CORE Metadata, citation and similar papers at core.ac.uk
Provided by Edinburgh Research Explorer

THE UNIVERSITY of EDINBURGH

Edinburgh Research Explorer

Bounding messages for free in security protocols - extension to various security properties

Citation for published version:
Arapinis, M & Duflot, M 2014, 'Bounding messages for free in security protocols - extension to various security properties' Information and Computation, vol. 239, pp. 182-215. DOI: 10.1016/j.ic.2014.09.003

Digital Object Identifier (DOI):
10.1016/j.ic.2014.09.003

Link:
Link to publication record in Edinburgh Research Explorer

Document Version:
Peer reviewed version

Published In:
Information and Computation

General rights
Copyright for the publications made accessible via the Edinburgh Research Explorer is retained by the author(s) and / or other copyright owners and it is a condition of accessing these publications that users recognise and abide by the legal requirements associated with these rights.

Take down policy
The University of Edinburgh has made every reasonable effort to ensure that Edinburgh Research Explorer content complies with UK legislation. If you believe that the public display of this file breaches copyright please contact openaccess@ed.ac.uk providing details, and we will remove access to the work immediately and investigate your claim.

OPEN ACCESS

Download date: 05. Apr. 2019

Bounding messages for free in security protocols

Myrto Arapinis and Marie Duflot

LACL - University Paris 12, France
myrto.arapinis@wanadoo.fr, duflot@univ-paris12.fr

Abstract. The verification of security protocols has been proven to be undecidable in general. Different approaches use simplifying hypotheses in order to obtain decidability for interesting subclasses. Amongst the most common is type abstraction, *i.e.* considering only well-typed runs, therefore bounding message length. In this paper we show how to get message boundedness "for free" under a reasonable (syntactic) assumption on protocols, which we call well-formedness. This enables us to improve existing decidability results.

1 Introduction

Security protocols are short programs that describe communication between two or more parties in order to achieve security goals such as data confidentiality, identification of a correspondent,... The protocols are executed in a hostile environment, such as the Internet, and aim at preventing a malicious agent from tampering with the messages, for instance, using encryption. However, encrypting messages is not sufficient to ensure security properties. History has shown that these protocols are extremely error-prone and careful, formal verification is needed.
Despite the apparent simplicity of such protocols, their verification is a difficult problem and has been proven undecidable in general [DLMS99,CC01]. Different attempts [DLMS99,Low99,RT01,CC03,RS03a,RS03b,BP05] have successfully exhibited decidable subclasses, and tools for proving security have been designed even though termination is not always guaranteed [Mea96,CJM00,Bla01,CMR01,SBP01]. In the literature, only very few results consider an unbounded number of sessions and even less allow unbounded creation of new nonces (constants generated to ensure freshness of messages). Most papers bound the number of nonces or even the number of sessions in order to obtain decidability (see the recapitulative table in [CDL06]).
In order to obtain decidability, many existing results [Low99,RS03a,DLMS99,CKR$^+$03] bound the message length in adopting a typing abstraction. An assumption according to which one can always tell the type of a given message. While this appears as an unrealistic assumption in the general case, this paper presents a simple way of justifying it. This question has been addressed in [LYH04,HLS03] amongst others, and solved with tagging schemes.
In this paper, we show that when a protocol satisfies a reasonable, syntactic condition of "well-formedness", then the typing abstraction is a correct one. Furthermore, the considered typing system here is much more fine-grained than the one in [LYH04,HLS03], thus refining existing results. Indeed, we prove that a well-formed protocol admits an

attack if and only if it admits a "well-typed" attack with respect to a stronger notion of typing (see section 3.1).

Our notion of well-formed protocols characterizes protocols in which a term cannot be mistaken for another, unless they are of the same type. This notion is often satisfied in protocols found in the literature [CJ97] and, even when the protocol is not well-formed, a light tagging like the one proposed in [BP05] permits to comply with the property and thus use our result. Note that, the syntactic restriction considered here significantly simplifies the ones in [LYH04,HLS03].

Fianlly, to the best of our knowledge, only very few papers [Low99,RS03a,RS03b] give decidability results with an unbounded number of sessions and nonces. In the last part, we show that hypotheses made in [RS03a,Low99] can be slightly modified in order to meet the typing assumption, thus allowing us to refine the result of [RS03b] since the tagging scheme we propose is strictly weaker that the one used in their paper.

2 Modelling security protocols

In this section we define the trace based model used throughout the paper to define and reason about cryptographic protocols.

