From one Session to many: Dynamic Tags for Security Protocols[*]

Myrto Arapinis, Stéphanie Delaune, and Steve Kremer

LSV, ENS Cachan & CNRS & INRIA, France

Abstract. The design and verification of cryptographic protocols is a notoriously difficult task, even in abstract Dolev-Yao models. This is mainly due to several sources of unboundedness (size of messages, number of sessions, ...). In this paper, we characterize a class of protocols for which secrecy for an unbounded number of sessions is decidable. More precisely, we present a simple transformation which maps a protocol that is secure for a single protocol session (a decidable problem) to a protocol that is secure for an unbounded number of sessions.

Our result provides an effective strategy to design secure protocols: (i) design a protocol intended to be secure for one protocol session (this can be verified with existing automated tools); (ii) apply our transformation and obtain a protocol which is secure for an unbounded number of sessions. The proof of our result is closely tied to a particular constraint solving procedure by Comon-Lundh *et al.*

1 Introduction

Security protocols are small distributed programs which aim at guaranteeing properties such as confidentiality of data, authentication of participants, etc. The security of these protocols relies on the one hand on the security of cryptographic primitives, e.g. encryption and digital signatures, and on the other hand the concurrency-related aspects of the protocols themselves. History has shown that even if cryptography is supposed to be perfect, such as in the classical Dolev-Yao model [16], the correct design of security protocols is notoriously error-prone. See for instance [7] for an early survey on attacks. These difficulties come mainly from two sources of unboundedness: a protocol may be executed several times (we get several protocol *sessions*) and the attacker is allowed to build messages of unbounded size. Indeed, secrecy is known to be undecidable when an unbounded number of sessions is allowed, even if the message size is bounded [17]. However, when the number of sessions is bounded, and even without assuming a bounded message size, the problem becomes co-NP-complete [25]. Moreover, special purpose verification tools (e.g. [3]) exist which are highly efficient when the number of sessions is small.

In this paper we consider the secrecy property and we propose a protocol transformation which maps a protocol that is secure for a single session to a protocol that is secure for an unbounded number of sessions. This provides an effective strategy to design secure protocols: (i) design a protocol intended to be secure for one protocol

---

session (this can be efficiently verified with existing automated tools); (ii) apply our transformation and obtain a protocol which is secure for an unbounded number of sessions.

Our transformation. Suppose that Π is a protocol between k participants $A_1, \ldots, A_k$. Our transformation adds to Π a preamble in which each participant sends a freshly generated nonce N_i together with his identity to all other participants. This allows each participant to compute a dynamic, session dependent *tag* $\langle A_1, N_1 \rangle, \ldots, \langle A_k, N_k \rangle$ that will be used to tag each encryption and signature in Π. Our transformation is surprisingly simple and does not require any cryptographic protection of the preamble. Intuitively, the security relies on the fact that the participant A_i decides on a given tag for a given session which is ensured to be fresh as it contains his own freshly generated nonce N_i. The transformation is computationally light as it does not add any cryptographic application; it may merely increase the size of messages to be encrypted or signed. The transformation applies to a large class of protocols, which may use symmetric and asymmetric encryption, digital signature and hash function.

We may note that, *en passant*, we identify a class of tagged protocols for which security is decidable for an unbounded number of sessions. This directly follows from our main result as it stipulates that verifying security for a single protocol session is sufficient to conclude security for an unbounded number of sessions.

Related Work. The kind of compiler we propose here has also been investigated in the area of cryptographic design in computational models, especially for the design of group key exchange protocols. For example, Katz and Yung [19] proposed a compiler which transforms a key exchange protocol secure against a passive eavesdropper into an authenticated protocol which is secure against an active attacker. Earlier work includes compilers for 2-party protocols (e.g. [5]). In the symbolic model, recent works [12, 4] allow one to transform a protocol which is secure in a weak sense (roughly no attacker [12] or just a passive one [4] and a single session) into a protocol secure in the presence of an active attacker and for an unbounded number of sessions. All of these works share however a common drawback: the proposed transformations make heavy use of cryptography. This is mainly due to the fact that the security assumptions made on the input protocol are rather weak. As already mentioned in [12], it is important, from an efficiency perspective to lighten the use of cryptographic primitives. In this work, we succeed in doing so at the price of requiring stronger security guarantees on the input protocol. However, we argue that this is acceptable since efficient automatic tools exist to decide this security criterion on the input protocols.

We can also compare our work with existing decidable protocol classes for an unbounded number of sessions. An early result is the PTIME complexity result by Dolev *et al.* [15] for a restricted class, called *ping-pong* protocols. Other classes have been proposed by Ramanujam and Suresh [23, 24], and Lowe [21]. However, in both cases, temporary secrets, composed keys and ciphertext forwarding are not allowed which discards protocols, such as the Yahalom protocol [7] (see also Section 4.3).

Outline of the paper. In Section 2 we describe the term algebra which is used to model protocol messages as well as the intruder capabilities to manipulate such terms. Then,

2

in Section 3, we define the protocol language we use to model protocols. In Section 4 we formally describe our transformation and state our main transference result. Finally, in Section 5, we prove our main result. Due to a lack of space the proofs are given in [1].

2 Messages and intruder capabilities

2.1 Messages

We use an abstract term algebra to model the messages of a protocol. For this we fix several disjoint sets. We consider an infinite set of *agents* $\mathcal{A} = \{\epsilon, a, b \ldots\}$ with the special agent ϵ standing for the attacker and an infinite set of *agent variables* $\mathcal{X} = \{x_A, x_B, \ldots\}$. We need also to consider an infinite set of *names* $\mathcal{N} = \{n, m \ldots\}$ and an infinite set of *variables* $\mathcal{Y} = \{y, z, \ldots\}$. We consider the following *signature* $\mathcal{F} = \{\text{enc}/2, \text{enca}/2, \text{sign}/2, \langle \rangle/2, \text{h}/1, \text{priv}/1, \text{shk}/2\}$. These function symbols model cryptographic primitives. The symbol $\langle \rangle$ represents pairing. The term $\text{enc}(m, k)$ (resp. $\text{enca}(m, k)$) represents the message m encrypted with the symmetric (resp. asymmetric) key k whereas the term $\text{sign}(m, k)$ represents the message m signed by the key k. The function h models a hash function whereas $\text{priv}(a)$ is used to model the private key of an agent a, and $\text{shk}(a, b)$ $(= \text{shk}(b, a))$ is used to model the long-term symmetric key shared by agents a and b. For simplicity, we confuse the agent names with their public key. Names are used to model atomic data such as nonces. The set of *terms* is defined inductively by the following grammar:

