A Sound Decision Procedure for the Compositionality of Secrecy[⋆]

Martín Ochoa[1,3], Jan Jürjens[1,2], and Daniel Warzecha[1]

[1] Software Engineering, TU Dortmund, Germany
[2] Fraunhofer ISST, Germany
[3] Siemens AG, Germany
firstname.lastname@cs.tu-dortmund.de

Abstract. The composition of processes is in general not secrecy preserving under the Dolev-Yao attacker model. In this paper, we describe an algorithmic decision procedure which determines whether the composition of secrecy preserving processes is still secrecy preserving. As a case-study we consider a variant of the TLS protocol where, even though the client and server considered separately would be viewed as preserving the secrecy of the data to be communicated, its composition to the complete protocol does not preserve that secrecy. We also show results on tool support that allows one to validate the efficiency of our algorithm for multiple compositions.

1 Introduction

The question of compositional model-checking [5] is crucial for achieving scalable verification of systems. Moreover, compositionality of secure protocols can cause unforeseen problems (see for example problems on the SAML based single-sign-on used by Google in [3]). Although this question has been studied extensively in the literature, in this paper we propose a novel methodology to specify protocols such that given a finite set of session variables, compositionality is decidable. This is equivalent to restrict the analysis of processes to finitely many runs. Indeed vulnerabilities in authentication protocols have been shown to be limited to finitely many parallel instantiations [14]. Technically, our analysis generates finite *dependency trees* that can be stored for further deciding on future compositions. The process of merging such trees can be shown to be empirically more efficient than re-analysing the composition from scratch, and constitutes our central contribution. Moreover, this process is relatively sound and complete with respect to the First Order Logic analysis of [10].

To validate our approach we have implemented our algorithm as an extension to the UMLsec Tool Suite. This validates the usability of the approach in a formally sound Software Development process, and has allowed us to measure

[⋆] This research was partially supported by the MoDelSec Project of the DFG Priority Programme 1496 "Reliably Secure Software Systems – RS^3" and the EU project NESSoS (FP7 256890).

$E ::=$	expression
d	data value ($d \in \mathcal{D}$)
N	unguessable value ($N \in \textbf{Secret}$)
K	key ($K \in \textbf{Keys}$)
$\mathsf{inp}(c)$	input on channel c ($c \in \mathsf{Channels}$)
x	variable ($x \in \textbf{Var}$)
$E_1 :: E_2$	concatenation
$\{E\}_e$	encryption ($e \in \textbf{Enc}$)
$\mathcal{D}ec_e(E)$	decryption ($e \in \textbf{Enc}$)
$\mathcal{S}ign_e(E)$	signature creation ($e \in \textbf{Enc}$)
$\mathcal{E}xt_e(E)$	signature extraction ($e \in \textbf{Enc}$)

Fig. 1. Grammar for simple expressions in the Domain-Specific Language

the efficiency of our approach given the derivation trees for up to 500 small components (amounting to about 1000 messages).

This paper is organized as follows: Sect. 2 presents some preliminaries about stream processing functions, composition and secrecy. Sect. 3 describes the main verification strategy, whereas Sect. 4 shows its application to an insecure variant of TLS. Sect. 5 reports on efficiency of the decision procedure compared to re-verification. Finally, Related Work is discussed on Sect. 6 and we conclude with Sect. 7.

