsf{begin}_1(A, S).$ $1\colon \mathsf{send}(\mathsf{attack}_1)$
R_1^3	$\forall A \in \mathsf{Hon}, S \in \mathsf{Ser}.$ $\boxed{1}.\ \star\colon NPK \mathrel{\dot\in} \mathsf{begin}_1(A, S).\ \star\colon NPK \mathrel{\dot\in} \mathsf{end}_1(A, S).$ $1\colon \mathsf{send}(\mathsf{attack}_1)$
R_1^4	$\forall A \in \mathsf{Dis}.\ \mathsf{new}\ PK.$ $\star\colon \mathsf{send}(PK).\ \star\colon \mathsf{send}(\mathsf{inv}(PK))$
R_1^5	$\forall A \in \mathsf{Hon}, S \in \mathsf{Ser}.\ \mathsf{new}\ PK.$ $1\colon \mathsf{insert}(PK, \mathsf{ring}(A)).\ \star\colon \mathsf{insert}(PK, \mathsf{valid}(A, S)).$ $\star\colon \mathsf{insert}(PK, \mathsf{begin}_1(A, S)).\ \star\colon \mathsf{insert}(PK, \mathsf{end}_1(A, S)).$ $\star\colon \mathsf{send}(PK)$
R_1^6	$\forall A \in \mathsf{Hon}, S \in \mathsf{Ser}.\ \mathsf{new}\ NPK.$ $1\colon PK \mathrel{\dot\in} \mathsf{ring}(A).\ 1\colon \mathsf{delete}(PK, \mathsf{ring}(A)).$ $1\colon \mathsf{insert}(NPK, \mathsf{ring}(A)).\ \star\colon \mathsf{insert}(NPK, \mathsf{begin}_1(A, S)).$ $\star\colon \mathsf{send}(NPK).\ 1\colon \mathsf{send}(\mathsf{sign}(\mathsf{inv}(PK), \mathsf{pair}(A, NPK)))$
R_1^7	$\forall A \in \mathsf{Hon}, S \in \mathsf{Ser}.$ $\boxed{1}.\ \star\colon NPK \mathrel{\dot\in} \mathsf{begin}_1(A, S).\ \star\colon NPK \mathrel{\dot\notin} \mathsf{end}_1(A, S).$ $\star\colon \mathsf{delete}(PK, \mathsf{valid}(A, S)).\ \star\colon \mathsf{insert}(NPK, \mathsf{valid}(A, S)).$ $1\colon \mathsf{insert}(PK, \mathsf{revoked}(A, S)).\ \star\colon \mathsf{insert}(NPK, \mathsf{end}_1(A, S)).$ $\star\colon \mathsf{send}(\mathsf{inv}(PK))$
R_1^8	$\forall A \in \mathsf{Dis}, S \in \mathsf{Ser}.$ $\boxed{1}.\ \star\colon \mathsf{delete}(PK, \mathsf{valid}(A, S)).$ $\star\colon \mathsf{insert}(NPK, \mathsf{valid}(A, S)).$ $1\colon \mathsf{insert}(PK, \mathsf{revoked}(A, S))$
R_1^9	$\forall A \in \mathsf{Dis}, S \in \mathsf{Ser}.$ $1\colon \mathsf{receive}(PK).\ \star\colon PK \mathrel{\dot\notin} \mathsf{valid}(\_).\ \star\colon \mathsf{insert}(PK, \mathsf{valid}(A, S))$
R_1^{10}	$1\colon \mathsf{receive}(\mathsf{attack}_1)$

Fig. 1. The transaction rules of the first keyserver protocol $\mathcal{P}_{ks,1}$.

We use here three atomic types: the type of agents Agent, public keys PubKey, and the type Attack of the attack_i constants. We partition type Agent into the honest users Hon, the dishonest users Dis, and the keyservers Ser. There are sets for authentication goals begin_1, end_1, begin_2, and end_2, and all protocol steps related to these sets are highlighted in gray; let us first ignore these.

Protocol $\mathcal{P}_{ks,1}$ In the first protocol, rule R_1^5 models that an honest user registers a new public key PK out of band (e.g., by physically visiting a registration site); this is achieved by inserting PK (in the same transaction) both into a keyring $\mathsf{ring}(A)$ for user A and into a shared database $\mathsf{valid}(A, S)$ of the user's currently valid keys. There is also a corresponding rule for dishonest users: R_1^9. Dishonest users may register in their name any key they know (hence the $\mathsf{receive}(PK)$ step),

$\boxed{2} \equiv 2\colon \mathsf{receive}(\mathsf{crypt}(PK, \mathsf{update}(A, S, NPK, \mathsf{pw}(A, S)))).$
$2\colon PK \mathrel{\dot\in} \mathsf{pubkeys}(S).\ 2\colon NPK \mathrel{\dot\notin} \mathsf{pubkeys}(\_).\ 2\colon NPK \mathrel{\dot\notin} \mathsf{seen}(\_)$