2.1 The syntax

Terms Data transmitted in the protocol are modelled as terms built by using the following components:

A set $\mathcal{P} = \{P_1, \ldots, P_k\}$ of *principal names* standing for the different participants of the protocol. During one protocol execution, each principal P_i can generate a set $\mathcal{K}_i = \{K_1^i, \ldots, K_{l_i}^i\}$ of short term keys or *session keys*, as well as a set $\mathcal{N}_i = \{N_1^i, \ldots, N_{m_i}^i\}$ of fresh values called *nonces*. The set of session-keys (resp. nonces) generated by all principals is denoted $\mathcal{K}$ (*resp.* $\mathcal{N}$). Finally the participants can use a finite set $\mathcal{C} = \{c_1, \ldots, c_n\}$ of *constants*.

Every principal P_i has its own set $\mathcal{X}_i = \{X_1^i, \ldots, X_{p_i}^i\}$ of *variables*. Variables are used to model the fact that a principal may receive data which it does not know. The set of variables is then $\mathcal{X} = \bigcup_{i=1}^k \mathcal{X}_i$.

The set of terms is defined inductively using the above defined components as follows:

$$\begin{aligned}\mathcal{T} ::= {} & \mathcal{P} \mid \mathtt{pbk}(\mathcal{P}) \mid \mathtt{pvk}(\mathcal{P}) \mid \mathtt{shk}(\mathcal{P},\mathcal{P}) \mid \mathcal{K} \mid \mathcal{N} \mid \mathcal{C} \\ & \mid \mathcal{X} \mid \langle \mathcal{T}, \mathcal{T} \rangle \mid \{\mathcal{T}\}_{\mathcal{T}} \mid \mathtt{sig}_{\mathcal{T}}(\mathcal{T})\end{aligned}$$

where $\mathtt{pbk}(P)$, $\mathtt{pvk}(P)$ and $\mathtt{shk}(P, P')$ are respectively the public key, private key of a principal P and shared key between principals P and P', and $\langle t_1, t_2 \rangle$, $\{t_1\}_{t_2}$, $\mathtt{sig}_{t_1}(t_2)$ represent pairing, encryption and signature.

In what follows, we denote the set of variables of a term t by $\mathcal{V}(t)$, the set of subterms of t by $St(t)$, and the set of encrypted sub-terms of t by $ESt(t) = \{f(t_1, t_2) \in St(t) \mid f \in \{\{\_\}_\_, \mathtt{sig}_\_(\_)\}\}$.

Actions In our formalism, we split the rules commonly used to describe protocols [CJ97] into send and receive actions, in order to capture precisely what one principal sends and what the receiver expects. We thus have a set of actions $\mathcal{D} = \mathcal{S} \cup \mathcal{R}$ where $\mathcal{S} = \{P_i!P_j : t \mid P_i, P_j \in \mathcal{P},\ P_i \neq P_j,\ t \in \mathcal{T}\}$ is the set of *send actions* and $\mathcal{R} = \{P_i?P_j : t \mid P_i, P_j \in \mathcal{P},\ P_i \neq P_j,\ t \in \mathcal{T}\}$ is the set of *receive actions*.
The term of an action is defined by $term(P_i!Pj : t) = term(P_i?Pj : t) = t$, and for every sequence of actions $D = d_1 \dots d_n$, $terms(D) = \bigcup_{i=1}^{n}(term(d_i))$. Similarly the set of variables of D is $\mathcal{V}(D) = \mathcal{V}(terms(D))$, the set of sub-terms of D is $St(D) = \cup_{t \in terms(D)} St(t)$, and the set of encrypted sub-terms of D is $ESt(D) = \cup_{t \in terms(D)} ESt(t)$.

Protocols

Definition 1. *A* protocol $\Pi = s_1 r_1 \dots s_l r_l$ *is a sequence of send-receive actions such that,* $\forall i, 1 \leq i \leq l$

1. $s_i \in \mathcal{S}$ *and* $r_i \in \mathcal{R}$
2. *if* $s_i = P!P' : t$, *then* $r_i = P'?P : t'$
3. *if* $X \in \mathcal{V}(term(s_i))$, *then* $\exists j,\ 1 \leq j < i$ *such that* $X \in \mathcal{V}(term(r_j))$
4. *for every* $1 \leq i \leq l$ *there exists a substitution* $\delta_i \neq \bot$, *with*

$$\begin{cases} \delta_1 = mgu(term(s_1), term(r_1)), and \\ \delta_k = mgu(\delta_{k-1}(\dots\delta_1(s_k)), \delta_{k-1}(\dots\delta_1(r_k))),\ \forall 1 \leq k \leq l. \end{cases}$$

The composition $\delta = \delta_l \circ \dots \circ \delta_1$ *is the honest substitution for all the variables occurring in the protocol specification.*

The first two points ensure that each send action is followed immediately by a receive between the same two principals. Point 3 says that variables are only introduced when receiving a term. Point 4 claims that matching send and receive events must be unifiable, and compatible with the unification made on the previous actions.
A *role* of the protocol is the restriction of Π to the actions (send and receive) of one principal, as illustrated in the following example.