$$t, t_1, t_2 \ldots ::= y \mid n \mid x \mid a \mid \text{priv}(u_1) \mid \text{shk}(u_1, u_2) \mid f(t_1, t_2) \mid \text{h}(t)$$

where $u_1, u_2 \in \mathcal{A} \cup \mathcal{X}$, and $f \in \{\langle \rangle, \text{enc}, \text{enca}, \text{sign}\}$. We sometimes write $\langle t_1, \ldots, t_n \rangle$ instead of writing $\langle t_1, \langle, \ldots, \langle t_{n-1}, t_n \rangle \ldots \rangle \rangle$. We say that a term is *ground* if it has no variable. We consider the usual notations for manipulating terms. We write $vars(t)$ (resp. $fresh(t)$, $agent(t)$) for the set of variables (resp. names, agents) occurring in t. We write $St(t)$ for the set of *subterms* of a term t and define the set of *cryptographic subterms* of a term t as $CryptSt(t) = \{f(t_1, \ldots, t_n) \in St(t) \mid f \in \{\text{enc}, \text{enca}, \text{sign}, \text{h}\}\}$. Moreover we define the set of *long-term keys* of a term t as

$$lgKeys(t) = \{\text{priv}(u) \mid u \in \mathcal{A} \cup \mathcal{X} \text{ and } u \in St(t)\} \cup \{\text{shk}(u_1, u_2) \in St(t)\}.$$

and we define $\mathcal{K}_\epsilon = \{\text{priv}(\epsilon)\} \cup \{\text{shk}(a, \epsilon) \mid a \in \mathcal{A}\}$. Intuitively $\mathcal{K}_\epsilon$ represents the set of long-term keys of the attacker. An *atom* is a long-term key, a name or a variable. We define the set of *plaintexts* of a term t as the set of atoms that occur in plaintext, i.e

- $plaintext(\text{h}(u)) = plaintext(f(u, v)) = plaintext(u)$ for $f \in \{\text{enc}, \text{enca}, \text{sign}\}$,
- $plaintext(\langle u, v \rangle) = plaintext(u) \cup plaintext(v)$, and
- $plaintext(u) = \{u\}$ otherwise.

All these notions are extended to sets of terms and to other kinds of term containers as expected. We denote by $\#S$ the cardinality of a set S. Substitutions are written $\sigma = \{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$ where its *domain* is $\text{dom}(\sigma) = \{x_1, \ldots, x_n\}$. The substitution σ is *ground* if all the t_i are ground. The application of a substitution σ to a term t is written $\sigma(t)$ or $t\sigma$. Two terms t_1 and t_2 are *unifiable* if $t_1\sigma = t_2\sigma$ for some substitution σ, that is called a *unifier*. We denote by $\text{mgu}(t_1, t_2)$ the *most general unifier* of t_1 and t_2.

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

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

$$\frac{T \vdash \mathsf{enca}(u, v) \quad T \vdash \mathsf{priv}(v)}{T \vdash u} \qquad \frac{T \vdash \mathsf{sign}(u, v)}{T \vdash u} \text{(optional)} \qquad \frac{}{T \vdash u} u \in T \cup \mathcal{A} \cup \mathcal{K}_\epsilon$$

Fig. 1. Intruder deduction system.

Example 1. Let $t = \mathsf{enc}(\langle n, a \rangle, \mathsf{shk}(a, b))$. We have that $vars(t) = \emptyset$, i.e. t is ground, $fresh(t) = \{n\}$, $agent(t) = \{a, b\}$, $lgKeys(t) = \{\mathsf{priv}(a), \mathsf{priv}(b), \mathsf{shk}(a, b)\}$ and $plaintext(t) = \{n, a\}$. The terms $\mathsf{shk}(a, b)$, a, n and $\mathsf{priv}(a)$ are atoms.

2.2 Intruder capabilities

We model the intruder's abilities to construct new messages by the deduction system given in Figure 1. The intuitive meaning of these rules is that an intruder can compose new messages by pairing, encrypting, signing and hashing previously known messages provided he has the corresponding keys. Conversely, he can decompose messages by projecting or decrypting provided he has the decryption keys. Our optional rule expresses that an intruder can retrieve the whole message from its signature. Whether this property holds depends on the actual signature scheme. Therefore we consider this rule to be optional. Our results work in both cases.

Definition 1 (Deducible). *We say that a term u is deducible from a set of terms T, denoted $T \vdash u$, if there exists a tree such that its root is labeled by $T \vdash u$ and for every node labeled by $T \vdash v$ having n sons labeled by $T \vdash v_1, \ldots, T \vdash v_n$ we have that $\frac{T \vdash v_1, \ldots, T \vdash v_n}{T \vdash v}$ is an instance of one of the inference rules given in Figure 1.*

Example 2. The term $\langle n, \mathsf{shk}(a, b) \rangle$ is deducible from $\{\mathsf{enc}(n, \mathsf{shk}(a, b)), \mathsf{shk}(a, b)\}$.

3 Model for security protocols

In this section, we give a language for specifying protocols and define their execution in the presence of an active attacker. Our model is similar to existing ones (see *e.g.* [25, 10]) and mostly inspired by [11] except for the fact that we introduce *parametrized roles* which allows us to instantiate new sessions.

3.1 Syntax

Our protocol model allows parties to exchange messages built from identities and randomly generated nonces using public key and symmetric encryption, digital signature and hashing. The individual behavior of each protocol participant is defined by a *role*. A role describes a sequence of *events*, i.e. a sequence of receiving and sending.

Definition 2 (Event, role and protocol). *An* event e *is either a* receive event, *denoted* rcv(u), *or a* send event, *denoted* snd(u), *where u is a term. A* role *is of the form* $\lambda x_1 \dots \lambda x_k . \nu y_1 \dots \nu y_p .$ seq, *where*

- $X = \{x_1, \dots, x_k\}$ *is a set of agent variables, i.e. the parameters of the role corresponding to the k participants of the protocol,*
- $Y = \{y_1, \dots, y_p\}$ *is a set of variables: the nonces generated by the role,*
- seq $=$ e$_1$; e$_2$; $\dots$; e$_\ell$ *is a sequence of events such that* $(vars(\text{seq}) \smallsetminus \{X\}) \subseteq \mathcal{Y}$, *i.e. all agents variables are parameters.*

Moreover, for all i, $1 \leq i \leq \ell$, we have that e$_i =$ snd(u) *implies*

1. $vars(u) \subseteq X \cup Y \cup \{vars(v) \mid$ e$_j =$ rcv(v) *and* $j < i\}$, *and*
2. $vars(plaintext(u)) \subseteq X \cup Y \cup \{vars(plaintext(v)) \mid$ e$_j =$ rcv(v) *and* $j < i\}$.