R_2^1	$\forall A \in \mathsf{Hon}, S \in \mathsf{Ser}.$ $2\colon \mathsf{receive}(\mathsf{inv}(PK)).\ \star\colon PK \mathrel{\dot\in} \mathsf{valid}(A, S).\ 2\colon \mathsf{send}(\mathsf{attack}_2)$
R_2^2	$\forall A \in \mathsf{Hon}, S \in \mathsf{Ser}.$ $\boxed{2}.\ \star\colon NPK \mathrel{\dot\notin} \mathsf{begin}_2(A, S).\ 2\colon \mathsf{send}(\mathsf{attack}_2)$
R_2^3	$\forall A \in \mathsf{Hon}, S \in \mathsf{Ser}.$ $\boxed{2}.\ \star\colon NPK \mathrel{\dot\in} \mathsf{begin}_2(A, S).\ \star\colon NPK \mathrel{\dot\in} \mathsf{end}_2(A, S).$ $2\colon \mathsf{send}(\mathsf{attack}_2)$
R_2^4	$\forall A \in \mathsf{Dis}.\ \mathsf{new}\ PK.$ $\star\colon \mathsf{send}(PK).\ \star\colon \mathsf{send}(\mathsf{inv}(PK))$
R_2^5	$\forall A \in \mathsf{Hon}, S \in \mathsf{Ser}.\ \mathsf{new}\ NPK.$ $2\colon PK \mathrel{\dot\in} \mathsf{pubkeys}(S).\ \star\colon \mathsf{insert}(NPK, \mathsf{begin}_2(A, S)).$ $\star\colon \mathsf{send}(NPK).\ 2\colon \mathsf{send}(\mathsf{crypt}(PK, \mathsf{update}(A, S, NPK, \mathsf{pw}(A, S))))$
R_2^6	$\forall A \in \mathsf{Hon}, S \in \mathsf{Ser}.$ $\boxed{2}.\ \star\colon NPK \mathrel{\dot\in} \mathsf{begin}_2(A, S).\ \star\colon NPK \mathrel{\dot\notin} \mathsf{end}_2(A, S).$ $\star\colon \mathsf{insert}(NPK, \mathsf{valid}(A, S)).\ \star\colon \mathsf{insert}(NPK, \mathsf{end}_2(A, S)).$ $2\colon \mathsf{insert}(NPK, \mathsf{seen}(A))$
R_2^7	$\forall A \in \mathsf{Dis}, S \in \mathsf{Ser}.$ $2\colon \mathsf{send}(\mathsf{pw}(A, S))$
R_2^8	$\forall A \in \mathsf{Dis}, S \in \mathsf{Ser}.$ $\boxed{2}.\ \star\colon \mathsf{insert}(PK, \mathsf{valid}(A, S)).\ 2\colon \mathsf{insert}(PK, \mathsf{seen}(A))$
R_2^9	$\forall S \in \mathsf{Ser}.\ \mathsf{new}\ PK.$ $2\colon \mathsf{insert}(PK, \mathsf{pubkeys}(S)).\ \star\colon \mathsf{send}(PK)$
R_2^{10}	$2\colon \mathsf{receive}(\mathsf{attack}_2)$

Fig. 2. The transaction rules of the second keyserver protocol $\mathcal{P}_{ks,2}$.

so the key is not necessarily freshly created; also we do not model a keyring for them. (Rule R_i^4 gives the intruder access to arbitrarily many fresh key pairs.)

Secondly, we model a key update with revocation of old keys. To request an update of key PK with a newly generated key NPK at server S, an honest user sends NPK signed with PK as in R_1^6. (For this rule there is no equivalent for the dishonest agents, since they may produce an arbitrary update request message.)

The rule R_1^7 shows how S receives the update message from an honest agent: it checks ($\boxed{1}$) that the key PK is currently valid, and that NPK is neither registered as valid or revoked. If so, it updates its databases accordingly: it moves the old key from $\mathsf{valid}(A, S)$ to $\mathsf{revoked}(A, S)$ and registers the new key NPK by inserting it into $\mathsf{valid}(A, S)$. Also, we reveal here $\mathsf{inv}(PK)$, in order to specify that the protocol must even be secure when old private keys are leaked. This is an example of declassification of a secret shared between two protocols: after intentionally revealing $\mathsf{inv}(PK)$ it should no longer count as a secret. The rule R_1^8 is the pendant for dishonest agents. The last rule R_1^{10} acts as a signal for when an attack has occurred in $\mathcal{P}_{ks,1}$.

Protocol $\mathcal{P}_{ks,2}$ The second protocol has another mechanism to register new keys: every user has a password $\mathsf{pw}(A, S)$ with the server (the dishonest agents reveal their password to the intruder with rule R_2^7). Instead of using a (possibly weak) password for an encryption, the registration message is encrypted with the public key of the server (rule R_2^5). For uniformity, we model the server's public keys in a set $\mathsf{pubkeys}(S)$ that is initialized with rule R_2^9 (in fact, the server may thus have multiple public keys). Rule R_2^6 models how the server receives a registration request (in case of honest users): to protect against replay, the server uses a set seen of seen keys (this may in a real implementation be a buffer-timestamp mechanism). Rule R_2^8 is the pendant for the dishonest users. Finally, the rule R_2^{10} acts as a signal for when an attack has occurred in $\mathcal{P}_{ks,2}$.

Authentication Besides the secrecy goal R_i^1 that no valid private key of an honest agent may ever be known by the intruder, the crucial authentication goal is that all insertions into $\mathsf{valid}(A, S)$ for honest A are authenticated. The classical injective agreement is modeled by the steps highlighted in gray: when an honest agent generates a fresh key for server, it inserts it into a special set begin, and whenever a server accepts a key that appears to come from an honest agent A, then it inserts it into a special set end. (Note that these sets exist only in our model to specify the goals.) It is a violation of non-injective agreement if the server accepts a key that is not in begin (rule R_i^2), and of injective agreement if the server accepts a key that is already in end (rule R_i^3).

