Safely Composing Security Protocols [⋆]

Véronique Cortier, Jérémie Delaitre, and Stéphanie Delaune

LORIA, CNRS & INRIA, project Cassis, Nancy, France

Abstract. Security protocols are small programs that are executed in hostile environments. Many results and tools have been developed to formally analyze the security of a protocol. However even when a protocol has been proved secure, there is absolutely no guarantee if the protocol is executed in an environment where other protocols, possibly sharing some common identities and keys like public keys or long-term symmetric keys, are executed.

In this paper, we show that whenever a protocol is secure, it remains secure even in an environment where arbitrary protocols are executed, provided each encryption contains some tag identifying each protocol, like e.g. the name of the protocol.

1 Introduction

Security protocols are small programs that aim at securing communications over a public network like the Internet. Considering the increasing size of networks and their dependence on cryptographic protocols, a high level of assurance is needed in the correctness of such protocols. The design of such protocols is difficult and error-prone; many attacks have been discovered even several years after the publication of a protocol. Consequently, there has been a growing interest in applying formal methods for validating cryptographic protocols and many results have been obtained. The main advantage of the formal approach is its relative simplicity which makes it amenable to automated analysis. For example, the secrecy preservation is co-NP-complete for a bounded number of sessions [19], and decidable for an unbounded number of sessions under some additional restrictions (e.g. [2, 5, 20]). Many tools have also been developed to automatically verify cryptographic protocols (e.g. [4]).

However even when a protocol has been proved secure for an unbounded number of sessions, against a fully active adversary that can intercept, block and send new messages, there is absolutely no guarantee if the protocol is executed in an environment where other protocols, possibly sharing some common identities and keys like public keys or long-term symmetric keys, are executed. This is however very likely to happen since a user connected to the Internet for example, usually uses simultaneously several protocols with the same identity. The interaction with the other protocols may dramatically damage the security

[⋆] This work has been partly supported by the RNTL project POSÉ and the ARA SSIA Formacrypt.

of a protocol. Consider for example the two following naive protocols.

$$P_1 : \quad A \to B : \{s\}_{\mathrm{pub}(B)} \qquad\qquad P_2 : \quad \begin{array}{l} A \to B : \{N_a\}_{\mathrm{pub}(B)} \\ B \to A : N_a \end{array}$$

In protocol P_1, the agent A simply sends a secret s encrypted under B's public key. In protocol P_2, the agent sends some fresh nonce to B encrypted under B's public key. The agent B acknowledges A's message by forwarding A's nonce. While P_1 executed alone easily guarantees the secrecy of s, even against an active adversary, the secrecy of s is no more guaranteed when the protocol P_2 is executed. Indeed, an adversary may use the protocol P_2 as an oracle to decrypt any message. More realistic examples illustrating interactions between protocols can be found in e.g. [15].

The purpose of this paper is to investigate sufficient and rather tight conditions for a protocol to be safely used in an environment where other protocols may be executed as well. Our main contribution is to show that whenever a protocol is proved secure when it is executed alone, its security is not compromised by the interactions with any other protocol, provided each protocol is given an identifier (e.g. the protocol's name) that should appear in any encrypted message. Continuing our example, let us consider the two slightly modified protocols.

$$P_1' : \quad A \to B : \{1, s\}_{\mathrm{pub}(B)} \qquad\qquad P_2' : \quad \begin{array}{l} A \to B : \{2, N_a\}_{\mathrm{pub}(B)} \\ B \to A : N_a \end{array}$$

Applying our result, we immediately deduce that P_1' can be safely executed together with P_2', without compromising the secrecy of s.

The idea of adding an identifier in encrypted messages is not novel. This rule is in the same spirit as those proposed in the paper of Abadi and Needham on prudent engineering practice for cryptographic protocols [1] (principle 10). The use of unique protocol identifiers is also recommended in [15, 7] and has also been used in the design of fail-stop protocols [13]. However, to the best of our knowledge, it has never been proved that it is sufficient for securely executing several protocols in the same environment. Note that some other results also use tags for different purposes. For instance, Blanchet uses tags to exhibit a decidable class [5] but his tagging policy is stronger since any two encrypted subterm in a protocol have to contain different tags.

A result closely related to ours is the one of Guttman and Thayer [14]. They show that two protocols can be safely executed together without damaging interactions, as soon as the protocols are "independent". The independence hypothesis requires in particular that the set of encrypted messages that the two protocols handle should be different. As in our case, this can be ensured by giving each protocol a distinguishing value that should be included in the set of encrypted messages that the protocol handles. However, the major difference with our result is that this hypothesis has to hold on any valid *execution* of the protocol. In particular, considering again the protocol P_2', an agent should not accept a message of the form $\{2, \{1, m\}_k\}_{\mathrm{pub}(B)}$ while he might not be able to decrypt the inside encryption and detect that it contains the wrong identifier. Another result has been recently obtained by Andova *et al.* for a broader class of composition operations and security properties [3]. In both cases, their result do not allow one to conclude when no typing hypothesis is assumed (that is, when agents are not required to check the type of each component of a message) or for protocols with cyphertext forwarding, that is, when agents have to forward unknown message components. Datta *et al.* (e.g. [12]) have also studied secure protocol composition in a more broader sense: protocols can be composed in parallel, sequentially or protocols may use other protocols as components. However, they do not provide any syntactic conditions for a protocol P to be safely executed in parallel with other protocols. For any protocol P' that might be executed in parallel, they have to prove that the two protocols P and P' satisfy each other invariants. Their approach is thus rather designed for component-based design of protocols. Our work is also related to those of Canetti *et al.* who, using a different approach, study universal composability of protocols [6]. They however require stronger security properties for their protocols to be composable.