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

The two last conditions on variables only discard protocols that are not executable by requiring that each variable which appears in a sent term (at a plaintext position) is either one of the parameters, nonces, or is a variable which has been bound by a previous receive event (at a plaintext position). Condition 1 is rather standard whereas Condition 2 will be useful later on to trace data that occur in plaintext position.

Example 3. We illustrate our protocol syntax on the familiar Needham-Schroeder public-key protocol. In our syntax this protocol is modeled as follows.

$$\Pi(1) = \lambda x_A . \lambda x_B . \nu y. \qquad\qquad \Pi(2) = \lambda x_A . \lambda x_B . \nu y'.$$
$$\text{snd}(\text{enca}(\langle y, x_A \rangle, x_B)); \qquad\qquad \text{rcv}(\text{enca}(\langle z', x_A \rangle, x_B));$$
$$\text{rcv}(\text{enca}(\langle y, z \rangle, x_A)); \qquad\qquad \text{snd}(\text{enca}(\langle z', y' \rangle, x_A));$$
$$\text{snd}(\text{enca}(z, x_B)) \qquad\qquad \text{rcv}(\text{enca}(y', x_B))$$

The initiator, role $\Pi(1)$ played by x_A, sends to the responder, role $\Pi(2)$ played by x_B, his identity together with a freshly genrated nonce y, encrypted with the responder's public key. The reponder replies by copying the initiator's nonce and adds a fresh nonce y', encrypted by the initiator's public key. The initiator acknowledges by forwarding the responder's nonce encrypted by its public key.

3.2 Scenarios and sessions

In our model, a session corresponds to the instantiation of one role. This means in particular that one "normal execution" of a k-party protocol requires k sessions, one per role[1]. We may want to consider several sessions corresponding to different instantiations of a same role. Since the adversary may block, redirect and send new messages, all the sessions might be interleaved in many ways. Such an interleaving is captured by the notion of a *scenario*.

---

[1] In the literature, the word session is often used in an abusive way to represent an execution of the *protocol*, i.e. one session per role, whereas we use it for the execution of a *role*.

Definition 3 (Scenario). *A* scenario *for a protocol* $\Pi : [k] \rightarrow$ Roles *is a sequence* $\mathsf{sc} = (r_1, sid_1) \cdots (r_n, sid_n)$ *where* r_i *is a role and* sid_i *a session identifier such that* $1 \leq r_i \leq k$, $sid_i \in \mathbb{N} \smallsetminus \{0\}$, *the number of identical occurrences of a pair* (r, sid) *is smaller than the length of the role* r, *and* $sid_i = sid_j$ *implies* $r_i = r_j$.

The condition on identical occurrences ensures that a role cannot execute more events than it contains. 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.

Given a scenario and an instantiation for the parameters, we define a *symbolic trace*, that is a sequence of events that corresponds to the interleaving of the scenario, for which the parameters have been instantiated, fresh nonces are generated and variables are renamed to avoid name collisions between different sessions.

Definition 4 (Symbolic trace). *Let* Π *be a* k*-party protocol with*

$$\Pi(j) = \lambda x_1^j \ldots \lambda x_k^j . \nu y_1^j \ldots . \nu y_{p_j}^j . \mathsf{e}_1^j; \ldots; \mathsf{e}_{\ell_j}^j \quad \text{for } 1 \leq j \leq k.$$

Given a scenario $\mathsf{sc} = (r_1, sid_1) \cdots (r_n, sid_n)$ *and a function* $\alpha : \mathbb{N} \rightarrow \mathcal{A}^k$, *the symbolic trace* $\mathsf{tr} = \mathsf{e}_1; \ldots; \mathsf{e}_n$ *associated to* sc *and* α *is defined as follows. Let* $q_i = \#\{(r_j, sid_j) \in \mathsf{sc} \mid j \leq i, sid_j = sid_i\}$, *i.e. the number of previous occurrences in* sc *of the session* sid_i. *We have* $q_i \leq \ell_{r_i}$ *and* $\mathsf{e}_i = (\mathsf{e}_{q_i}^{r_i})\sigma_{r_i, sid_i}$, *where*

- $\mathrm{dom}(\sigma_{r, sid}) = \{x_1^r, \ldots, x_k^r\} \cup \{y_1^r, \ldots, y_{p_r}^r\} \cup vars(\Pi(r))$,
- $\sigma_{r, sid}(y) = n_{y, sid}$ *if* $y \in \{y_1^r, \ldots, y_{p_r}^r\}$, *where* $n_{y, sid}$ *is a fresh name;*
- $\sigma_{r, sid}(x_i^r) = a_i$ *when* $\alpha(sid) = (a_1, \ldots, a_k)$;
- $\sigma_{r, sid}(z) = z_{sid}$ *otherwise, where* z_{sid} *is a fresh variable.*

A session sid *is said to be* honest *w.r.t.* α *when* $\alpha(sid) \in (\mathcal{A} \smallsetminus \{\epsilon\})^k$.

Intuitively, a session sid is honest if all of its participants, from the point of view of the agent playing the session sid, are honest (i.e. $\neq \epsilon$). Since agent variables only occur as parameters in a protocol (see Def. 2), a symbolic trace does not contain agent variables.

We define an operator K which associates to a symbolic trace tr the knowledge gained by the adversary, i.e. the set of (possibly non ground) terms that are sent in this symbolic trace. More precisely, we have that $\mathsf{K}(\mathsf{e}_1; \cdots; \mathsf{e}_\ell) = \bigcup_{1 \leq i \leq \ell} \mathsf{K}(\mathsf{e}_i)$ where $\mathsf{K}(\mathsf{rcv}(u)) = \emptyset$ and $\mathsf{K}(\mathsf{snd}(u)) = \{u\}$. This operator is useful in the following when we associate a constraint system to a symbolic trace.

3.3 Constraint systems

Constraint systems have been successfully used for verifying secrecy properties of finite scenarios (see for instance [25, 22, 13]). We now recall the definition of constraint systems. In the next section we discuss how secrecy for an *unbounded number of sessions* can be specified using (infinite) families of constraint systems.