Such a specification is more declarative when one separates the protocol rules from the attack rules, but that has one drawback: if the protocol indeed had an attack, then one would allow the server to actually insert an unauthenticated key into its database and then in the next step the attack rule fires. For the composition result, however, we want that each protocol can rely on the other protocols to never insert unauthenticated keys into the database. This is why we integrate in rules R_i^6 of each protocol the checks that we are in an authenticated case (otherwise, the rules R_i^2 or R_i^3 fire). This is similar to a "lookahead" where we prevent the execution of a transition if it leads to an attack, and directly trigger an attack. This computation of the lookahead version of goals may of course be lifted from the user by verification tools.

5 The Compositionality Results

With stateful protocols and parallel composition defined we can now formally define the concepts underlying our results and state our compositionality theorems. We first provide a result on the level of constraints and afterwards show our main theorems for stateful protocols.

5.1 Protocol Abstraction

Note that all steps containing the valid set family in our keyserver example have been labeled with $\star$. Labeling operations on the shared sets with $\star$ is actually an important part of our compositionality result and we now explain why.

Essentially, compositionality results aim to prevent that attacks can arise from the composition itself, i.e., attacks that do not similarly work on the components in isolation. Thus we want to show that attacks on the composed system can be sufficiently decomposed into attacks on the components. This however cannot directly work if the components have shared sets like valid in the example: if one protocol inserts something to a set and the other protocol reads from the set, then this trace in general does not have a counter-part in the second protocol alone. We thus need a kind of *interface* to how the two protocols can influence their shared sets. In the key server example, both protocols can insert public keys into the shared set valid, the first protocol can even remove them. The idea is now that we develop from each protocol an *abstract* version that subsumes all the modifications that the concrete protocol can perform on the shared sets. This can be regarded as a "contract" for the composition: each protocol *guarantees* that it will not make any modifications that are not covered by its abstract protocol, and it will *assume* that the other protocol only makes modifications covered by the other protocol's abstraction. We will still have to verify that each individual protocol is also secure when running together with the other abstract protocol, but this is in general much simpler than the composition of the two concrete protocols. (In the special case that the protocols share no sets, i.e. like in all previous parallel composition results, the abstractions are empty, i.e., we have to verify only the individual components.)

In general, the abstraction of a component protocol $\mathcal{P}$ is defined by restriction to those steps that are labeled $\star$, i.e., $\mathcal{P}^\star$. We require that at least the modification of shared sets are labeled $\star$. In the keyserver example we have also labeled the operations on the authentication-related sets with a $\star$ (everything highlighted in gray): we need to ensure that we insert into the set of valid keys of an honest agent only those keys that really have been created by that agent and that have not been previously inserted. So the contract between the two protocols is that they only insert keys that are properly authenticated, but the abstraction ignores how each protocol achieves the authentication (e.g. signatures vs. passwords and seen-set). There are also some outgoing messages labeled with $\star$ which we discuss a little below.[3]

Example 2. Consider the abstractions of rules R_2^5 and R_2^6:

$$\begin{array}{ll}
\forall A \in \mathsf{Hon}, S \in \mathsf{Ser}.\ \mathsf{new}\ NPK. & \forall A \in \mathsf{Hon}, S \in \mathsf{Ser}. \\
\quad \star\colon \mathsf{insert}(NPK, \mathsf{begin}_2(A,S)). & \quad \star\colon NPK \stackrel{\cdot}{\in} \mathsf{begin}_2(A,S). \\
\quad \star\colon \mathsf{send}(NPK) & \quad \star\colon NPK \stackrel{\cdot}{\notin} \mathsf{end}_2(A,S). \\
 & \quad \star\colon \mathsf{insert}(NPK, \mathsf{valid}(A,S)). \\
 & \quad \star\colon \mathsf{insert}(NPK, \mathsf{end}_2(A,S))
\end{array}$$

Notice that the gray steps prevent unauthenticated key registration because keys can only be registered if inserted into begin_2 by an honest agent. If we *did not*

[3] We require also well-formedness of the $\star$-projected protocols. This is violated, for instance, if a protocol contains a rule where only one outgoing message is labeled $\star$ and this message contains variables. However, given that the concrete protocol is already well-formed, this is easy to fix automatically, transparent to the user.

ensure such authenticated key-registration then the intruder would be able to register arbitrary keys in $\mathcal{P}^{\star}_{ks,2}$. This would lead to an attack on secrecy in the protocol $\mathcal{P}_{ks,1} \parallel \mathcal{P}^{\star}_{ks,2}$.

One may wonder why there is no similar specification for secrecy, i.e., that $\mathsf{inv}(NPK)$ is secret for every key NPK that is being inserted into valid. In fact, below we will declare all private keys to be secret by default. Thus, unless explicitly declassified, they are (implicitly) required to be secret.

5.2 Shared Terms

Before giving the compositionality conditions we first formally define what terms can be shared: Every term t that occurs in multiple component protocols must be either a *basic public term* (meaning that the intruder can derive t without prior knowledge, i.e., $\emptyset \vdash t$) or a *shared secret*. If the intruder learns a shared secret (that has not been explicitly declassified) then it is considered a violation of secrecy in *all* component protocols. For instance, agent names are usually basic public terms whereas private keys are secrets. In fact, we will have that *all* shared terms (except basic public terms) are by default secrets—even public keys—before they are declassified.