Due to lack of space, proofs are omitted. They can be found in [10].

2 Models for Security Protocols

2.1 Syntax

Cryptographic primitives are represented by *function symbols*. More specifically, we consider the *signature* $\mathcal{F} = \{\mathrm{enc}, \mathrm{enca}, \mathrm{sign}, \langle\,\rangle, \mathrm{pub}, \mathrm{priv}\}$ together with arities of the form $\mathrm{ar}(f) = 2$ for the four first symbols and $\mathrm{ar}(f) = 1$ for the two last ones. The symbol $\langle\,\rangle$ represents the pairing function. The terms $\mathrm{enc}(m, k)$ and $\mathrm{enca}(m, k)$ represent respectively the message m encrypted with the symmetric (resp. asymmetric) key k. The term $\mathrm{sign}(m, k)$ represents the message m signed by the key k. The terms $\mathrm{pub}(a)$ and $\mathrm{priv}(a)$ represent respectively the public and private keys of an agent a. We fix an infinite set of *names* $\mathcal{N} = \{a, b \ldots\}$ among which we distinguish two particular names init and stop; and an infinite set of *variables* $\mathcal{X} = \{x, y \ldots\}$. The set of Terms is defined inductively by

$$\begin{array}{lll}
t ::= & & \text{term} \\
& \mid\ x & \text{variable } x \\
& \mid\ a & \text{name } a \\
& \mid\ f(a) & \text{application of symbol } f \in \{\mathrm{pub}, \mathrm{priv}\} \text{ on a name} \\
& \mid\ f(t_1, t_2) & \text{application of symbol } f \in \{\mathrm{enc}, \mathrm{enca}, \mathrm{sign}, \langle\,\rangle\}
\end{array}$$

As usual, we write $\mathit{vars}(t)$ (resp. $\mathrm{names}(t)$) for the set of variables (resp. names) occurring in t. A term is *ground* if and only if it has no variables. We write $\mathit{St}(t)$ for the set of *subterms* of a term t. For example, let $t = \mathrm{enc}(\langle a, b\rangle), k)$, we have that $\mathit{St}(t) = \{t, \langle a, b\rangle, a, b, k\}$. This notion is extended as expected to sets of terms. *Extended names* are names or terms of the form $\mathrm{pub}(a)$, $\mathrm{priv}(a)$. The set of *Extended names* associated to a term t, denoted $\mathrm{n}(t)$, is $\mathrm{n}(t) = \mathrm{names}(t) \cup \{\mathrm{pub}(t), \mathrm{priv}(t) \mid \mathrm{pub}(t) \text{ or } \mathrm{priv}(t) \in \mathit{St}(t)\}$. For example, we have that $\mathrm{n}(\mathrm{enc}(a, \mathrm{pub}(b))) = \{a, b, \mathrm{pub}(b), \mathrm{priv}(b)\}$. Substitutions are written $\sigma = \{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$ with $\mathrm{dom}(\sigma) = \{x_1, \ldots, x_n\}$. The substitution σ is *closed* if and only if all the t_i are ground. The application of a substitution σ to a term t is written $\sigma(t)$ or $t\sigma$.

$$\frac{T \vdash u \quad T \vdash v}{T \vdash \langle u, v \rangle} \qquad \frac{T \vdash u \quad T \vdash v}{T \vdash \mathrm{enc}(u, v)} \qquad \frac{T \vdash u \quad T \vdash v}{T \vdash \mathrm{enca}(u, v)} \qquad \frac{T \vdash u \quad T \vdash v}{T \vdash \mathrm{sign}(u, v)}$$

$$\frac{T \vdash \langle u, v \rangle}{T \vdash u} \qquad \frac{T \vdash \langle u, v \rangle}{T \vdash v} \qquad \frac{T \vdash \mathrm{enc}(u, v) \quad T \vdash v}{T \vdash u}$$

$$\frac{T \vdash \mathrm{enca}(u, \mathrm{pub}(v)) \quad T \vdash \mathrm{priv}(v)}{T \vdash u} \qquad \frac{T \vdash \mathrm{sign}(u, \mathrm{priv}(v))}{T \vdash u}\ \textit{(optional)} \qquad \frac{}{T \vdash u}\ u \in T$$

Fig. 1. Intruder deduction system.

2.2 Intruder Capabilities

The ability of the intruder is modelled by a deduction system described in Figure 1 and corresponds to the usual Dolev-Yao rules. The first line describes the *composition* rules. The two last lines describe the *decomposition* rules and the axiom. Intuitively, these deduction rules say that an intruder can compose messages by pairing, encrypting and signing messages provided he has the corresponding keys. Conversely, it can decompose messages by projecting or decrypting provided it has the decryption keys. For signatures, the intruder is also able to *verify* whether a signature $\mathrm{sign}(m, k)$ and a message m match (provided she has the verification key), but this does not give her any new message. That is why this capability is not represented in the deduction system. We also consider an optional rule that expresses that an intruder can retrieve the whole message from its signature. This property may or may not hold depending on the signature scheme, and that is why this rule is optional. Our results hold in both cases (that is, when the deduction relation $\vdash$ is defined with or without this rule).