Definition 5 (Constraint system). *A* constraint system C *is either* $\perp$ *or a finite sequence of expressions* $(T_i \Vdash u_i)_{1 \leq i \leq n}$, *called* constraints. *Each* T_i *is a finite 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. Moreover, we assume that terms in* C *do not contain agent variables and are such that:*

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

A solution *of* C *is a ground substitution* θ *with* $\mathrm{dom}(\theta) = vars(C)$ *such that for every* $(T \Vdash u) \in C$, *we have that* $T\theta \vdash u\theta$. *The empty constraint system is always satisfiable whereas* $\perp$ *denotes an unsatisfiable system. We denote by* $\mathsf{maxlhs}(C)$ *(resp.* $\mathsf{minlhs}(C)$*) the maximal (resp. minimal) left-hand side of* C, *i.e.* T_n *(resp.* T_1*). We denote by* $\mathsf{rhs}(C)$ *the set of its right-hand sides, i.e.* $\{u_1, \ldots, u_n\}$.

In the remainder of the paper we often consider constraint systems as sets rather than sequences of constraints, keeping the ordering induced by set inclusion of the left-hand side of constraints implicit. The left-hand side of a constraint system usually represents the messages sent on the network. Hence, the first condition states that the intruder knowledge is always increasing. The second condition in Definition 5 says that each variable occurs first in some right-hand side.

3.4   Secrecy

We now define the secrecy preservation problem for an unbounded number of sessions. Intuitively, a term m is secret if for all possible instantiations and scenarios, the ground term m' obtained when all parameters and nonces have been instantiated during an *honest* session remains secret. This definition leads us to consider an infinite family of constraint systems.

Definition 6 (Secrecy). *Let* Π *be a* k-*party protocol with*

$$\Pi(j) = \lambda x_1^j \ldots \lambda x_k^j . \nu y_1^j \ldots \nu y_{p_j}^j . \mathsf{seq}^j \quad for \ 1 \leq j \leq k.$$

and let $m \in St(\mathsf{seq}^r)$ *for some role* $1 \leq r \leq k$. *We say that* Π *preserves the secrecy of* m *w.r.t.* T_0 *(a finite set of ground atoms), if for any scenario* sc, *for any function* $\alpha :$ $\mathbb{N} \to \mathcal{A}^k$ *and for any honest session* sid_h *(i.e.* $\alpha(sid_h) \in (\mathcal{A} \setminus \{\epsilon\})^k$*) such that* $(r, sid_h) \in \mathsf{sc}$, *the following constraint system is not satisfiable*

$$\{T_0 \cup \mathsf{K}(\mathsf{tr}_i) \Vdash u \mid \mathsf{tr}_i = \mathsf{tr}_{i-1}; \mathsf{rcv}(u) \ and \ 1 \leq i \leq \ell\} \cup \{T_0 \cup \mathsf{K}(\mathsf{tr}) \Vdash m\sigma_{r,sid_h}\}$$

where tr *is the symbolic trace of length* ℓ *associated to* (sc, α) *and* tr_i *its prefix of length* i. *The substitution* σ_{r,sid_h} *is defined as in Definition 4.*

Example 4. Consider again the Needham-Schroeder protocol. Let $\Pi(1)$ and $\Pi(2)$ be the two roles introduced in Example 3. This protocol is well-known to be insecure w.r.t. $m = y'$ for any T_0. Let sid_1 and sid_2 be two session identifiers such that $sid_1 \neq sid_2$ and consider the scenario $\mathsf{sc} = (1, sid_1) (2, sid_2) (2, sid_2), (1, sid_1) (1, sid_1)$ and the function α such that $\alpha(sid_1) = (a, \epsilon)$ and $\alpha(sid_2) = (a, b)$. The constraint system C associated to T_0, sc, α and $m\sigma_{2,sid_2} = n_{y',sid_2}$ (according to Definition 6) is given below.

$$C := \begin{cases} T_1 \overset{\mathsf{def}}{=} T_0, \ \mathsf{enca}(\langle n_{y,sid_1}, a \rangle, \epsilon) \Vdash \mathsf{enca}(\langle z'_{sid_2}, a \rangle, b) \\ T_2 \overset{\mathsf{def}}{=} T_1, \ \mathsf{enca}(\langle z'_{sid_2}, n_{y',sid_2} \rangle, a) \Vdash \mathsf{enca}(\langle n_{y,sid_1}, z_{sid_1} \rangle, a) \\ \qquad T_2, \ \mathsf{enca}(z_{sid_1}, \epsilon) \Vdash n_{y',sid_2} \end{cases}$$

The substitution $\sigma = \{z'_{sid_2} \mapsto n_{y,sid_1}, z_{sid_1} \mapsto n_{y',sid_2}\}$ is a solution of C. However, this protocol preserves the secrecy of m (w.r.t. $T_0 = \emptyset$ for instance) when considering one honest session for each role. This has been formally verified with the AVISPA tool [3]. Our transference result (described in the next section) will ensure that the protocol $\tilde{\Pi}$ (obtained from Π by applying our transformation) is secure for an unbounded number of sessions.

4 Transformation of protocols

In Section 4.1 we define our transformation before we state our main result in Section 4.2 whose proof is postponed to Section 5. Finally, we discuss the tags which are used in our transformation in Section 4.3.

4.1 Our transformation

Given an input protocol Π, our transformation will compute a new protocol $\tilde{\Pi}$ which consists of two phases. During the first phase, the protocol participants try to agree on some common, dynamically generated, session identifier τ. For this, each participant sends a freshly generated nonce N_i together with his identity A_i to all other participants. (Note that if broadcast is not practical or if not all identities are known to each participant, the message can be sent to some of the participants who forward the message.) At the end of this preamble, each participant computes a session identifier: $\tau = \langle\langle A_1, N_1\rangle, \ldots, \langle A_k, N_k\rangle\rangle$. Note that an active attacker may interfere with this initialization phase and may intercept and replace some of the nonces. Hence, the protocol participants do not necessarily agree on the same session identifier τ after this preamble. In fact, each participant computes his own session identifier, say τ_j. During the second phase, each participant j executes the original protocol in which the dynamically computed identifier is used for tagging each application of a cryptographic primitive. In this phase, when a participant opens an encryption, he will check that the tag is in accordance with the nonces he received during the initialization phase. In particular he can test the presence of his own nonce.