2 Preliminaries

In [10] the underlying process model used to model component communication is based on Broy's stream-processing functions [4]. A *process* is of the form $P = (I, O, L, (p_c)_{c \in O \cup L})$ where $I \subseteq \mathsf{Channels}$ is called the set of its *input channels* and $O \subseteq \mathsf{Channels}$ the set of its *output channels* and where for each $c \in \tilde{O} \overset{\text{def}}{=} O \cup L$, p_c is a closed program with input channels in $\tilde{I} \overset{\text{def}}{=} I \cup L$ (where $L \subseteq \mathsf{Channels}$ is called the set of *local channels*). From inputs on the channels in $\tilde{I}$ at a given point in time, p_c computes the output on the channel c. Each channel defines thus a stream processing function based on its input variables allowing for a rigorous notion of sequential composition, which is denoted by $\otimes$. For cryptographic protocol analysis, the programs are specified in a domain specific language defined by the expressions as in Fig. 1 and a simple programming language with non-deterministic choice (where loops can be modelled by using local channels). To proceed with the Dolev-Yao secrecy analysis, one defines rules to translate programs to first-order logic formulas. With the predicate $\mathsf{knows}(E)$ we can express the fact that an adversary may know an expression E during the execution of the protocol, therefore it models the *man in the middle*. For example, if-constructs are translated by the following formula:

$$\begin{aligned}\phi(\textit{if } E = E' \textit{ then } p \textit{ else } p') = \forall i_1, \ldots, i_n . \big[& \mathsf{knows}(i_1) \wedge \ldots \wedge \mathsf{knows}(i_n) \Rightarrow \\ & [E(i_1, \ldots, i_n) = E'(i_1, \ldots, i_n) \Rightarrow \phi(p)] \\ & \wedge [E(i_1, \ldots, i_n) \neq E'(i_1, \ldots, i_n) \Rightarrow \phi(p')]\big]\end{aligned}$$

To verify the secrecy of data $s \in \textbf{Secret}$, one then has to check whether the adversary can derive $\mathsf{knows}(s)$, given the formulas that arise from the evaluation ϕ of the single program constructs and the following axioms:

$$\forall E_1, E_2.$$
$$\big[\mathsf{knows}(E_1) \wedge \mathsf{knows}(E_2) \;\Rightarrow\; \mathsf{knows}(E_1 :: E_2) \wedge \; \mathsf{knows}(\{E_1\}_{E_2}) \wedge \; \mathsf{knows}(\mathcal{S}ign_{E_2}(E_1))\big]$$
$$\wedge \; \big[\mathsf{knows}(E_1 :: E_2) \;\Rightarrow\; \mathsf{knows}(E_1) \wedge \mathsf{knows}(E_2)\big]$$
$$\wedge \; \big[\mathsf{knows}(\{E_1\}_{E_2}) \wedge \mathsf{knows}(E_2^{-1}) \;\Rightarrow\; \mathsf{knows}(E_1)\big]$$
$$\wedge \; \big[\mathsf{knows}(\mathcal{S}ign_{E_2^{-1}}(E_1)) \wedge \mathsf{knows}(E_2) \;\Rightarrow\; \mathsf{knows}(E_1)\big]$$

The conjunction of the formulae ϕ for all channel programs of a process is called ψ. In the following, we will discuss composition at the level of this First Order Logic translation and not at the underlying stream processing function level because the FOL translation contains implicitly all the possible actions an adversary process could perform (defined by the structural formulas). Moreover, and adversary that completely controls the communication channels between processes, might act as an adaptor creating unforeseen compositions between input and output channels. Therefore we want to approximate the knowledge an adversary can gain given all possible outputs of the processes (considering all possible well-formed inputs).

3 Decision procedure

If we assume that both P and P' preserve the secrecy of the data value s, our goal is to show a procedure so that we can decide if $\psi(P \otimes P') \nvdash \mathsf{knows}(s)$. In general this does not hold. For example consider a process P which outputs $\{s\}_K$ and a process P' which outputs K^{-1}. Independently this both processes preserve the secrecy of s, but when composed an adversary could trivially compute s. To achieve this, we will construct proof artifacts on each single process called *derivation trees*. Moreover, in order ensure that this trees are finite, we will require that the number of *keys* and *nonces* are also finite and that the conditions in the "if" constructs of the process programs admit only variables that are of type *key* or *nonce*. This will imply the decidability of our approach.