Let *Sec* be a set of ground terms, representing the initially shared secrets of the protocols. Note that the set of shared secrets *Sec* is not a fixed predefined set of terms, but rather just a parameter to our compositionality condition. We require that all shared terms of the protocols are either in *Sec* or basic public terms. To precisely define this requirement, we first define the *ground sub-message patterns (GSMP)* of a set of terms M as $GSMP(M) \equiv \{t \in SMP(M) \mid fv(t) = \emptyset\}$. This definition is extended to constraints $\mathcal{A}$ as the set $GSMP(\mathcal{A}) \equiv GSMP(trms(\mathcal{A}) \cup setops(\mathcal{A}))$, and similarly for protocols. To make matters smooth, we also require that $Sec \cup \{t \mid \emptyset \vdash t\}$ is closed under subterms (which is trivially the case for the basic public terms).

Example 3. We will typically study the ground subterms of each individual protocol in parallel with the abstraction of the other. For the example, the set $GSMP(\mathcal{P}_{ks,1} \parallel \mathcal{P}^{\star}_{ks,2})$ is the closure under subterms of the following set:

$$\{\mathsf{attack}_1, (pk, \mathsf{ring}(a)), (pk, \mathsf{valid}(a,s)), (pk, \mathsf{revoked}(a,s)), (pk, \mathsf{begin}_i(a,s)),\\ (pk, \mathsf{end}_i(a,s)), \mathsf{sign}(\mathsf{inv}(pk), \mathsf{pair}(a, npk)) \mid i \in \{1,2\}, pk, npk, a, s \in \mathcal{C},\\ \Gamma(\{pk, npk\}) = \{\mathsf{PubKey}\}, \Gamma(\{a,s\}) = \{\mathsf{Agent}\}\}$$

and $GSMP(\mathcal{P}^{\star}_{ks,1} \parallel \mathcal{P}_{ks,2})$ is the closure under subterms of the following set:

$$\{\mathsf{attack}_2, (pk, \mathsf{valid}(a,s)), (pk, \mathsf{seen}(a,s)), (pk, \mathsf{begin}_i(a,s)), (pk, \mathsf{end}_i(a,s)),\\ (pk, \mathsf{pubkeys}(s)), \mathsf{inv}(pk), \mathsf{crypt}(pk, \mathsf{update}(a, s, npk, \mathsf{pw}(a,s))) \mid i \in \{1,2\},\\ pk, npk, a, s \in \mathcal{C}, \Gamma(\{pk, npk\}) = \{\mathsf{PubKey}\}, \Gamma(\{a,s\}) = \{\mathsf{Agent}\}\}$$

For composition we will require that two protocols are disjoint in their ground sub-message patterns except for basic public terms and shared secrets:

Definition 4 (GSMP disjointedness). *Given two sets of terms M_1 and M_2, and a ground set of terms Sec (the shared secrets), we say that M_1 and M_2 are Sec-*GSMP disjoint *iff $GSMP(M_1) \cap GSMP(M_2) \subseteq Sec \cup \{t \mid \emptyset \vdash t\}$. This is extended to constraints and protocols as expected.*

5.3 Declassification and Leaking

Up until now the set of shared secrets has been static. We now remove this restriction by introducing a notion of declassification that will allow shared secrets to become public during protocol execution. For instance, in protocol $\mathcal{P}_{ks,1}$ we give revoked private keys of the form $\mathsf{inv}(PK)$ to the intruder by transmitting them over the network: $\mathsf{send}(\mathsf{inv}(PK))$. The transmitted key $\mathsf{inv}(PK)$ should no longer be secret after transmission and so we call such steps *declassification*. Since declassification involves shared secrets we require that they are declassified for all component protocols together. Thus we label them with $\star$.

For any constraint $\mathcal{A}$ with model $\mathcal{I}$ we can now formally define the set of secrets that has been declassified in $\mathcal{A}$ under $\mathcal{I}$:

Definition 5 (Declassification). *Let $\mathcal{A}$ be a labeled constraint and $\mathcal{I}$ a model of $\mathcal{A}$. Then $declassified(\mathcal{A}, \mathcal{I}) \equiv \mathcal{I}(\{t \mid \star\colon \mathsf{receive}(t)$ occurs in $\mathcal{A}\})$ is the set of* declassified secrets of $\mathcal{A}$ under $\mathcal{I}$.

Given a protocol $\mathcal{P}$, a reachable constraint $\mathcal{A}$ (i.e., $0 \Rightarrow_{\mathcal{P}}^{\bullet\star} \mathcal{A}$), and a model $\mathcal{I}$ of $\mathcal{A}$, then $\mathcal{I}(\mathcal{A})$ represents a concrete protocol run and the set $declassified(\mathcal{A}, \mathcal{I})$ represents the messages that have been declassified by honest agents during the protocol run. Note that in this definition we have reversed the direction of the declassification transmission, because the send and $\mathsf{receive}$ steps of reachable constraints are duals of the transaction rules they originated from.

Declassification also allows us to share terms that have shared secrets as subterms but which are not themselves meant to be secret. For instance, public key certificates have as subterm the private key of the signing authority, and such certificates can be shared between protocols by modeling them as shared secrets that are declassified when first published.