A term u is *deducible* from a set of terms T, denoted by $T \vdash u$ if there exists a *proof*, i.e. a tree such that the root is $T \vdash u$, the leaves are of the form $T \vdash v$ with $v \in T$ (*axiom* rule) and every intermediate node is an instance of one of the rules of the deduction system. For instance, the term $\langle k_1, k_2 \rangle$ is deducible from the set $T_1 = \{\mathrm{enc}(k_1, k_2), k_2\}$.

2.3 Protocols

We consider protocols specified in a language similar to the one of [19] allowing parties to exchange messages built from identities and randomly generated nonces using public key, symmetric encryption and digital signatures. The individual behavior of each protocol participant is defined by a *role* describing a sequence of message receptions/transmissions, and a k-party protocol is given by k such roles.

Definition 1 (Roles and protocols). *The set* Roles *of roles for protocol participants is the set of sequences of the form* $(\mathsf{rcv}_1, N_1, \mathsf{snd}_1) \cdots (\mathsf{rcv}_\ell, N_\ell, \mathsf{snd}_\ell)$

where each element, called rule*, satisfies* $(\mathsf{rcv}_i, N_i, \mathsf{snd}_i) \in \mathsf{Terms} \times 2^{\mathcal{X}} \times \mathsf{Terms}$*, and for any variable,* $x \in \mathit{vars}(\mathsf{snd}_i)$ *implies* $x \in \bigcup_{1 \leq j \leq i} N_j \cup \mathit{vars}(\mathsf{rcv}_j)$*.*

The length *of a role is the number of elements in its sequence. A* k-party protocol *is a mapping* $\Pi : [k] \to \mathsf{Roles}$*, where* $[k] = \{1, 2, \ldots, k\}$*.*

The last condition ensures that each variable which appears in a sent term is either a nonce or has been introduced in a previously received message. The set of variables, names or extended names of a protocol is defined as expected, considering all the terms occurring in the role's specification.

The j^{th} role of a protocol Π is denoted by $(\mathsf{rcv}_1^j \xrightarrow{N_1^j} \mathsf{snd}_1^j) \cdots (\mathsf{rcv}_{k_j}^j \xrightarrow{N_{k_j}^j} \mathsf{snd}_{k_j}^j)$. It specifies the messages to be sent/received by the party executing the role: at step i, the j^{th} party expects to receive a message conformed to rcv_i^j, instantiate the variables of N_i^j with fresh names and returns the message snd_i^j. We assume the sets N_i^j to be pairwise disjoint. The special names init and stop will be used to specify that no message is expected or sent.

The *composition* of two protocols Π_1 and Π_2, denoted by $\Pi_1 \mid \Pi_2$ is simply the protocol obtained by the union of the roles of Π_1 and Π_2. If $\Pi_1 : [k_1] \to \mathsf{Roles}$ and $\Pi_2 : [k_2] \to \mathsf{Roles}$, then $\Pi = \Pi_1 \mid \Pi_2 : [k_1 + k_2] \to \mathsf{Roles}$ with $\Pi(i) = \Pi_1(i)$ for any $1 \leq i \leq k_1$ and $\Pi(k_1 + i) = \Pi_2(i)$ for any $1 \leq i \leq k_2$.

Example 1. Consider the famous Needham-Schroeder protocol [18].

$$\begin{array}{rl} A \to B: & \{N_a, A\}_{\mathrm{pub}(B)} \\ B \to A: & \{N_a, N_b\}_{\mathrm{pub}(A)} \\ A \to B: & \{N_b\}_{\mathrm{pub}(B)} \end{array}$$

The agent A sends to B his name and a fresh nonce (a randomly generated value) encrypted with the public key of B. The agent B answers by copying A's nonce and adds a fresh nonce N_B, encrypted by A's public key. The agent A acknowledges by forwarding B's nonce encrypted by B's public key. For instance, let a, b, and c be three agent names. The role $\Pi(1)$ corresponding to the first participant played by a talking to c is:

$$(\mathsf{init} \xrightarrow{\{X\}} \mathrm{enca}(\langle X, a\rangle, \mathrm{pub}(c))), (\mathrm{enca}(\langle X, x\rangle, \mathrm{pub}(a)) \xrightarrow{\emptyset} \mathrm{enca}(x, \mathrm{pub}(c))).$$

The role $\Pi(2)$ corresponding to the second participant played by b with a is:

$$(\mathrm{enca}(\langle y, a\rangle, \mathrm{pub}(b)) \xrightarrow{\{Y\}} \mathrm{enca}(\langle y, Y\rangle, \mathrm{pub}(a))), (\mathrm{enca}(Y, \mathrm{pub}(b)) \xrightarrow{\emptyset} \mathsf{stop}).$$

Note that, since our definition of role is not parametric, we have also to consider a role corresponding to the first participant played by a talking to b for example. If more agent identities need to be considered, then the corresponding roles should be added to the protocol. It has been shown however that two agents are sufficient (one honest and one dishonest) for proving security properties [8].

Clearly, not all protocols written using the syntax above are meaningful. In particular, some of them might not be *executable*. A precise definition of executability is not relevant for our result. We use instead a weaker hypothesis (see Section 3). In particular, our combination result also holds for non executable protocols that satisfy our hypothesis.