Definition 1 (Subterm). *We say that a symbol* x *is a subterm of the symbol* T *and denote it* $\mathsf{x} \,\hat{\in}\, \mathsf{T}$ *when one of the following holds:*

x=T
T=$\{\mathsf{T}'\}_{\mathsf{K}}$ *and* $\mathsf{x} \,\hat{\in}\, \mathsf{T}'$
T=$\mathcal{S}ign_{\mathsf{K}}\{\mathsf{T}'\}$ *and* $\mathsf{x} \,\hat{\in}\, \mathsf{T}'$
T= h::k *and* $\mathsf{x} \,\hat{\in}\, \mathsf{h}$ *or* $\mathsf{x} \,\hat{\in}\, \mathsf{k}$

Example $\mathsf{s} \,\hat{\in}\, \{\mathsf{s}\}_{\mathsf{K}}$ but is not true that $\mathsf{K} \,\hat{\in}\, \{\mathsf{s}\}_{\mathsf{K}}$. We denote this by $\mathsf{K} \,\hat{\notin}\, \{\mathsf{s}\}_{\mathsf{K}}$. This means that an adversary could potentially compute s from $\{\mathsf{s}\}_{\mathsf{K}}$ using the structural formulas with the necessary previous knowledge, but he could not compute K.

Definition 2 (Inverse). *Let* $\mathsf{x} \,\hat{\in}\, \mathsf{J}$*. We define the cryptographic inverse of a symbol* J *with respect to* x *and denote it* $\mathsf{J}^{-1}(\mathsf{x})$ *in the following way:*

$$\mathsf{x}^{-1}(\mathsf{x}) = \epsilon$$

If J=$\mathsf{h::k}$ *and* $\mathsf{x}\,\hat{\notin}\,\mathsf{h}$ *then* $\mathsf{J}^{-1}(\mathsf{x})$=$\mathsf{k}^{-1}(\mathsf{x})$
If J=$\mathsf{h::k}$ *and* $\mathsf{x}\,\hat{\notin}\,\mathsf{k}$ *then* $\mathsf{J}^{-1}(\mathsf{x})$=$\mathsf{h}^{-1}(\mathsf{x})$
If J=$\mathsf{h::k}$ *and* $\mathsf{x}\,\hat{\in}\,\mathsf{k}$, $\mathsf{x}\,\hat{\in}\,\mathsf{h}$ *then* $\mathsf{J}^{-1}(\mathsf{x}) = \mathsf{and}(\mathsf{h}^{-1}(\mathsf{x}), \mathsf{k}^{-1}(\mathsf{x}))$
If J=$\{\mathsf{J}'\}_{\mathsf{K}}$ *or* J=$\mathcal{S}ign_{\mathsf{K}}\{\mathsf{J}'\}$ *then* $\mathsf{J}^{-1}(\mathsf{x}) = \mathsf{or}(\mathsf{J}'^{-1}(\mathsf{x}), K^{-1})$.

Example Let $\mathsf{J} = \{\{\mathsf{s}\}_{\mathsf{K}_1}\}_{\mathsf{K}_2}$. Then $\mathsf{J}^{-1}(\mathsf{s}) = \mathsf{or}(K_1^{-1}, K_2^{-1})$ which we will interpret later as "to preserve the secrecy s we need to preserve either K_1^{-1} or K_2^{-1}".

Let $\psi(P)$ be the first order logic formula associated to P. We define $\bar{\psi}(P)$ to be the set of instantiated formulas of $\psi(P)$ with all possible values satisfying the constraints in $\psi(P)$. Since we require that all constraints only contain variables of type *key* or *nonce*, and that the respective sets are finite, then $\bar{\psi}(P)$ is also finite. It is possible to show by induction on the program constructs that $\bar{\psi}(P)$ consists of formulas F_i of the form $\mathsf{knows}(\mathsf{E}_i) \Rightarrow \mathsf{knows}(\mathsf{J}_i)$ for closed expressions E_i and J_i. Let $\mathsf{Pres(x,P)}$ be the following inductively defined predicate:

$$\begin{aligned}
&[(\ \forall \mathsf{F}_i \in \bar{\psi}(P)\ \mathsf{x}\,\hat{\notin}\, J_i\ ) \Rightarrow \mathsf{Pres(x,P)})\\
&\wedge\ (\ \ \forall \mathsf{F}_i \in \bar{\psi}(P)\ (\mathsf{x}\,\hat{\in}\,\mathsf{J}_i) \Rightarrow ((\mathsf{Pres}(\mathsf{E}_i,\mathsf{P}) \vee \mathsf{Pres}({\mathsf{J_i}}^{-1}(\mathsf{x}),\mathsf{P}))\\
&\wedge\ (\ \ (\mathsf{x} = \{\mathsf{x}'\}_{\mathsf{K}} \vee \mathsf{x} = \mathcal{S}ign_{\mathsf{K}}\{\mathsf{x}'\}) \Rightarrow (\mathsf{Pres(x',P)} \vee \mathsf{Pres(K,P)})\\
&\wedge\ ((\mathsf{x} = \mathsf{h::k} \Rightarrow (\mathsf{Pres(h,P)} \vee \mathsf{Pres(k,P)})\\
&\wedge\ ((\mathsf{x} = \mathsf{and(h,k)} \Rightarrow (\mathsf{Pres(h,P)} \wedge \mathsf{Pres(k,P)})\\
&\wedge\ ((\mathsf{x} = \mathsf{or(h,k)} \Rightarrow (\mathsf{Pres(h,P)} \vee \mathsf{Pres(k,P)})\,]\\
&\Rightarrow \mathsf{Pres(x,P)}
\end{aligned}$$

and $\neg\mathsf{Pres}(\epsilon,\mathsf{P})$. If we can not derive $\mathsf{Pres(x,P)}$ for some x, it follows $\neg\mathsf{Pres(x,P)}$.

Theorem 1. *If it is possible to derive* Pres(x,P) *(conversely* $\neg$Pres(x,P)*) then* $\psi(P) \nvdash \mathsf{knows}(x)$ *(*$\psi(P) \vdash \mathsf{knows}(x)$*).*

Proof idea In case $\neg\mathsf{Pres}(\epsilon,\mathsf{P})$ since $\mathsf{knows}(\epsilon) \in \bar{\psi}(P)$ for all P. If $\forall \mathsf{F}_i \in \bar{\psi}(P)$ $\mathsf{x}\,\hat{\notin}\, J_i$ that means that there is no formula in $\bar{\psi}(P)$ containing x in a conclusive position, and therefore there is no way to derive $\mathsf{knows(x)}$ from the structural formulas. Now assume it is possible to derive $\mathsf{Pres(x,P)}$. We have already covered the base cases so we can assume that $\psi(P) \nvdash \mathsf{knows}(y)$ for all the $\mathsf{Pres(y,P)}$ $\mathsf{y} \neq \mathsf{x}$ needed in the precondition. Since in this formulas all the cases where we could apply the Structural Formulas are covered, it is impossible to derive $\mathsf{knows(x)}$. The case $\neg\mathsf{Pres(x,P)}$ is similar. □

Notice that the converse does not hold, that is $\psi(P) \nvdash \mathsf{knows}(x)$ does not mean we can derive $\mathsf{Pres(x,P)}$, because for some pathological cases we will have an infinite loop, for example for $\bar{\psi}(P) = \mathsf{knows(x)} \Rightarrow \mathsf{knows(x)}$. It is although easy to detect and avoid this loops in a machine implementation of the preservation predicate by running an initial check on the formulas. This makes the verification of the $\mathsf{Pres(x,P)}$ predicate sound and complete with respect to the First Order Logic embedding of the process programs.