The transformation, using the informal Alice-Bob notation, is described below and relies on the tagging operation that is formally defined in Definition 7.

$$\Pi = \begin{cases} A_{i_1} \to A_{j_1} : m_1 \\ \quad\vdots \\ A_{i_\ell} \to A_{j_\ell} : m_\ell \end{cases} \quad \tilde{\Pi} = \begin{cases} \text{Phase 1} \qquad\qquad \text{Phase 2} \\ A_1 \to All : \langle A_1, N_1\rangle \quad A_{i_1} \to A_{j_1} : [m_1]_\tau \\ \quad\vdots \qquad\qquad\qquad \vdots \\ A_k \to All : \langle A_k, N_k\rangle \quad A_{i_\ell} \to A_{j_\ell} : [m_\ell]_\tau \\ \text{where } \tau = \langle \text{tag}_1, \ldots, \text{tag}_k \rangle \text{ with } \text{tag}_i = \langle A_i, N_i\rangle \end{cases}$$

Note that, the Alice-Bob notation only represents what happens in a normal execution, i.e. with no intervention of the attacker. Of course, in such a situation, the participants agree on the same session identifier τ used in the second phase.

8

Definition 7 (k-**tag**, k-**tagging**)**.** *A k-tag is a term $\langle\langle a_1, v_1\rangle, \ldots, \langle a_k, v_k\rangle\rangle$ where each $a_i \in \mathcal{A}$ and each v_i is a term. Let u be a term and tag be a k-tag. The k-tagging of u with tag, denoted $[u]_{\mathsf{tag}}$, is inductively defined as follows:*

$$
\begin{aligned}
[\langle u_1, u_2\rangle]_{\mathsf{tag}} &= \langle [u_1]_{\mathsf{tag}}, [u_2]_{\mathsf{tag}}\rangle \\
[f(u_1, u_2)]_{\mathsf{tag}} &= f(\langle \mathsf{tag}, [u_1]_{\mathsf{tag}}\rangle, [u_2]_{\mathsf{tag}}) \qquad \textit{for } f \in \{\mathsf{enc}, \mathsf{enca}, \mathsf{sign}\} \\
[\mathsf{h}(u_1)]_{\mathsf{tag}} &= \mathsf{h}(\langle \mathsf{tag}, [u_1]_{\mathsf{tag}}\rangle) \\
[u]_{\mathsf{tag}} &= u \qquad\qquad\qquad\qquad\qquad\qquad\quad\; \textit{otherwise}
\end{aligned}
$$

This notion is extended to sequences of events as expected. We are now able to formally define our transformation.

Definition 8 (Protocol transformation). *Let Π be a k-party protocol such that*

$$
\Pi(j) = \lambda x_1^j \ldots \lambda x_k^j . \nu y_1^j \ldots \nu y_{p_j}^j . \mathsf{seq}^j \quad \textit{for } 1 \le j \le k.
$$

and the variables z_i^j ($1 \le i, j \le k$) do not appear in Π (which can always be ensured by renaming variables in Π). The transformed protocol $\tilde{\Pi}$ is a k-party protocol defined as follows:

$$
\tilde{\Pi}(j) = \lambda x_1^j \ldots \lambda x_k^j . \nu y_1^j \ldots \nu y_{p_j}^j . \nu z_j^j . \tilde{\Pi}^{\mathsf{init}}(j); [\mathsf{seq}^j]_{\tau_j} \quad \textit{for } 1 \le j \le k
$$

where $\tau_j = \langle u_1^j, \ldots, u_k^j\rangle$ with $u_i^j = \langle x_i^j, z_i^j\rangle$, and

$$
\tilde{\Pi}^{\mathsf{init}}(j) = \mathsf{rcv}(u_1^j); \ldots; \mathsf{rcv}(u_{j-1}^j); \mathsf{snd}(u_j^j); \mathsf{rcv}(u_{j+1}^j); \ldots; \mathsf{rcv}(u_k^j)
$$

In the above definition, the protocol $\tilde{\Pi}^{\mathsf{init}}$ models the initialization phase and the variables z_i^j correspond to the nonces that are exchanged during this phase. In particular for the role j, the variable z_j^j is a freshly generated nonce while the other variables z_i^j, $i \ne j$, are expected to be bound to the other participant's nonces in the receive events. Remember also that the variables x_i^j are the role parameters which correspond to the agents. The tag computed by the jth role in our transformation consists in the concatenation of the $k - 1$ nonces received during the initialization phase together with the fresh nonce generated by the role j itself, i.e. z_j^j. We illustrate this transformation on the Needham-Schroeder protocol introduced in Section 2.

Example 5. Consider the Needham-Schroeder protocol described in Example 3. Applying our transformation we obtain a 2-party protocol $\tilde{\Pi}$. The role $\tilde{\Pi}(2)$ is described below. The role $\tilde{\Pi}(1)$ can be obtained in a similar way.

$$
\begin{aligned}
\tilde{\Pi}(2) = \lambda x_A \lambda x_B . \nu y' . \nu z_B . & \mathsf{rcv}(\langle x_A, z_A\rangle); \mathsf{snd}(\langle x_B, z_B\rangle); \\
& \mathsf{rcv}(\mathsf{enca}(\langle \tau, \langle z', x_A\rangle\rangle, x_B)); \\
& \mathsf{snd}(\mathsf{enca}(\langle \tau, \langle z', y'\rangle\rangle, x_A)); \\
& \mathsf{rcv}(\mathsf{enca}(\langle \tau, y'\rangle, x_B))
\end{aligned}
$$

where $\tau = \langle\langle x_A, z_A\rangle, \langle x_B, z_B\rangle\rangle$. Note that Lowe's famous man-in-the-middle attack [20] described in Example 4 does not exist anymore on $\tilde{\Pi}$.

4.2 Main theorem

We are now able to state our main transference result.

Theorem 1. *Let Π be a k-party protocol, $\tilde{\Pi}$ be its corresponding transformed protocol, and T_0 be a finite set of ground atoms. Let $m \in St(\Pi(j))$ for some $1 \leq j \leq k$ and $\tilde{m}$ be its counterpart in $\tilde{\Pi}(j)$. Let $\mathsf{CK} = lgKeys(\Pi) \smallsetminus (T_0 \cup \mathcal{K}_\epsilon)$ and assume that $\mathsf{CK} \cap plaintext(\Pi) = \emptyset$, i.e. critical keys do not appear in plaintext.*

If Π preserves the secrecy of m w.r.t. T_0 when considering one honest session of each role, then $\tilde{\Pi}$ preserves the secrecy of $\tilde{m}$ w.r.t. T_0.