2.4 Constraint Systems

Constraint systems are quite common (see e.g. [19, 9, 11]) in modeling security protocols. They are used to specify secrecy preservation of security protocols under a particular, finite scenario. We recall here their formalism and we show in the next section that the secrecy preservation problem for an *unbounded number of sessions* can be specified using (infinite) families of constraint systems.

Definition 2 (constraint system). *A* constraint system $\mathcal{C}$ *is either* $\perp$ *or a finite sequence of expressions* $(T_i \Vdash u_i)_{1\leq i\leq n}$, *called* constraints, *where each* T_i *is a non empty set of terms, called the* left-hand side *of the constraint and each* u_i *is a term, called the* right-hand side *of the constraint, such that:*

- $T_i \subseteq T_{i+1}$ *for every* i *such that* $1 \leq i < n$;
- *if* $x \in vars(T_i)$ *for some* i *then there exists* $j < i$ *such that* $x \in vars(u_j)$.

A solution *of* $\mathcal{C}$ *is a closed substitution* θ *such that for every* $(T \Vdash u) \in \mathcal{C}$, *we have that* $T\theta \vdash u\theta$. *The empty constraint system is always satisfiable whereas* $\perp$ *denotes an unsatisfiable system.*

A constraint system $\mathcal{C}$ is usually denoted as a conjunction of constraints $\mathcal{C} = \bigwedge_{1\leq i\leq n}(T_i \Vdash u_i)$ with $T_i \subseteq T_{i+1}$, for all $1 \leq i < n$. The second condition in Definition 2 says that each time a variable occurs first in some right-hand side, it must not have occurred before in some left-hand side. The left-hand side of a constraint system usually represents the messages sent on the network.

2.5 Secrecy

We define the general secrecy preservation problem for an unbounded number of sessions, using infinite families of constraint systems. A role may be executed in several sessions, using different nonces at each session. Moreover, since the adversary may block, redirect and send new messages, all the sessions might be interleaved in many ways. This is captured by the notion of *scenario*.

Definition 3 (scenario). *A* scenario *for a protocol* $\Pi : [k] \rightarrow \mathsf{Roles}$ *is a sequence* $\mathsf{sc} = (r_1, s_1) \cdots (r_n, s_n)$ *such that* $1 \leq r_i \leq k$, $s_i \in \mathbb{N}$, *the number of identical occurrences of a pair* (r, s) *is smaller than the length of the role* r, *and whenever* $s_i = s_j$ *then* $r_i = r_j$.

The numbers r_i and s_i represent respectively the involved role and the session number. An occurrence of (r, s) in sc means that the role r of session s executes its next receive-send action. The condition on the number of occurrences of a pair ensures that such an action is indeed available. The last condition ensures that a session number is not reused on other roles. We say that $(r, s) \in \mathsf{sc}$ if (r, s) is an element of the sequence sc. Let $\Pi = \Pi_1 \mid \Pi_2$ be a protocol obtained by composition of Π_1 and Π_2 and let sc be a scenario for Π. The scenario $\mathsf{sc}|_{\Pi_1}$ is simply the sequence obtained from sc by removing any element (r, s) where r is a role of Π_2. Given a scenario, we can define a sequence of rules that corresponds to the sequence of expected and sent messages.

Definition 4. *Given a scenario* $\mathsf{sc} = (r_1, s_1) \cdots (r_n, s_n)$ *for a k-party protocol* Π, *the* sequence of rules $(u_1, v_1) \cdots (u_n, v_n)$ associated to sc *is defined as follows. Let* $\Pi(j) = (\mathsf{rcv}_1^j \xrightarrow{N_1^j} \mathsf{snd}_1^j) \cdots (\mathsf{rcv}_{k_j}^j \xrightarrow{N_{k_j}^j} \mathsf{snd}_{k_j}^j)$ *for* $1 \leq j \leq k$. *Let* $p_i = \#\{(r_j, s_j) \in \mathsf{sc} \mid j \leq i, r_j = r_i\}$, *i.e. the number of previous occurrences in* sc *of the role* r_i. *We have* $p_i \leq k_{r_i}$ *and* $(u_i, v_i) = (\mathsf{rcv}_{p_i}^{r_i}\sigma_{r_i,s_i}, \mathsf{snd}_{p_i}^{r_i}\sigma_{r_i,s_i})$, *where*

- $\mathrm{dom}(\sigma_{r,s}) = \bigcup_{1 \leq i \leq k_r} (N_i^r \cup vars(\mathsf{rcv}_i^r))$, *i.e. variables occurring in* $\Pi(r)$,
- $\sigma_{r,s}(x) = n_{x,s}$ *if* $x \in \bigcup_{1 \leq i \leq k_r} N_i^r$, *where* $n_{x,s}$ *is a name.*
- $\sigma_{r,s}(x) = x_s$ *otherwise, where* x_s *is a variable.*

We assume that names (resp. variables) with different indexes are pairwise different and also different from the names (resp. variables) occurring in Π,

We say that a protocol preserves the secrecy of a data if it preserves its secrecy for any scenario. In particular, the secrecy of the data must be preserved for any possible instances of its fresh values (e.g. nonces and keys).