As we derive $\mathsf{Pres(s,P)}$ for some symbol s and formulas P, we can build a *derivation tree* consisting of the symbols we need to consider to be able to conclude the preservation status of s. If we generate and store the derivation tree for every symbol x appearing in a process P in a relevant position (that

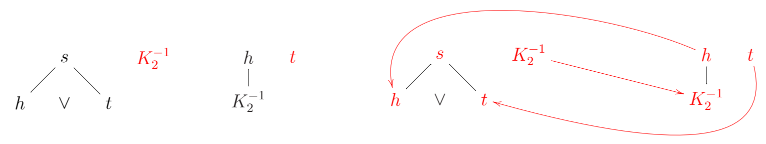

Fig. 2. Processes P and P' before and after composition

is $x \mathrel{\hat{\in}} J_i$ for some i), then we can decide whether the composition with process P' will preserve the secrecy of any given symbol if we also have the derivation trees for P'. Consider for example $P = (\{s\}_{h::t}, K_2^{-1})$ and $P' = (\{\mathsf{h}\}_{\mathsf{K}_2}, t)$. The symbol dependency trees of both process are depicted in Fig. 2 (the symbols in red are the ones which secrecy is compromised). Clearly both processes preserve separately the secrecy of s. To see if the composition also does, we update the information on the tree of s by checking whether the truth values of h and t are altered by the composition as depicted in Fig. 2.

4 An insecure variant of the TLS protocol

As an example we apply our approach to a variant of TLS [2] (not the version of TLS in current use) that does not preserve secrecy as a composition of the client C, the server S and the authority CA . We have that the predicate for C and S after the programs are translated to F.O.L are (for details on the translation see [10]) :

$$\psi(\mathsf{C}) = \mathsf{knows}(N_C :: K_C :: \mathcal{S}ign_{\mathsf{K}_\mathsf{C}^{-1}}\{\mathsf{C} :: \mathsf{K_C}\}) \wedge (\mathsf{knows}(s_2) \wedge \mathsf{knows}(s_3) \Rightarrow \mathsf{knows}(\{m\}_y))$$
$$\psi(\mathsf{S}) = \mathsf{knows}(c_1) \wedge \mathsf{knows}(c_2) \wedge \mathsf{knows}(c_3) \;\Rightarrow \mathsf{knows}(N_S :: \{\ \mathcal{S}ign_{\mathsf{K}_\mathsf{S}^{-1}}\{\mathsf{k_{CS}} :: \mathsf{c_1}\}\ \}_{c_2})$$

where $\{s_3\}_{K_{CA}} = S :: x \wedge \{Dec_{K_C^{-1}}(s_2)\}_x = y :: N_C$ and $\{c_3\}_{c_2} = C :: c_2$ where $\mathsf{key}(c_2), \mathsf{key}(x)$ and $\mathsf{key}(y)$. The set of keys is $\mathsf{Keys} = \{K_A, K_A^{-1}, k_{CS}, k_A, K_C, K_C^{-1}, K_S, K_S^{-1}, K_{CA}, K_{CA}^{-1}\}$ where k_{CS} and k_A are symmetric keys. The nonces are $\mathsf{Nonces} = \{N_C, N_S, N_A\}$. We assume that the authority CA has already distributed certificates to all parties and that the adversary is in possession of this information: $\mathsf{knows}(K_{CA}) \wedge \mathsf{knows}(\mathcal{S}ign_{\mathsf{K}_\mathsf{CA}^{-1}}\{\mathsf{S} :: \mathsf{K_S}\}) \wedge \mathsf{knows}(\mathcal{S}ign_{\mathsf{K}_\mathsf{CA}^{-1}}\{\mathsf{A} :: \mathsf{K_A}\})$.