Finally, if the intruder learns a secret that has not been declassified then it counts as an attack. We say that protocol $\mathcal{P}$ *leaks* a secret s if there is a reachable satisfiable constraint $\mathcal{A}$ where the intruder learns s before it is declassified:

Definition 6 (Leakage). *Let Sec be a set of secrets and $\mathcal{I}$ be a model of the labeled constraint $\mathcal{A}$. $\mathcal{A}$* leaks a secret from *Sec* under $\mathcal{I}$ *iff there exists $s \in Sec \setminus declassified(\mathcal{A}, \mathcal{I})$ such that $\mathcal{I} \models \mathcal{A}|_1.\mathsf{send}(s)$ or $\mathcal{I} \models \mathcal{A}|_2.\mathsf{send}(s)$.*

Our notion of leakage requires that one of the components in isolation leaks a secret. This is important for our compositionality result later—we will require protocols not to leak in isolation (which can be verified on the protocols in isolation) for the composition to work. Note also that the set $declassified(\mathcal{A}, \mathcal{I})$ is unchanged during projection of $\mathcal{A}$, and so it suffices to pick the leaked s from the set $Sec \setminus declassified(\mathcal{A}, \mathcal{I})$ instead of $Sec \setminus declassified(\mathcal{A}|_i, \mathcal{I})$.

Example 4. The terms occurring in the GSMP intersection of the two keyserver protocols are (a) public keys pk, (b) private keys of the form $\mathsf{inv}(\mathsf{pk})$, (c) agent names, and (d) operations on the shared set families valid, begin_i, and end_i. Agent names are basic public terms in our example, i.e., $\emptyset \vdash \mathsf{a}$ for all constants a of type Agent. The public keys are initially secret, but we immediately declassify them whenever they are generated. To satisfy GSMP disjointedness of $\mathcal{P}_{ks,1} \parallel \mathcal{P}^{\star}_{ks,2}$ and $\mathcal{P}^{\star}_{ks,1} \parallel \mathcal{P}_{ks,2}$ it thus suffices to choose the following set as the set of shared secrets (where the sec_f are special secret constants used in the encoding of the private function symbol f):

$$\begin{aligned} Sec = \{&pk, \mathsf{inv}(pk), (pk, f(a,s)), f(a,s), \mathsf{sec}_{\mathsf{inv}}, \mathsf{sec}_f \mid \Gamma(\{a,s\}) = \{\mathsf{Agent}\}, \\ &\Gamma(pk) = \mathsf{PubKey}, f \in \{\mathsf{valid}, \mathsf{begin}_1, \mathsf{end}_1, \mathsf{begin}_2, \mathsf{end}_2\}, pk, a, s \in \mathcal{C}\} \end{aligned}$$

Note that we want the set symbols like valid to be private. This is because terms like $\mathsf{valid}(A,S)$ occurs in both component protocols and so we have to prevent the intruder from constructing them.

5.4 Parallel Compositionality for Constraints

With these concepts defined we can list the requirements on constraints that are necessary to apply our result on the constraint level:

Definition 7 (Parallel composability). *Let $\mathcal{A}$ be a constraint and let Sec be a ground set of terms. Then $(\mathcal{A}, Sec)$ is* parallel composable *iff*

1. *$\mathcal{A}|_1$ and $\mathcal{A}|_2$ are Sec-GSMP disjoint,*
2. *for all terms t the step $\star\colon \mathsf{send}(t)$ does not occur in $\mathcal{A}$,*
3. *for all $s \in Sec$ and $s' \sqsubseteq s$, either $\emptyset \vdash s'$ or $s' \in Sec$,*
4. *for all $\ell\colon (t,s), \ell'\colon (t',s') \in labeledsetops(\mathcal{A})$, if (t,s) and (t',s') are unifiable then $\ell = \ell'$,*
5. *$\mathcal{A}$ is type-flaw resistant and $\mathcal{A}$, $\mathcal{A}|_1$, $\mathcal{A}|_2$, and $\mathcal{A}|_\star$ are all well-formed,*

where $labeledsetops(\mathcal{A}) \equiv \{\ell\colon (t,s) \mid \ell\colon \mathsf{insert}(t,s)$ or $\ell\colon \mathsf{delete}(t,s)$ or $\ell\colon t \doteq s$ or $\ell\colon (\forall \vec{x}.\ t \not\doteq s)$ occurs in $\mathcal{A}\}$. (This definition is also extended to protocols.)

The first requirement is at the core of our compositionality result and states that the protocols can only share basic public terms and shared secrets. The second requirement ensures that $\star$ steps are only used for declassification, checks, and stateful steps. The third condition is our only requirement on the shared terms; it ensures that the set $Sec \cup \{t \mid \emptyset \vdash t\}$ is closed under subterms. The fourth condition is our requirement on stateful protocols; it implies that shared sets must be labeled with a $\star$. Finally, the last condition is needed to apply the typing result and it is orthogonal to the other conditions; it is indeed only necessary so that we can apply Theorem 1 and restrict ourselves to well-typed attacks. Typing results with different requirements could potentially be used instead. Note that we require well-formedness of *all* projections of $\mathcal{A}$. This is because we usually consider constraints reachable in composed and augmented protocols, and we need well-formedness to apply the typing result to these constraints.

With these requirements defined we can state our main result on constraints:

Theorem 2 *If $(\mathcal{A}, Sec)$ is parallel composable and $\mathcal{I} \models \mathcal{A}$ then there exists a well-typed interpretation $\mathcal{I}_\tau$ such that either $\mathcal{I}_\tau \models \mathcal{A}|_1$ and $\mathcal{I}_\tau \models \mathcal{A}|_2$ or some prefix $\mathcal{A}'$ of $\mathcal{A}$ leaks a secret from Sec under $\mathcal{I}_\tau$.*

That is, we can obtain a well-typed model of projections $\mathcal{A}|_1$ and $\mathcal{A}|_2$ for satisfiable parallel composable constraints $\mathcal{A}$—or one of the projections has leaked a secret. In other words, if we can verify that a parallel composable constraint $\mathcal{A}$ does not have any well-typed model of both projections, and no prefix of $\mathcal{A}$ leaks a secret under any well-typed model, then it is unsatisfiable.