Definition 5 (secrecy). *A protocol* Π preserves the secrecy *of a term* m for the initial knowledge T_0 *if for any scenario* sc *for* Π, *for any role number* $1 \leq i \leq k$, *for any session number* $s_i \in \mathbb{N}$ *that either corresponds to role i, that is* $(i, s_i) \in \mathsf{sc}$ *or does not appear in the scenario, that is* $\forall j, (j, s_i) \notin \mathsf{sc}$, *the following constraint system is not satisfiable*

$$T_0' \Vdash u_1 \wedge \bigwedge_{1 \leq i < n} (T_0' \cup \{v_1, \ldots, v_i\} \Vdash u_{i+1}) \wedge (T_0' \cup \{v_1, \ldots, v_n\} \Vdash m\sigma_{1,s_1} \cdots \sigma_{k,s_k})$$

where $T_0' = T_0 \cup \{\mathsf{init}\}$ *and* $(u_1, v_1) \cdots (u_n, v_n)$ *is the sequence of rules associated to* sc *and* $\sigma_{r,s}$ *is the substitution defined in Definition 4.*

The initial knowledge typically contains the names and the public keys of all agents and the private keys of all dishonest agents.

Example 2. Consider again the Needham-Schroeder protocol. Let $\Pi(1)$ and $\Pi(2)$ the two roles introduced in Example 1. This protocol is well-known to be insecure w.r.t. $m = Y$ and $T_0 = \{\mathrm{priv}(c), \mathrm{pub}(c), a, b, \mathrm{pub}(a), \mathrm{pub}(b)\}$ (see [16] for a complete description of the attack). Let s_1 and s_2 be two session numbers ($s_1 \neq s_2$) and consider $\mathsf{sc} = (1, s_1)\,(2, s_2)\,(1, s_1)\,(2, s_2)$. The system $\mathcal{C}$ associated to T_0, sc and $m\sigma_{1,s_1}\sigma_{2,s_2} = n_{Y,s_2}$ (according to Definition 5) is given below.

$$\mathcal{C} := \begin{cases} T_0, \mathsf{init} \Vdash \mathsf{init} \\ T_1 \overset{\mathsf{def}}{=} T_0,\ \mathsf{init},\ \mathrm{enca}(\langle n_{X,s_1}, a\rangle, \mathrm{pub}(c)) \Vdash \mathrm{enca}(\langle y_{s_2}, a\rangle, \mathrm{pub}(b)) \\ T_2 \overset{\mathsf{def}}{=} T_1,\ \mathrm{enca}(\langle y_{s_2}, n_{Y,s_2}\rangle, \mathrm{pub}(a)) \Vdash \mathrm{enca}(\langle n_{X,s_1}, x_{s_1}\rangle, \mathrm{pub}(a)) \\ T_2,\ \mathrm{enca}(x_{s_1}, \mathrm{pub}(c)) \Vdash \mathrm{enca}(n_{Y,s_2}, \mathrm{pub}(b)) \\ T_2,\ \mathrm{enca}(x_{s_1}, \mathrm{pub}(c)) \Vdash n_{Y,s_2} \end{cases}$$

The substitution $\sigma = \{y_{s_2} \mapsto n_{X,s_1}, x_{s_1} \mapsto n_{Y,s_2}\}$ is a solution of $\mathcal{C}$.

3 Composition Result

3.1 Hypothesis

Even if a protocol is secure for an unbounded number of sessions, its security may collapse if the protocol is executed in an environment where other protocols sharing some common keys are executed. We have seen a first example in the introduction. To avoid a cyphertext from a protocol Π_1 to be decrypted in an another protocol Π_2, we introduce the notion of *well-tagged* protocol.

Definition 6 (well-tag, α-tag). *Let α be a term. We say that a term t is α-tagged if for every $t' \in St(t)$ of the form $t' = \mathrm{enc}(t_1, t_2)$, $t' = \mathrm{enca}(t_1, t_2)$, or $t' = \mathrm{sign}(t_1, t_2)$, we have $t_1 = \langle \alpha, t'_1 \rangle$ for some term t'_1. A term is said* well-tagged *if it is α-tagged for some term α.*

A protocol Π is α-tagged is any term occurring in the role of the protocol is α-tagged. A protocol is said well-tagged *if it is α-tagged for some term α.*

Requiring that a protocol is well-tagged can be very easily achieved in practice: it is sufficient for example to add the name of the protocol in each encrypted term. Moreover, note that (as opposite to [14]) this does not require that the agents check that nested encrypted terms are correctly tagged. For example, let Π be a protocol with one role as follows:

$$\Pi(1) = (\mathrm{enca}(\langle \alpha, x \rangle, \mathrm{pub}(a)) \to \mathrm{enca}(\langle \alpha, x \rangle, \mathrm{pub}(b))).$$

The protocol Π is α-tagged and still the message $\mathrm{enca}(\langle \alpha, \mathrm{enc}(a, k) \rangle, \mathrm{pub}(a))$ (which is not α-tagged) would be accepted by the agent playing the role.

Tagging protocols is not sufficient, indeed critical long-term keys should not be revealed in clear. Consider for example the following two well-tagged protocols

$$P_3: \quad A \to B : \{\alpha, s\}_{k_{ab}} \qquad\qquad P_4: \quad A \to B : k_{ab}$$

The security of protocol P_3 is again compromised by the execution of P_4. Thus we will require that long-term keys (except possibly the public ones) do not occur in plaintext in the protocol.