We further assume that an adversary posses a key pair $\mathsf{knows}(K_A) \wedge \mathsf{knows}(K_A^{-1})$. Now we show that $\mathsf{C} \otimes \mathsf{S}$ does not preserve the secrecy of m although C and S separately do. First of all, in order to be able to apply our approach and generate the dependency tree, we have to solve the constraints for all the processes involved. So we have:

$$\bar{\psi}(\mathsf{C}) = \ \mathsf{knows}(N_C :: K_C :: \mathcal{S}ign_{\mathsf{K}_\mathsf{C}^{-1}}\{\mathsf{C} :: \mathsf{K_C}\})$$
$$\wedge\ (\mathsf{knows}(\{\ \mathcal{S}ign_{\mathsf{x}^{-1}}\{\mathsf{y} :: \mathsf{N_C}\}\ \}_{K_C}) \wedge \mathsf{knows}(\ \mathcal{S}ign_{\mathsf{K}_\mathsf{CA}^{-1}}\{\mathsf{S} :: \mathsf{x}\}\ ) \Rightarrow \mathsf{knows}(\{m\}_y))$$

where $x \in \{K_C, K_S, K_A\}$ (the public keys) and $y \in \{k_A, k_{CS}\}$ (the symmetric keys). We do not explicit the whole dependency tree for C but we note that the secrecy of m is preserved because: if $y = k_{CS}$ the adversary does not have knowledge of k_{CS}; if $y = k_A$ the adversary would need knowledge of $\mathcal{S}ign_{\mathsf{x}^{-1}}\{\mathsf{k_A} :: \mathsf{N_C}\}$

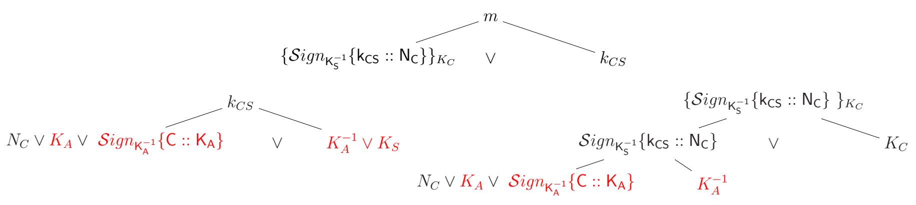
m
$\{\mathcal{S}ign_{\mathsf{K_S^{-1}}}\{\mathsf{k_{CS}} :: \mathsf{N_C}\}\}_{K_C}$
$\vee$
k_{CS}
k_{CS}
$N_C \vee K_A \vee \mathcal{S}ign_{\mathsf{K_A^{-1}}}\{\mathsf{C} :: \mathsf{K_A}\}$
$\vee$
$K_A^{-1} \vee K_S$
$\{\mathcal{S}ign_{\mathsf{K_S^{-1}}}\{\mathsf{k_{CS}} :: \mathsf{N_C}\}\ \}_{K_C}$
$\mathcal{S}ign_{\mathsf{K_S^{-1}}}\{\mathsf{k_{CS}} :: \mathsf{N_C}\}$
$\vee$
K_C
$N_C \vee K_A \vee \mathcal{S}ign_{\mathsf{K_A^{-1}}}\{\mathsf{C} :: \mathsf{K_A}\}$
K_A^{-1}

Fig. 3. Partial trees for m in C and for k_{CS} and $\{\ \mathcal{S}ign_{\mathsf{K_S^{-1}}}\{\mathsf{k_{CS}} :: \mathsf{N_C}\}\ \}_{K_C}$ in S

and $\mathcal{S}ign_{\mathsf{K_{CA}^{-1}}}\{\mathsf{S} :: \mathsf{x}\}$ for some x. Since he only knows $\mathcal{S}ign_{\mathsf{K_{CA}^{-1}}}\{\mathsf{S} :: \mathsf{K_S}\}$ then $x = K_S$. In that case to gain knowledge of $\mathcal{S}ign_{\mathsf{K_S}^{-1}}\{\mathsf{k_A} :: \mathsf{N_C}\}$ he needs to posses ${K_S}^{-1}$ which he does not. In Fig. 3 we depict partially this dependency tree for the case $y = k_{CS}$, $x = K_S$. Now, the instantiated formulas for S are:

$$\bar{\psi}(\mathsf{S}) = \mathsf{knows}(c_1) \wedge \mathsf{knows}(c_2) \wedge \mathsf{knows}(\mathcal{S}ign_{\mathsf{c_2^{-1}}}\{\mathsf{C} :: \mathsf{c_2}\})$$
$$\Rightarrow \mathsf{knows}(N_S :: \{\ \mathcal{S}ign_{\mathsf{K_S^{-1}}}\{\mathsf{k_{CS}} :: \mathsf{c_1}\}\ \}_{c_2})$$

with $c_1 \in \{N_S, N_C, N_A\}$, $c_2 \in \{K_C, K_S, K_A\}$. The secrecy of m is preserved in S simply because m is not a subterm of any formula in S.

To see why the composition fails to preserve secrecy, we illustrate (partially) the dependency trees of k_{CS} and $\{\ \mathcal{S}ign_{\mathsf{K_S^{-1}}}\{\mathsf{k_{CS}} :: \mathsf{c_1}\}\ \}_{c_2}$ in case $c_1 = N_C$ and $c_2 = K_A$ in Fig. 3. In fact, since C leaks N_C and K_C, k_{CS} turns to be not secret after composition in the tree of S. This also modifies the secrecy status of $\{\ \mathcal{S}ign_{\mathsf{K_S^{-1}}}\{\mathsf{k_{CS}} :: \mathsf{N_C}\}\ \}_{K_C}$ which results in a secrecy violation for m after updating the tree of C. We have performed a similar analysis for a fix to this protocol proposed in [10] where the composition preserves secrecy but for space reasons we do not explicit the details here.

5 Validation and Efficiency

We have implemented our approach as an extension to the UMLsec tool support [4]. That is, we can extract the protocol specification from a sequence diagram using the DSL described in Sect. 2 and translate it to First Order Logic. Since by construction each guard accepts only finitely many messages (depending on the set of keys and nonces), we can build finite dependency trees for all relevant symbols by means of a properly generated prolog program.

Reasoning about composition amounts then to join the trees from two processes. Therefore, we can at least avoid to recompute the constraint solving for the single processes. We have conducted experiments to measure the time of the composition, and compare it to the overall process of constraint-solving and prolog generation as depicted in Table 1. The first column contains the number of messages for a single session of the composition and the second column corresponds to the number of composed processes. The third column is the time

[4] http://www-jj.cs.tu-dortmund.de/jj/umlsec/

# Messages	# Compositions	Generation trees (ms)	Composition (ms)
11	5	3660	47
21	10	6214	88
31	15	9323	114
51	25	15406	198
101	50	31730	401
501	250	182771	1948
1001	500	375474	3963

Table 1. Execution times of our experiment

in ms. needed to extract the FOL formulas from the UML diagram and generate the derivation trees. The last column is the time needed for deciding the composition given the single derivation trees. In other words, if we would have a repository of 500 processes that by themselves are secrecy preserving, and we would like to check whether the composition of any 5 of them is also secrecy preserving, it would be highly desirable if we could use the existing results as opposed to re-verify from scratch every time.

6 Related Work

Overviews of applications of formal methods to security protocols can be found for example in [1, 12], some examples in [11, 13]. The question of protocol composition has been studied by different authors. More prominently, Datta, Mitchell et al. [6] have defined the PCL (Protocol Composition Logic), aimed at the verification of security protocol by re-using proofs of sub-protocols using a Hoare-like logic, focusing on authenticity. Guttmann [7] gives results about protocol composition at a lower abstraction level, considering unstructured 'blank slots' and compound keys that result from hashes of other messages. Jürjens [9] has explored the question of composability aiming at given sufficient conditions under which composition holds. Stoller [14] has computed bounds of parallel executions that could compromise the authenticity of protocols. These approaches aim at giving at a collection of theorems that if satisfied by two protocols in a composition, ensure a given property. One must show (by using a theorem prover, or by hand) that some properties are satisfied by both protocols like *disjointness* in [8]. Our approach differs from this assume/guarantee reasoning in that we efficiently check whether the composition harms secrecy given pre-computed 'proof artifacts': the dependency trees. In other words, we give accurate results about compositions (that are equivalent to re-verification), by amortizing the cost of verification at an initial phase.