5.5 Parallel Compositionality for Protocols

Until now our parallel compositionality result has been stated on the level of constraints. As a final step we now explain how we can use Theorem 2 to prove a parallel compositionality result for protocols.

First, we define the *traces* of a protocol $\mathcal{P}$ as the set of reachable constraints: $traces(\mathcal{P}) \equiv \{\mathcal{A} \mid 0 \Rightarrow_{\mathcal{P}}^{\bullet\star} \mathcal{A}\}$. We then define a compositionality requirement on protocols that ensures that all traces are parallel composable:

Definition 8 (Parallel composability, for protocols). *Let $\mathcal{P}_1 \parallel \mathcal{P}_2$ be a composed protocol and let Sec be a ground set of terms. Then $(\mathcal{P}_1 \parallel \mathcal{P}_2, Sec)$ is* parallel composable *iff*

1. *$\mathcal{P}_1 \parallel \mathcal{P}_2^\star$ and $\mathcal{P}_1^\star \parallel \mathcal{P}_2$ are Sec-GSMP disjoint,*
2. *for all terms t the step $\star\colon \mathsf{receive}(t)$ does not occur in $\mathcal{P}_1 \parallel \mathcal{P}_2$,*
3. *for all $s \in Sec$ and $s' \sqsubseteq s$, either $\emptyset \vdash s'$ or $s' \in Sec$,*
4. *for all $\ell\colon (t, s), \ell'\colon (t', s') \in labeledsetops(\mathcal{P}_1 \parallel \mathcal{P}_2)$, if (t, s) and (t', s') are unifiable then $\ell = \ell'$,*
5. *$\mathcal{P}_1 \parallel \mathcal{P}_2$ is type-flaw resistant and $\mathcal{P}_1$, $\mathcal{P}_2$, $\mathcal{P}_1^\star$, and $\mathcal{P}_2^\star$ are all well-formed.*

For protocols we need to require that their composition is type-flaw resistant. It is not sufficient to simply require it for the component protocols in isolation; unifiable messages from different protocols might break type-flaw resistance otherwise. Note also that type-flaw resistance of a protocol $\mathcal{P}$ implies that the traces of $\mathcal{P}$ are type-flaw resistant, because $SMP(\mathcal{A}) \subseteq SMP(\mathcal{P})$ for any $\mathcal{A} \in traces(\mathcal{P})$ and because the traces consists of the duals of the transaction strands occurring in the protocol; likewise for GSMP disjointedness. Thus if $(\mathcal{P}_1 \parallel \mathcal{P}_2, Sec)$ is parallel composable then $(\mathcal{A}, Sec)$ is parallel composable for any $\mathcal{A} \in traces(\mathcal{P}_1 \parallel \mathcal{P}_2)$.

Example 5. Continuing example 4 we now show that $\mathcal{P}_{ks,1} \parallel \mathcal{P}_{ks,2}$ is parallel composable, i.e., that it satisfies the conditions of Definition 8. We have previously shown type-flaw resistance and well-formedness for a similar keyserver protocol [17] and so we focus on the remaining four conditions here. GSMP disjointedness of the composed keyserver protocols was explained in example 4. Hence the first condition of Definition 8 is satisfied. Conditions two and three are satisfied since $\mathcal{P}_{ks,1} \parallel \mathcal{P}_{ks,2}$ does not contain any steps of the form $\star\colon \mathsf{receive}(t)$ and since any subterm of a term from *Sec* (as defined in the previous example) is either in *Sec* or an agent name (a basic public term). Note that

labeledsetops$(\mathcal{P}_{ks,1} \parallel \mathcal{P}_{ks,2})$ consists of instances of labeled terms from the following set: $\{1\colon (PK_0, \mathsf{ring}(A_0)),\ 1\colon (PK_1, \mathsf{revoked}(A_1, S_1)),\ 2\colon (PK_2, \mathsf{seen}(A_2, S_2)),\ \star\colon (PK_3, \mathsf{valid}(A_3, S_3)),\ \star\colon (PK_4^i, \mathsf{begin}_i(A_4^i, S_4^i)),\ \star\colon (PK_5^i, \mathsf{end}_i(A_5^i, S_5^i)) \mid i \in \{1, 2\}\}$. For all pairs $\ell\colon (t, s)$, $\ell'\colon (t', s')$ in this set we have that $\ell = \ell'$ if (t, s) and (t', s') are unifiable. Hence condition 4 is satisfied.