Definition 7 (plaintext). *The set $plaintext(t)$ of* plaintext *of a term t is the set of extended names and variables, that is recursively defined as follows.*

$$\begin{array}{ll}
plaintext(u) = \{u\} & \textit{if } u \textit{ is a variable or a name} \\
plaintext(f(u)) = \{f(u)\} & \textit{for } f \in \{\mathrm{pub}, \mathrm{priv}\} \\
plaintext(\langle u_1, u_2 \rangle) = plaintext(u_1) \cup plaintext(u_2) & \\
plaintext(f(u_1, u_2)) = plaintext(u_1) & \textit{for } f \in \{\mathrm{enc}, \mathrm{enca}, \mathrm{sign}\}
\end{array}$$

This notation is extended to set of terms and protocols as expected .

Some weird protocols may still reveal critical keys in a hidden way. Consider for example the following one role (α-tagged) protocol.

$$\Pi(1) = (\mathsf{init} \to \mathrm{enc}(\langle \alpha, a \rangle, k_{ab})),\ (\mathrm{enc}(\langle \alpha, a \rangle, x) \to x)$$

While the long-term key k_{ab} does not appear in plaintext, the key k_{ab} is revealed after simply one normal execution of the role. This protocol is however not realistic since an unknown value cannot be learned (and sent) if it does not appear previously in plaintext. Thus we will further require (Condition 2 of Theorem 1) that a variable occurring in plaintext in a sent message, has to previously occur in plaintext in a received message.

3.2 Composition Theorem

We show that two well-tagged protocols can be safely composed as soon as they use different tags and that critical long-term keys do not appear in plaintext.

Theorem 1. *Let Π_1 and Π_2 be two well-tagged protocols such that Π_1 is α-tagged and Π_2 is β-tagged with $\alpha \neq \beta$. Let T_0 (intuitively the initial knowledge of the intruder) be a set of extended names. Let $\mathsf{KC} = (\mathrm{n}(\Pi_1) \cup \mathrm{n}(\Pi_2)) \smallsetminus T_0$ be the set of* critical extended names. *Let m be a term constructed from Π_1 such that m is α-tagged and $vars(m) \subseteq vars(\Pi_1)$. Moreover, we assume that*

1. *critical extended names do not appear in plaintext, that is*
$$\mathsf{KC} \cap (plaintext(\Pi_1) \cup plaintext(\Pi_2)) = \emptyset.$$
2. *for any role $(\mathsf{rcv}_1 \xrightarrow{N_1} \mathsf{snd}_1) \cdots (\mathsf{rcv}_k \xrightarrow{N_k} \mathsf{snd}_k)$ of Π_1 or Π_2, for any variable $x \in plaintext(\mathsf{snd}_i)$, we have $x \in \bigcup_{1 \leq j \leq i} N_j \cup \{plaintext(\mathsf{rcv}_j)\}$.*

Then Π_1 preserves the secrecy of m for the initial knowledge T_0 if and only if $\Pi_1 \mid \Pi_2$ preserves the secrecy of m for T_0.

We have seen in Section 3.1 that conditions 1 and 2 are necessary conditions. Moreover, condition 2 will be satisfied by any realistic (executable) protocol. We require that terms from Π_1 and Π_2 are tagged with distinct tags for simplicity. The key condition is actually that for any encrypted (or signed) subterm t_1 of Π_1 and for any encrypted (or signed) subterm t_2 of Π_2, the terms t_1 and t_2 cannot be unified.

Theorem 1 is proved by contradiction. Assume that $\Pi_1 \mid \Pi_2$ does not preserve the secrecy of m for T_0. It means that there exists a scenario sc for $\Pi_1 \mid \Pi_2$ such that the constraint system associated to sc, T_0 and m is satisfiable. Proposition 1 ensures that in this case, there exists a scenario sc' for Π_1 such that the constraint system associated to sc', T_0 and m is satisfiable, which means that Π_1 does not preserve the secrecy of m for some initial knowledge T_0, contradiction.

Proposition 1. *Let $\Pi_1 = [k_1] \to \mathsf{Roles}$, $\Pi_2 = [k_2] \to \mathsf{Roles}$, T_0 and m defined as in Theorem 1 and satisfying the conditions 1 and 2. Let $k = k_1 + k_2$ and sc be a scenario for $\Pi_1 \mid \Pi_2$. For any role number $1 \leq i \leq k$, let $s_i \in \mathbb{N}$ such that $(i, s_i) \in \mathsf{sc}$ or $\forall j, (j, s_i) \notin \mathsf{sc}$. Let $\mathcal{C}$ be the constraint system associated to sc, T_0 and $m\sigma_{1,s_1} \cdots \sigma_{k,s_k}$. Let $\mathsf{sc}' = \mathsf{sc}|_{\Pi_1}$ and $\mathcal{C}'$ be the constraint system associated to sc', T_0 and $m\sigma_{1,s_1} \cdots \sigma_{k_1,s_{k_1}}$. If $\mathcal{C}$ is satisfiable, then $\mathcal{C}'$ is also satisfiable.*

4 Proof of our Combination Result

To prove our result, we first refine an existing decision procedure for solving constraint systems. Several decision procedures already exist [17, 9, 11, 19] for solving constraint systems. Some of them [17, 9, 11] are based on a set of simplification rules allowing a general constraint system to be reduced to some simpler one, called *solved*, on which satisfiability can be easily decided. A constraint system is said *solved* [11] if it is different from $\perp$ and if each of its constraints is of the form $T \Vdash x$, where x is a variable. Note that the empty constraint system is solved. Solved constraint systems are particularly simple since they always have a solution. Indeed, let T_1 be the smallest (w.r.t. inclusion) left hand side of a constraint. From the definition of a constraint system we have that T_1 is non empty and has no variable. Let $t \in T_1$. Then the substitution τ defined by $x\tau = t$ for every variable x is a solution since $T \vdash x\theta$ for any constraint $T \Vdash x$ of the solved constraint system.