Our result states that if the compiled protocol admits an attack that may involve several honest and dishonest sessions, then there exists an attack which only requires one honest session of each role (and no dishonest sessions). The situation is however slightly more complicated than it may seem at first sight since there is an infinite number of honest sessions, which one would need to verify separately. Actually we can avoid this combinatorial explosion thanks to the following well-known result [9]: when verifying secrecy properties it is sufficient to consider one single honest agent (which is allowed to "talk to herself"). Hence we can instantiate all the parameters with the same agent $a \in \mathcal{A} \smallsetminus \{\epsilon\}$.

Our dynamic tagging is useful to avoid interaction between different sessions of the same role in a protocol execution and allows us for instance to prevent man-in-the-middle attacks (see Example 5). A more detailed discussion showing that *static tags* are not sufficient follows in Section 4.3. As stated in Theorem 1 we need also to forbid long-term secrets in plaintext position (even under an encryption). Note that this condition is generally satisfied by protocols and considered as a prudent engineering practice.

4.3 Other ways of tagging

We have also considered an alternative, slightly different transformation that does not include the identities in the tag, i.e., the tag is simply the sequence of nonces. In that case we obtain a different result: if a protocol admits an attack then there exists an attack which only requires one (not necessarily honest) session for each role. In this case, we need to additionally check for attacks that involve a session engaged with the attacker. On the example of the Needham-Schroeder protocol the man-in-the-middle attack is not prevented by this weaker tagging scheme. However, the result requires one to also consider one dishonest session for each role, hence including the attack scenario. In both cases, it is important for the tags to be *collaborative*, *i.e.* all participants do contribute by adding a fresh nonce.

Finally, different kinds of tags have also been considered in [2, 6, 23]. However these tags are *static* and have a different aim. While our dynamic tagging scheme avoids confusing messages from different sessions, these static tags avoid confusing different messages inside a same session and do not prevent that a same message is reused in two different sessions. Under some additional assumptions (e.g. no temporary secret, no ciphertext forwarding), several decidability results [24, 21] have been obtained by showing that it is sufficient to consider one session per role. But those results can not

deal with protocols such as the Yahalom protocol or protocols which rely on a temporary secret. In the framework we consider here, the question whether such static tags would be sufficient to obtain decidability is still an open question (see [2]). In a similar way, static tags have also been used by Heather et al. [18] to avoid type confusion attacks.

5 Proof of our main result

The proof of our main result is closely tied to a particular procedure for solving constraint systems (see [8, 13, 10]). We therefore first give a brief description of this procedure before outlining the proof itself.

5.1 Constraint solving procedure

The procedure we consider for constraint solving uses the simplification rules of Figure 2 to transform a given constraint system into another, simpler one. Such a simplification step is denoted $C \rightsquigarrow_\sigma C'$ where σ is a substitution that has been applied to C during this step (when omitted it implicitly refers to the identity function). We also write $C \rightsquigarrow_\sigma^n C'$ for a sequence of n steps where σ is the composition of the substitutions applied at each step.

Our constraint solving procedure is very similar to the ones presented in [8, 13, 10, 14]: soundness, completeness and termination can be proved in a similar way. This means that the procedure always terminates after a finite number of steps resulting either in $\perp$ when no solution exists or in a constraint system in *solved form*. A constraint system is in solved form when the right-hand side of each constraint is a variable. In that case the constraint system can be trivially satisfied. Moreover, we assume that rules R_2, R_3, R_4, R_5, and R_6 are applied in priority, i.e. before any instance of the rule R_1. It is easy to see that the procedure remains complete.

For the purpose of our proof, we decorate each term t by a pair (r, s) which denotes the role number and the session identifier in which t originated. The resulting term $t^{(r,s)}$ is called a labeled term. By convention, terms in T_0 (the initial attacker knowledge) are labeled with $(0, 0)$. These decorations do not influence the procedure but provide additional information that is useful in the proof. We could have added these decorations before, but it would increase notational clutter and harm readability.
We extend all notations defined on terms to labeled terms, by providing a session identifier as an additional argument: e.g. $vars(T, sid) = \bigcup_{t^{(r,sid)} \in T} vars(t)$.

5.2 Proof of Theorem 1

Theorem 1 is proved by contradiction. Assume that $\tilde{\Pi}$ admits an attack. This means that there exists a scenario sc, a function $\alpha : \mathbb{N} \to \mathcal{A}^k$ and an honest session sid_h such that the associated constraint system C (according to Definition 6) is satisfiable. We will prove that the constraint system C' associated to sc′ (the subsequence of sc where we only consider some particular sessions, say S, chosen according to the tag τ_{sid_h}

$R_1 : C \wedge T \Vdash u^{(r,sid)} \rightsquigarrow C$ $\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad$ if $T \cup \{x \mid T' \Vdash x \in C, T' \subsetneqq T\} \vdash u$

$R_2 : C \wedge T \Vdash u^{(r,sid)} \rightsquigarrow_\sigma C\sigma \wedge T\sigma \Vdash u\sigma^{(r,sid)}$
$\quad\quad\quad\quad$ if $\sigma = \mathrm{mgu}(t, u)$ where $t \in St(T), t \neq u, t, u$ are neither variables nor pairs

$R_3 : C \wedge T \Vdash u^{(r,sid)} \rightsquigarrow_\sigma C\sigma \wedge T\sigma \Vdash u\sigma^{(r,sid)}$
$\quad\quad\quad\quad$ if $\sigma = \mathrm{mgu}(t_1, t_2), t_1, t_2 \in St(T), t_1 \neq t_2, t_1, t_2$ are neither variables nor pairs

$R_4 : C \wedge T \Vdash u^{(r,sid)} \rightsquigarrow_\sigma C\sigma \wedge T\sigma \Vdash u\sigma^{(r,sid)}$
if $\sigma = \mathrm{mgu}(t_2, t_3)$, $\mathsf{enca}(t_1, t_2) \in St(T)$, $\mathsf{priv}(t_3) \in (plaintext(T) \cup \{\mathsf{priv}(\epsilon)\})$, $t_2 \neq t_3$

$R_5 : C \wedge T \Vdash u^{(r,sid)} \rightsquigarrow \perp$ $\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad$ if $vars(T \cup \{u\}) = \emptyset$ and $T \nvdash u$

$R_6 : C \wedge T \Vdash f(u_1, \ldots, u_n)^{(r,sid)} \rightsquigarrow C \wedge \{T \Vdash u_i^{(r,sid)} \mid 1 \leq i \leq n\}$
$\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad$ for $f \in \{\langle\rangle, \mathsf{enc}, \mathsf{enca}, \mathsf{sign}, \mathsf{h}\}$

Fig. 2. Simplification rules

involved in the cryptographic subterms of the honest session sid_h) and α, is also satisfiable. Intuitively $sid \in S$ if and only if the nonce generated during the initialization phase of the session sid appears in tag τ_{sid_h} at the expected position, i.e. at the rth position where r is the role associated to the session identifier sid. Initially, we have that $C' = C|_S$ according to the following definition.