As a consequence of Theorem 2 we have that any protocol $\mathcal{P}_1$ can be safely composed with another protocol $\mathcal{P}_2$ provided that $\mathcal{P}_1 \parallel \mathcal{P}_2^\star$ is secure and that $\mathcal{P}_1^\star \parallel \mathcal{P}_2$ does not leak a secret:

Theorem 3 *If $(\mathcal{P}_1 \parallel \mathcal{P}_2, Sec)$ is parallel composable, $\mathcal{P}_1 \parallel \mathcal{P}_2^\star$ is well-typed secure in isolation, and $\mathcal{P}_1^\star \parallel \mathcal{P}_2$ does not leak a secret under any well-typed model, then all goals of $\mathcal{P}_1$ hold in $\mathcal{P}_1 \parallel \mathcal{P}_2$ (even in the untyped model).*

Note that the only requirement on protocol $\mathcal{P}_2$ is that it does not leak any secrets (before declassifying), but we do not require that $\mathcal{P}_2$ is completely secure. This means, if we have a secure protocol $\mathcal{P}_1$, that the goals of $\mathcal{P}_1$ continue to hold in any composition with another protocol $\mathcal{P}_2$ that satisfies the composability conditions and does not leak secrets, even if $\mathcal{P}_2$ has some attacks. This is in particular interesting if we run a protocol $\mathcal{P}_1$ in composition with a large number of other protocols that are too complex to verify in all detail.

Finally, the composition of parallel composable and secure protocols is secure:

Corollary 1. *If $(\mathcal{P}_1 \parallel \mathcal{P}_2, Sec)$ is parallel composable and $\mathcal{P}_1 \parallel \mathcal{P}_2^\star$ and $\mathcal{P}_1^\star \parallel \mathcal{P}_2$ are both secure in isolation then the composition $\mathcal{P}_1 \parallel \mathcal{P}_2$ is also secure (even in the untyped model).*

5.6 Sequential Composition

Until now we have focused entirely on parallel composition where protocols are run "side-by-side". Another type of protocol composition is sequential composition where protocols are run in sequence, e.g. most recently [6] for PKIs. Thanks to the generality of our result, we can cover such sequential compositions as a parallel composition with sets dedicated to the hand-over between the protocols. Let us take a key-exchange protocols like TLS as an example, where the handshake protocol establishes a pair of shared keys between a client A and a server S, and then subsequently, the transport protocol uses these keys to encrypt communication between A and S. We illustrate how the last transition of the handshake and the first transition of the transport protocol look for A where t_1 and t_2 are terms representing the two shared keys established in the handshake (and there are similar rules for S):

$$\begin{array}{ll}
\forall A \in \mathsf{Hon}, S \in \mathsf{Ser}. & \forall A \in \mathsf{Hon}, S \in \mathsf{Ser}. \\
\quad 1\colon \cdots & \quad \star\colon (K_1, K_2) \doteq \mathsf{keys}(A, S). \\
\quad \star\colon \mathsf{insert}((t_1, t_2), \mathsf{keys}(A, S)) & \quad \star\colon \mathsf{delete}((K_1, K_2), \mathsf{keys}(A, S)). \\
& \quad 2\colon \cdots
\end{array}$$

Note that, like in the keyserver example, the set $\mathsf{keys}(A, S)$ does not represent a means of communication between two participants, but rather a buffer or glue between two protocols: one protocol is producing keys, the other protocol is consuming them. Of course, one needs to require here that the first protocol only inserts authenticated and secret keys into the set, which is similar to the assume-guarantee reasoning we have illustrated for our keyserver example.

In fact, our result allows for a generalization of existing sequential composition results: while all results like [6] and the similar vertical result [11] are specialized to a particular set of data to be transferred from one protocol to another, our result does not prescribe a particular setup, but allows for any exchange of data through shared sets. This only requires one to specify sufficient assumptions on the shared-set operations for the assume-guarantee reasoning, but one does no longer need to establish a new composition theorem for each new form of sequential composition. In fact, the composition does not even need to be strictly sequential, e.g. if the first protocol establishes keys for the second protocol, one may well have that additionally the second protocol can also establish new keys for subsequent sessions.

6 Conclusion and Related Work

Our composition theorem for parallel composition is the newest in a sequence of parallel composition results that are each pushing the boundaries of the class of protocols that can be composed [14,13,2,12,9,8,7,3,1]. The first results simply require completely disjoint encryptions; subsequent results allowed the sharing of long-term keys, provided that wherever the common keys are used, the content messages of the different protocols are distinguished, for instance by tagging. Other aspects are which primitives are supported as well as what forms of negative conditions, e.g. to support as goals the full geometric fragment.

Our result lifts the common requirement that the component protocols only share a fixed set of long-term public and private constants. Our result allows for stateful protocols that maintain databases (such as a key server) and the databases may even be shared between these protocols. This includes the possibility to declassify long-term secrets, e.g., to verify that a protocol is even secure if the intruder learns all old private keys. Both databases, shared databases, and declassification are considerable generalizations over the existing results.

Like [1] our result links the parallel compositionality result with a typing result such as the result of [17], i.e., essentially requiring that all messages of different meaning have a distinguishable form. Under this requirement it is sound to restrict the intruder model to using only well-typed messages which greatly simplifies many related problems. While one may argue that such a typing result is not strictly necessary for composition, we believe it is good practice and also fits well with disjointness requirements of parallel composition. Moreover, many existing protocols already satisfy our typing requirement, since, unlike tagging schemes, this does not require a modification of a protocol as long as there is some way to distinguish messages of different meaning.

There are other types of compositionality results for sequential and vertical composition, where the protocols under composition do build upon each other, e.g., one protocol establishes a key that is then subsequently used by another protocol [2,10,8,6,19,11]. This requires that one protocol satisfies certain properties (e.g. that the key exchange is authenticated and secret) for the other protocol to rely on. Our composition result allows for such sequential composition through shared databases: a key exchange protocol may enter keys into a shared set, and the other protocol consumes these keys. Thus our concept of sharing sets generalizes the interactions between otherwise independent protocols, and one only needs to think about the interface (e.g., only authenticated, fresh, secret keys can be entered into the database; they can only be used once). Moreover, we believe that sets are also a nice way to talk about this interaction.