The *simplification rules* we consider are the following ones:

R_1 :	$\mathcal{C} \wedge T \Vdash u \rightsquigarrow \mathcal{C}$	if $T \cup \{x \mid T' \Vdash x \in \mathcal{C}, T' \subsetneq T\} \vdash u$
R_2 :	$\mathcal{C} \wedge T \Vdash u \rightsquigarrow_\sigma \mathcal{C}\sigma \wedge T\sigma \Vdash u\sigma$	if $\sigma = \mathrm{mgu}(t, u)$ where $t \in \mathit{St}(T)$, $t \neq u$, and t, u are neither variables nor pairs
R_3 :	$\mathcal{C} \wedge T \Vdash u \rightsquigarrow_\sigma \mathcal{C}\sigma \wedge T\sigma \Vdash u\sigma$	if $\sigma = \mathrm{mgu}(t_1, t_2)$, $t_1, t_2 \in \mathit{St}(T)$, $t_1 \neq t_2$, and t_1, t_2 are neither variables nor pairs
R_4 :	$\mathcal{C} \wedge T \Vdash u \rightsquigarrow \perp$	if $\mathit{vars}(T, u) = \emptyset$ and $T \nvdash u$
R_5 :	$\mathcal{C} \wedge T \Vdash f(u, v) \rightsquigarrow \mathcal{C} \wedge T \Vdash u \wedge T \Vdash v$	for $f \in \{\langle\rangle, \mathrm{enc}, \mathrm{enca}, \mathrm{sign}\}$

All the rules are indexed by a substitution (when there is no index then the identity substitution is implicitly considered). We write $\mathcal{C} \rightsquigarrow_\sigma^* \mathcal{C}'$ if there are $\mathcal{C}_1, \ldots, \mathcal{C}_n$ such that $\mathcal{C} \rightsquigarrow_{\sigma_0} \mathcal{C}_1 \rightsquigarrow_{\sigma_1} \ldots \rightsquigarrow_{\sigma_n} \mathcal{C}'$ and $\sigma = \sigma_0\sigma_1 \ldots \sigma_n$. Our rules are the same than in [11] except that we forbid unification of terms headed by $\langle\ \rangle$. Correction and termination are still ensured by [11] and we show that they still form a complete decision procedure. Intuitively, unification between pairs is useless since pairs can be decomposed in order to perform unification on its components. Then, it is possible to build again the pair if necessary. Note that this is not always possible for encryption since the key used to decrypt or encrypt may be unknown by the attacker. Proving that forbidding unification between pairs still leads to a complete decision procedure required in particular to introduce a new notion of minimality for tree proofs for deduction. Note that this result is of independent interest. Indeed, we provide a more efficient decision procedure for solving constraint systems, thus for deciding secrecy for a bounded number of sessions. Of course, the theoretical complexity remains the same (NP).

Theorem 2. *Let $\mathcal{C}$ be an unsolved constraint system.*

1. *(Correctness) If $\mathcal{C} \rightsquigarrow_\sigma^* \mathcal{C}'$ for some constraint system $\mathcal{C}'$ and some substitution σ and if θ is a solution of $\mathcal{C}'$ then $\sigma\theta$ is a solution of $\mathcal{C}$.*
2. *(Completeness) If θ is a solution of $\mathcal{C}$, then there exist a solved constraint system $\mathcal{C}'$ and substitutions σ, θ' such that $\theta = \sigma\theta'$, $C \rightsquigarrow_\sigma^* \mathcal{C}'$ and θ' is a solution of $\mathcal{C}'$.*

3. *(Termination) There is no infinite chain* $\mathcal{C} \rightsquigarrow_{\sigma_1} \mathcal{C}_1 \ldots \rightsquigarrow_{\sigma_n} \mathcal{C}_n$.

Proposition 1 is then proved in three main steps. First, Theorem 2 serves as a key result for proving that if $\mathcal{C}$ is satisfiable, then there exists a solution θ where messages from Π_1 and Π_2 are not mixed up. This is obtained by observing that the simplification rules enable us to build θ step by step through unification of subterms of Π_1 and Π_2. Now, since unification between pairs is forbidden, the rules R_2 and R_3 only involve subterms that convey the same tag, i.e subterms issued from the same protocol. Second, conditions 1 and 2 ensure that for any solution θ of $\mathcal{C}$, the critical extended names of KC do not appear in plaintext in $\mathcal{C}\theta$. Third, thanks to the two previous steps, we prove that β-tagged terms (intuitively messages from Π_2) are not useful for deducing α-tagged terms. For this, we establish that $T \vdash u$ implies $\overline{T} \vdash \overline{u}$ where $\overline{\cdot}$ is a function that keep the terms issued from Π_1 unchanged and projects the terms issued from Π_2 on the special name init. The proof is done by induction on the proof tree witnessing $T \vdash u$. It requires in particular the introduction of a new locality lemma for deduction of ground terms. Then, we deduce that, removing from $\mathcal{C}$ all constraints inherited from Π_2 and all β-tagged terms, we obtain a satisfiable constraint $\mathcal{C}'$ that is associated to a scenario of Π_1.