7 Conclusions

The problem of compositionality is of particular importance for software development when the security of reusable components has been established, since guarantees about the composition are needed. The decision procedure should also scale efficiently to be of practical use, and most of all, sound. We have shown that our procedure is sound and complete with respect to previous work on First Order Logic protocol verification. This comes at the price of an initial verification of the single components that considers all the possible acceptable messages. Nevertheless, this is compensated when it comes to decide compositionality with an arbitrary process for which the same process has also taken place, since this can be done very efficiently, as we have empirically tested. There are different ways in which this work could be further extended. On the one hand, one can further explore the efficiency of the approach, for example by formally deriving its complexity . On the other hand, one could extend the approach to cope with the preservation of other security properties like authenticity.

References

1. M. Abadi. Security protocols and their properties. In F. Bauer and R. Steinbrüggen, editors, *Foundations of Secure Computation*, pages 39–60. IOS Press, Amsterdam, 2000. 20th International Summer School, Marktoberdorf, Germany.
2. G. Apostolopoulos, V. Peris, and D. Saha. Transport layer security: How much does it really cost? In *Proceedings of the IEEE Infocom*, pages 717–725, 1999.
3. A. Armando, R. Carbone, L. Compagna, J. Cuéllar, and M. L. Tobarra. Formal analysis of SAML 2.0 web browser single sign-on: breaking the SAML-based single sign-on for google apps. In V. Shmatikov, editor, *FMSE*, pages 1–10. ACM, 2008.
4. M. Broy. A logical basis for component-based systems engineering. In *Calculational System Design. IOS*. Press, 1999.
5. E. M. Clarke, D. E. Long, and K. L. Mcmillan. Compositional model checking. *Proceedings of the Fourth Annual Symposium on Logic in Computer Science (LICS '89)* IEEE Computer Society, 1989.
6. A. Datta, A. Derek, J. C. Mitchell, and A. Roy. Protocol composition logic (pcl). *Electronic Notes in Theoretical Computer Science*, 172(0):311 – 358, 2007. Computation, Meaning, and Logic: Articles dedicated to Gordon Plotkin.
7. J. D. Guttman. Cryptographic protocol composition via the authentication tests. In *Proceedings of the 12th International Conference on Foundations of Software Science and Computational Structures (FOSSACS '09)*, pages 303–317, Berlin, Heidelberg, 2009. Springer-Verlag.
8. J. D. Guttman, F. Javier, and F. J. T. Fábrega. Protocol independence through disjoint encryption. In *In Proceedings, 13th Computer Security Foundations Workshop. IEEE Computer*, pages 24–34. Society Press, 2000.
9. J. Jürjens. Composability of secrecy. In *Proceedings of the International Workshop on Information Assurance in Computer Networks: Methods, Models, and Architectures for Network Security*, MMM-ACNS '01, pages 28–38, London, UK, 2001. Springer-Verlag.
10. J. Jürjens. A domain-specific language for cryptographic protocols based on streams. *J. Log. Algebr. Program.*, 78(2):54–73, 2009.
11. G. Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. *Software Concepts and Tools*, 17(3):93–102, 1996.
12. C. Meadows. Open issues in formal methods for cryptographic protocol analysis. In *DARPA Information Survivability Conference and Exposition (DISCEX 2000)*, pages 237–250. IEEE Computer Society, 2000.
13. L. Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Computer Security*, 6(1–2):85–128, 1998.
14. S. D. Stoller. A bound on attacks on authentication protocols. *Proc. of the 2nd IFIP International Conference on Theoretical Computer Science: Foundations of Information Technology in the Era of Network and Mobile Computing*, 2001.