There are several interesting aspects of compositionality that our result does not cover, for instance, [7] discusses the requirements for composing password-based protocols, and [3] investigates conditions under which privacy properties can be preserved under protocol composition.

So far, compositionality results are solely "paper-and-pencil" proofs. The proof arguments are often quite subtle, e.g., given an attack where the intruder learned a nonce from one protocol and uses it in another protocol, one has to prove that the attack does not rely on this, but would similarly work for distinct nonces. It is not uncommon that parts of such proofs are a bit sketchy with the danger of overlooking some subtle problems as for instance described in [16]. For this reason, we have formalized the compositionality result—on the level of ordinary constraints—in the proof assistant Isabelle/HOL [20], extending the formalization of [16,17], giving the extremely high correctness guarantee of machine-checked proofs. To our knowledge, this work is the first such formalization of a compositionality result in a proof assistant, with the notable exception of a study in Isabelle/HOL of compositional reasoning on concrete protocols [5].

Finally, all the works discussed so far are based on a black-box model of cryptography. There are several cryptographic frameworks for composition, most notably universal composability, reactive simulatability [4], and [18]. Considering the real cryptography makes compositional reasoning several orders of magnitude harder than abstract cryptography models. It is an intriguing question whether stateful protocol composition can be lifted to the full cryptographic level.

Acknowledgments. This work was supported by the Sapere-Aude project "Composec: Secure Composition of Distributed Systems", grant 4184-00334B of the Danish Council for Independent Research. We thank Luca Viganò for helpful comments and discussions.

References

1. Almousa, O., Mödersheim, S., Modesti, P., Viganò, L.: Typing and compositionality for security protocols: A generalization to the geometric fragment. In: ESORICS. pp. 209–229 (2015)

2. Andova, S., Cremers, C.J.F., Gjøsteen, K., Mauw, S., Mjølsnes, S.F., Radomirović, S.: A framework for compositional verification of security protocols. Inf. Comput. **206**(2-4), 425–459 (2008)
3. Arapinis, M., Cheval, V., Delaune, S.: Composing security protocols: From confidentiality to privacy. In: Focardi, R., Myers, A. (eds.) POST. pp. 324–343. Springer Berlin Heidelberg, Berlin, Heidelberg (2015). http://doi.org/10.1007/978-3-662-46666-7_17
4. Backes, M., Pfitzmann, B., Waidner, M.: The reactive simulatability (RSIM) framework for asynchronous systems. Inf. Comput. **205**(12), 1685–1720 (2007)
5. Butin, D.F.: Inductive analysis of security protocols in Isabelle/HOL with applications to electronic voting. Ph.D. thesis, Dublin City University (Nov 2012)
6. Cheval, V., Cortier, V., Warinschi, B.: Secure composition of PKIs with public key protocols. In: CSF. pp. 144–158 (Aug 2017). http://doi.org/10.1109/CSF.2017.28
7. Chevalier, C., Delaune, S., Kremer, S., Ryan, M.D.: Composition of password-based protocols. Formal Methods in System Design **43**(3), 369–413 (Dec 2013). http://doi.org/10.1007/s10703-013-0184-6
8. Ştefan Ciobâcă, Cortier, V.: Protocol composition for arbitrary primitives. In: CSF. pp. 322–336. IEEE (2010)
9. Cortier, V., Delaune, S.: Safely composing security protocols. Formal Methods in System Design **34**(1), 1–36 (2009). http://doi.org/10.1007/s10703-008-0059-4
10. Escobar, S., Meadows, C.A., Meseguer, J., Santiago, S.: Sequential protocol composition in Maude-NPA. In: ESORICS. pp. 303–318 (2010)
11. Groß, T., Mödersheim, S.: Vertical protocol composition. In: CSF. pp. 235 –250 (2011). http://doi.org/10.1109/CSF.2011.23
12. Guttman, J.D.: Cryptographic Protocol Composition via the Authentication Tests. In: FOSSACS. pp. 303–317. Springer (2009)
13. Guttman, J.D., Thayer, F.J.: Protocol independence through disjoint encryption. In: CSFW. pp. 24–34. IEEE (2000)
14. Heintze, N., Tygart, J.D.: A model for secure protocols and their compositions. In: Security and Privacy. pp. 2–13 (May 1994). http://doi.org/10.1109/RISP.1994.296596
15. Hess, A.V., Mödersheim, S.A., Brucker, A.D.: Stateful protocol composition (extended version). Tech. rep., DTU Compute (2018), Technical Report-2018-03. https://people.compute.dtu.dk/samo/
16. Hess, A.V., Mödersheim, S.: Formalizing and proving a typing result for security protocols in Isabelle/HOL. In: CSF (2017)
17. Hess, A.V., Mödersheim, S.: A typing result for stateful protocols. In: CSF (2018)
18. Küsters, R., Tuengerthal, M.: Composition theorems without pre-established session identifiers. In: CCS. pp. 41–50. ACM, New York, NY, USA (2011). http://doi.org/10.1145/2046707.2046715
19. Mödersheim, S., Viganò, L.: Secure pseudonymous channels. ESORICS pp. 337–354 (2009)
20. Nipkow, T., Paulson, L.C., Wenzel, M.: Isabelle/HOL - A Proof Assistant for Higher-Order Logic, Lecture Notes in Computer Science, vol. 2283. Springer (2002). http://doi.org/10.1007/3-540-45949-9