5 Conclusion

In this paper, we have shown how to safely compose secure protocols by tagging encryption, focusing on secrecy properties. Whenever a protocol preserves the secrecy of some data s, it still preserves s secrecy when other tagged protocols are executed in the same environment. We plan to consider the protocol composition problem for larger classes of security properties. In particular, we believe that our result can be extended to authentication-like properties.

More broadly, we foresee composition results in a more general way. In this paper, protocols are composed in the sense that they can be executed in the same environment. We plan to develop composition results where protocols can use other protocols as sub-programs. For example, a protocol could use a secure channel, letting the implementation of the secure channel underspecified. This secure channel could be then possibly implemented by any protocol establishing session keys.

Acknowledgment. We wish to thank J. Guttman for his helpful comments on a preliminary version of this work.

References

1. M. Abadi and R. M. Needham. Prudent engineering practice for cryptographic protocols. *IEEE Trans. Software Eng.*, 22(1):6–15, 1996.
2. R. Amadio and W. Charatonik. On name generation and set-based analysis in the Dolev-Yao model. In *Proc. Inter. Conference on Concurrency Theory (CONCUR'02)*, volume 2421 of *LNCS*, pages 499–514. Springer-Verlag, 2002.

3. S. Andova, C. Cremers, K. G. Steen, S. Mauw, S. M. lsnes, and S. Radomirović. Sufficient conditions for composing security protocols. *Information and Computation*, 2007. To appear.
4. B. Blanchet. An efficient cryptographic protocol verifier based on Prolog rules. In *Proc. 14th Computer Security Foundations Workshop (CSFW'01)*, pages 82–96. IEEE Comp. Soc. Press, 2001.
5. B. Blanchet and A. Podelski. Verification of cryptographic protocols: Tagging enforces termination. In *Foundations of Software Science and Computation Structures (FoSSaCS'03)*, volume 2620 of *LNCS*. Springer, 2003.
6. R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Proc. 42nd Annual Symposium on Foundations of Computer Science (FOCS'01)*, pages 136–145, Las Vegas (Nevada, USA), 2001. IEEE Comp. Soc.
7. R. Canetti, C. Meadows, and P. F. Syverson. Environmental requirements for authentication protocols. In *Proc. Symposium on Software Security – Theories and Systems*, volume 2609 of *LNCS*, pages 339–355. Springer, 2002.
8. H. Comon-Lundh and V. Cortier. Security properties: two agents are sufficient. *Science of Computer Programming*, 50(1-3):51–71, 2004.
9. H. Comon-Lundh and V. Shmatikov. Intruder deductions, constraint solving and insecurity decision in presence of exclusive or. In *Proc. 18th Annual Symposium on Logic in Comp. Sc. (LICS'03)*, pages 271–280. IEEE Comp. Soc. Press, 2003.
10. V. Cortier, J. Delaitre, and S. Delaune. Safely composing security protocols. Research Report 6234, INRIA, 2007. 26 pages.
11. V. Cortier and E. Zalinescu. Deciding key cycles for security protocols. In *Proc. 13th Inter. Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR'06)*, volume 4246 of *LNCS*, pages 317–331. Springer, 2006.
12. A. Datta, A. Derek, J. C. Mitchell, and A. Roy. Protocol composition logic (PCL). *Electr. Notes Theor. Comput. Sci.*, 172:311–358, 2007.
13. L. Gong and P. Syverson. Fail-stop protocols: An approach to designing secure protocols. In *Proc. 5th Inter. Working Conference on Dependable Computing for Critical Applications*, pages 44–55, 1995.
14. J. D. Guttman and F. J. Thayer. Protocol independence through disjoint encryption. In *Proc. 13th Computer Security Foundations Workshop (CSFW'00)*, pages 24–34. IEEE Comp. Soc. Press, 2000.
15. J. Kelsey, B. Schneier, and D. Wagner. Protocol interactions and the chosen protocol attack. In *Proc. 5th Inter. Workshop on Security Protocols*, volume 1361 of *LNCS*, pages 91–104. Springer, 1997.
16. G. Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In *Proc. 2nd International Workshop on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'96)*, volume 1055 of *LNCS*, pages 147–166, Berlin (Germany), 1996. Springer-Verlag.
17. J. K. Millen and V. Shmatikov. Constraint solving for bounded-process cryptographic protocol analysis. In *Proc. 8th ACM Conference on Computer and Communications Security (CCS'01)*, pages 166–175, 2001.
18. R. Needham and M. Schroeder. Using encryption for authentication in large networks of computers. *Communication of the ACM*, 21(12):993–999, 1978.
19. M. Rusinowitch and M. Turuani. Protocol insecurity with finite number of sessions and composed keys is NP-complete. *Theoretical Comp. Sc.*, 299:451–475, 2003.
20. H. Seidl and K. N. Verma. Flat and one-variable clauses: Complexity of verifying cryptographic protocols with single blind copying. In *Proc. 11th Inter. Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR'04)*, volume 3452 of *LNCS*. Springer, 2005.