Example 1. The Needham-Schroeder protocol

$$\begin{aligned} \Pi^{NS} = {} & P_1 \ !\ P_2 : \{P_1, N_1^1\}_{\mathtt{pbk}(P_2)} \\ & P_2 \ ?\ P_1 : \{P_1, X_1^2\}_{\mathtt{pbk}(P_2)} \\ & P_2 \ !\ P_1 : \{X_1^2, N_1^2\}_{\mathtt{pbk}(P_1)} \\ & P_1 \ ?\ P_2 : \{N_1^1, X_1^1\}_{\mathtt{pbk}(P_1)} \\ & P_1 \ !\ P_2 : \{X_1^1\}_{\mathtt{pbk}(P_2)} \\ & P_2 \ ?\ P_1 : \{N_1^2\}_{\mathtt{pbk}(P_2)} \end{aligned}$$

The protocol has two principals, hence two roles described here.

$$\begin{aligned} \Pi_1^{NS} = {} & P_1 \ !\ P_2 : \{P_1, N_1^1\}_{\mathtt{pbk}(P_2)} & \Pi_2^{NS} = {} & P_2 \ ?\ P_1 : \{P_1, X_1^2\}_{\mathtt{pbk}(P_2)} \\ & P_1 \ ?\ P_2 : \{N_1^1, X_1^1\}_{\mathtt{pbk}(P_1)} & & P_2 \ !\ P_1 : \{X_1^2, N_1^2\}_{\mathtt{pbk}(P_1)} \\ & P_1 \ !\ P_2 : \{X_1^1\}_{\mathtt{pbk}(P_2)} & & P_2 \ ?\ P_1 : \{N_1^2\}_{\mathtt{pbk}(P_2)} \end{aligned}$$

2.2 The semantics

After having described the roles of a protocol, *i.e.* the way things should happen in a honest execution of the protocol, we will now describe how things really happen. In particular, we have to take into account the fact that a protocol can be executed several times, or by different agents, and that in each case the nonces and keys should be different, in order to ensure freshness.

We use the notion of *session* as a partial instantiation of one of the roles of the protocol. Since we do not assume the number of sessions to be bounded, we consider an infinite set $\Sigma = \{\sigma_n \mid n \in \mathbb{N}\}$ of *session ids*. In the same vein, we consider an infinite set $\mathcal{A} = \{a_n \mid n \in \mathbb{N}\} \cup \{\epsilon\}$ of *agents* that will play the roles of the protocol, with the special agent ϵ standing for the *intruder*.

The real values of nonces, session-keys and variables are instanciated with the session Id σ as well as the name of each participant taking part in this session. This yields three infinite sets:
$\mathfrak{K} = \{K_j^i(\sigma, b_1, \ldots, b_k) \mid K_j^i \in \mathcal{K},\ \sigma \in \Sigma,\ (b_1, \ldots, b_k) \in \mathcal{A}^k\}$ of *session-keys*, $\mathfrak{N} = \{N_j^i(\sigma, b_1, \ldots, b_k) \mid N_j^i \in \mathcal{N},\ \sigma \in \Sigma,\ (b_1, \ldots, b_k) \in \mathcal{A}^k\}$ of *nonces* and $\mathfrak{X} = \{X_j^i(\sigma, b_1, \ldots, b_k) \mid X_j^i \in \mathcal{X},\ \sigma \in \Sigma,\ (b_1, \ldots, b_k) \in \mathcal{A}^k\}$ of *variables*.
We do not need to consider the intruder as a normal agent that generates keys and nonces during a session. It is provided at the beginning with a set of nonces and session keys, one for each key/nonce that can be generated during the normal execution of the protocol. These sets are denoted

$\mathfrak{N}_\epsilon = \{\mathfrak{n}_j^i \mid i, j \textit{ s.t. } N_j^i \in \mathcal{N}\}$ and $\mathfrak{K}_\epsilon = \{\mathfrak{k}_j^i \mid i, j \textit{ s.t. } K_j^i \in \mathcal{K}\}$.