Definition 9 (Constraint system $C|_S$). *Let C be a constraint system and S a set of session identifiers, we define the restriction of C to S as follows*

$$C|_S := \{T|_S \Vdash u^{(r,s)} \mid s \in S \text{ and } (T \Vdash u^{(r,s)}) \in C\},$$

where $T|_S = \{v^{(r,s)} \in T \mid s \in S \cup \{0\}\}$.

We want to ensure that the simplification steps are stable by restriction to some well-chosen set S of sessions (see Lemma 2). This property does not hold for general constraint systems but only for *well-formed* constraint systems. This notion relies on some additional definitions.

Definition 10 (k-tagged). *A term t is k-tagged if all its cryptographic subterms are tagged with a k-tag, i.e. $\forall u \in CryptSt(t), \exists \mathsf{tag}, u_1, \ldots, u_n. u = f(\langle \mathsf{tag}, u_1 \rangle, \ldots, u_n)$ where $f \in \{\mathsf{enc}, \mathsf{enca}, \mathsf{sign}, \mathsf{h}\}$.*

We denote by $tags(t)$ the set of k-tags which occur in a tagging position in t. Given a set T of k-tagged labeled terms, $tags(T, sid) = \bigcup_{t^{(i,sid)} \in T} tags(t)$.

Definition 11 (Well-formed). *A constraint system $C = (T_i \Vdash u_i)_{1 \leq i \leq n}$ is well-formed w.r.t. a set T of k-tagged labeled terms if the following hold:*

1. $\mathsf{maxlhs}(C) \subseteq T$ *and* $\mathsf{rhs}(C) \subseteq St(T)$;
2. *the constraint system C satisfies the* plaintext origination property, *i.e.*
   if $x \in vars(plaintext(T_i))$ then $\exists j < i$ such that $x \in vars(plaintext(u_j))$;
3. *for all sid we have that $|tags(T, sid)| \leq 1$;*
4. *for all sid_1, sid_2 such that $tags(T, sid_1) \neq tags(T, sid_2)$, we have that*
   $vars(T, sid_1) \cap vars(T, sid_2) = \emptyset \wedge fresh(T, sid_1) \cap fresh(T\ sid_2) = \emptyset.$

12

Intuitively, Condition 1 states that the terms in C are k-tagged. Condition 2 ensures that any variable appearing as a plaintext has been previously received in a plaintext position (this is ensured thanks to the condition 2 of Definition 2). Condition 3 says that all terms that originated in the same session have the same tag. Finally, Condition 4 ensures that sessions that are currently tagged in different ways in C use different variables and different nonces. Note that terms issued from different sessions are not necessarily tagged differently. First, we show that the simplification rules maintain well-formedness.

Lemma 1. *Let T be a set of k-tagged labeled terms, and C be a constraint system well-formed w.r.t. T. Let D be a constraint system, σ be a substitution and n be an integer such that $C \leadsto_\sigma^n D$. Then, we have that D is well-formed w.r.t. $T\sigma$ and, for any session sid, we have that $tags(T\sigma, sid) = (tags(T, sid))\sigma$.*

Relying on Lemma 1 we show that there exists a derivation from $C|_S$ to a constraint system in solved form, i.e. the existence of an attack involving only sessions in S.

Lemma 2. *Let CK_0 be a set of ground atoms such that $\mathsf{CK}_0 \cap \mathcal{K}_\epsilon = \emptyset$, T be a set of k-tagged labeled terms and C be a constraint system well-formed w.r.t. T and such that*

1. *$(lgKeys(C) \smallsetminus \mathsf{CK}_0) \subseteq \mathsf{minlhs}(C)$ and those terms are labeled with $(0, 0)$, and*
2. *$\mathsf{CK}_0 \cap plaintext(\mathsf{maxlhs}(C)) = \emptyset$.*

Let D be a satisfiable constraint system, σ be a substitution and n be an integer such that $C \leadsto_\sigma^n D$. Let tag be a k-tag, $Sid(\mathsf{tag}) = \{sid \mid tags(T\sigma, sid) = \mathsf{tag}\}$. If $(\mathsf{maxlhs}(C) \Vdash u^{(r,sid)}) \in C$ for some u, r and $sid \in Sid(\mathsf{tag})$ then there exists $m \leq n$ such that $C|_{Sid(\mathsf{tag})} \leadsto_{\sigma|_Y}^m D|_{Sid(\mathsf{tag})}$, where $Y = \bigcup_{sid \in Sid(\mathsf{tag})} vars(T, sid)$.

This lemma is proved by induction. The proof is technical and the details can be found in [1]. We consider the different rules and distinguish several cases depending on whether the terms involved are labeled with a session identifier in S or not. For instance, the rules R_5 (resp. R_6) are mimicked by using the same instance of the same rule when the labeled term $u^{(r,sid)}$ (right-hand side of the constraint) is such that $sid \in S$. Otherwise, we keep the constraint system unchanged. For the rule R_2 (resp. R_3) the key point is that terms which are tagged differently cannot be unified and do not share any variables nor fresh names (this is due to well-formedness). Thus, the unifier σ used in this step involved two terms labeled by sid_1 and sid_2 that are either both in S or both not in S. This is due to the fact that, after application of σ, these two terms will be tagged in the same way and thus by definition of S, have the same status. If both are in S, we can apply the same rule. If none of them is in S, we show that σ has no effect and we keep the constraint system unchanged. The case of the rules R_1 and R_4 can also be proved in a similar way.

In order to pursue the proof of Theorem 1, we apply Lemma 2 on the derivation $C \leadsto_\sigma^n D$ witnessing the existence of an attack on $\tilde{\Pi}$ and we consider $S = \{sid \mid tags(T\sigma, sid) = tags(T\sigma, sid_h)\}$, i.e. the sessions that are tagged in the same way that sid_h. We obtain that $C|_S$ can also reach a constraint system in solved form, namely $D|_S$. Moreover, the satisfiability of $C|_S$ witnesses the fact that there is an attack on $\tilde{\Pi}$ that only involves sessions in S. In order to conclude, it remains to show that:

1. *S does not contain two distinct sessions that execute the same role.* Intuitively, this comes from the fact that sessions in S are tagged in the same way (after application of σ) and this is not possible for two distinct sessions that execute the same role. Indeed, the fresh nonce generated by different sessions of the same role ensures that their tags are distinct.
2. *S only contains honest sessions.* First sid_h is an honest session by definition of the secrecy property. Second, since the names of the agents engaged in a role occur in the tag and sessions in S are tagged in the same way as the session sid_h, we conclude that this property is also true for any $sid \in S$.

Thus, there is an attack on $\tilde{\Pi}$ that involves at most one honest session of each role. To conclude, it is easy to see that this attack can also be mounted on the protocol Π.

6 Future work

Our current result applies to transfer secrecy properties. As future work we foresee to extend the scope of our result to other security properties, e.g. authentication or more challenging equivalence based properties. We also plan to extend the result to other intruder theories. We foresee that such results require new proof methods which are not based on the decision procedure as in this paper, but directly on the semantics. Another challenging topic for future research is to obtain more fine-grained characterizations of decidable classes of protocols for an unbounded number of sessions. The new insights gained by our work seem to be a good starting point to extract the conditions needed to reduce the security for an unbounded number of sessions to a finite number of sessions.

References

1. M. Arapinis, S. Delaune, and S. Kremer. From one session to many: Dynamic tags for security protocols. Research Report LSV-08-20, ENS Cachan, France, June 2008. 30 pages.
2. M. Arapinis and M. Duflot. Bounding messages for free in security protocols. In *Proc. 27th Conference on Foundations of Software Technology and Theoretical Computer Science (FST&TCS'07)*, volume 4855 of *LNCS*, pages 376–387. Springer, 2007.
3. A. Armando et al. The Avispa tool for the automated validation of internet security protocols and applications. In *Proc. 17th International Conference on Computer Aided Verification (CAV'05)*, volume 3576 of *LNCS*, pages 281–285. Springer, 2005.
4. D. Beauquier and F. Gauche. How to guarantee secrecy for cryptographic protocols. *CoRR*, abs/cs/0703140, 2007.
5. M. Bellare, R. Canetti, and H. Krawczyk. A modular approach to the design and analysis of authentication and key exchange protocols (extended abstract). In *Proc. 30th Annual ACM Symposium on the Theory of Computing (STOC'98)*, pages 419–428. ACM Press, 1998.
6. B. Blanchet and A. Podelski. Verification of cryptographic protocols: Tagging enforces termination. In *Proc. Foundations of Software Science and Computation Structures (FoS-SaCS'03)*, volume 2620 of *LNCS*, pages 136–152. Springer, 2003.

7. J. Clark and J. Jacob. A survey of authentication protocol literature. 1997.

8. H. Comon. Résolution de contraintes et recherche d'attaques pour un nombre borné de sessions. http://www.lsv.ens-cachan.fr/ comon/CRYPTO/bounded.ps.

9. H. Comon-Lundh and V. Cortier. Security properties: two agents are sufficient. *Science of Computer Programming*, 50(1-3):51–71, 2004.

10. V. Cortier, J. Delaitre, and S. Delaune. Safely composing security protocols. In *Proc. 27th Conference on Foundations of Software Technology and Theoretical Computer Science (FST&TCS'07)*, volume 4855 of *LNCS*, pages 352–363. Springer, 2007.

11. V. Cortier and S. Delaune. Safely composing security protocols. Research Report LSV-08-06, ENS Cachan, France, Mar. 2008. 39 pages.

12. V. Cortier, B. Warinschi, and E. Zălinescu. Synthesizing secure protocols. In *Proc. 12th European Symposium On Research In Computer Security (ESORICS'07)*, volume 4734 of *LNCS*, pages 406–421. Springer, 2007.

13. V. Cortier and E. Zălinescu. 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.

14. S. Delaune. Note: Constraint solving procedure. http://www.lsv.ens-cachan.fr/Publis/PAPERS/PDF/CDD-fsttcs07-addendum.pdf.

15. D. Dolev, S. Even, and R. M. Karp. On the security of ping-pong protocols. In *Proc. Advances in Cryptology (CRYPTO'82)*, pages 177–186, 1982.

16. D. Dolev and A. C. Yao. On the security of public key protocols. In *Proc. of the 22nd Symposium on Foundations of Computer Science (FOCS'81)*. IEEE Comp. Soc. Press, 1981.

17. N. Durgin, P. Lincoln, J. Mitchell, and A. Scedrov. Undecidability of bounded security protocols. In *Proc. Workshop on Formal Methods and Security Protocols*, 1999.

18. J. Heather, G. Lowe, and S. Schneider. How to prevent type flaw attacks on security protocols. In *Proc. 13th Computer Security Foundations Workshop (CSFW'01)*, pages 255–268. IEEE Comp. Soc. Press, 2000.

19. J. Katz and M. Yung. Scalable protocols for authenticated group key exchange. In *Proc. 23rd Annual International Cryptology Conference (CRYPTO'03)*, volume 2729 of *LNCS*, pages 110–125. Springer, 2003.

20. 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.

21. G. Lowe. Towards a completeness result for model checking of security protocols. *Journal of Computer Security*, 7(1), 1999.

22. J. Millen and V. Shmatikov. Constraint solving for bounded-process cryptographic protocol analysis. In *Proc. 8th ACM Conference on Computer and Communications Security (CCS'01)*. ACM Press, 2001.

23. R. Ramanujam and S. P. Suresh. Tagging makes secrecy decidable for unbounded nonces as well. In *Proc. 23rd Conference on Foundations of Software Technology and Theoretical Computer Science (FST&TCS'03)*, volume 2914 of *LNCS*, pages 363–374. Springer, 2003.

24. R. Ramanujam and S. P. Suresh. Decidability of context-explicit security protocols. *Journal of Computer Security*, 13(1):135–165, 2005.

25. M. Rusinowitch and M. Turuani. Protocol insecurity with finite number of sessions and composed keys is NP-complete. *Theoretical Computer Science*, 299(1-3):451–475, 2003.