Using the above defined sets and the notions of long term keys, we can inductively define the set of (instanciated) terms.

$$\begin{aligned} \mathfrak{T} ::= {} & \mathcal{A} \mid \texttt{pbk}(\mathcal{A}) \mid \texttt{pvk}(\mathcal{A}) \mid \texttt{shk}(\mathcal{A}, \mathcal{A}) \mid \mathfrak{K} \mid \mathfrak{K}_\epsilon \mid \mathfrak{N} \mid \mathfrak{N}_\epsilon \\ & \mid \mathcal{C} \mid \mathfrak{X} \mid \langle \mathfrak{T}, \mathfrak{T} \rangle \mid \{\mathfrak{T}\}_{\mathfrak{T}} \mid \texttt{sig}_{\mathfrak{T}}(\mathfrak{T}) \end{aligned}$$

The set $\mathfrak{M}$ of actual messages exchanged on the network is the set of ground terms (*i.e.* without variables) and can be recursively defined with a grammar similar to the previous one. Based on this definition of instanciated terms, we define the set $\mathfrak{D} = \mathfrak{S} \cup \mathfrak{R}$ of possible instantiations of send and reveive actions.

In order to specify the correspondence between terms of $\mathcal{T}$ from the description of the protocol and instanciated terms of $\mathfrak{T}$, we define for every $(\sigma, b_1, \ldots, b_k) \in \Sigma \times \mathcal{A}^k$ the function $||.||_{(\sigma, b_1, \ldots, b_k))} : \mathcal{T} \to \mathfrak{T}$. This function associates to a principal P_j the agent b_j, to each nonce N_j^i (*resp.* session-key, variable) the corresponding instanciated object $N_j^i(\sigma, b_1, \ldots, b_k)$ and is defined inductively on encrypted subterms, pairs and more generally on terms. It can also be extended to actions by:
$||P_j!P_l : t||_{(\sigma, b_1, \ldots, b_k)} = ||P_j||_{(\sigma, b_1, \ldots, b_k)}!||P_l||_{(\sigma, b_1, \ldots, b_k)} : ||t||_{(\sigma, b_1, \ldots, b_k)}$ and a similar definition for receive actions.

A *substitution* is a map $\theta : \mathfrak{X} \Rightarrow \mathfrak{T}$. $\theta(t)$ or $t\theta$ will denote indifferently the application of substitution θ to term t. A *unifier* of two terms t and t' is a substitution θ such that $\theta(t) = \theta(t')$. The *most general unifier* of two terms t, t', denoted mgu(t, t'), is a unifier θ of t and t' such that for all unifer ψ of t and t' there exists a substitution ϕ such that $\psi = \phi \circ \theta$. We will denote the fact that two terms t and t' are not unifiable by mgu$(t, t') = \bot$.

The formal execution model is a state transition system. A global state of the system is given by (SId, q, I) where SId is a set of sessions, q is a function that describes the local state of each session in SId and $I \subseteq \mathfrak{M}$ represents the intruder's knowledge.
More precisely, $\forall \sigma \in SId$, $q(\sigma) = (i, b_1, \ldots, b_k, \theta, p)$ is the local state of session σ:

- i is the index of the role that is executed in this session,
- $(b_1, \ldots, b_k) \in \mathcal{A}^k$ are the identities of the parties that are involved in the session,
- θ is a partial instantiation of variables occuring in $||\Pi_i||_{(\sigma,b_1,\ldots,b_k)}$,
- p is the control point of the program.

Given a protocol Π, the initial state of Π is (SId_0, q_0, I_0), with $SId_0 = \emptyset$ (and thus the definition of q_0 is useless) and $I_0 = \mathcal{A} \cup \mathcal{C} \cup \mathfrak{K}_\epsilon \cup \mathfrak{N}_\epsilon \cup \{\mathtt{pbk}(a) \mid a \in \mathcal{A}\} \cup \{\mathtt{shk}(a, \epsilon), \mathtt{shk}(\epsilon, a) \mid a \in \mathcal{A}\} \cup \{\mathtt{pvk}(\epsilon)\}$ (the intruder knows the agent names, constants, his own session-keys and nonces, every agent's public key as well as his own private key and the keys he shares with other agents).
Let $Q = (SId, q, I)$ be a global state for Π. Three types of transitions $Q \xrightarrow{e} update(Q, e)$ may be allowed:

1. *Initiate a new session* for the i^{th} role ($e = \mathtt{new}(\sigma, i, b_1, \ldots, b_k)$):
   - Event e is enabled at state Q whenever the session σ does not belong to SId, the agent b_i is not the intruder and any two agents taking part in this new session are distinct.
   - The effect of firing this transition is $update(Q, e) = (SId \cup \{\sigma\}, q', I)$ with
     $\begin{cases} q'(\sigma') = q(\sigma'),\ \forall \sigma' \in SId \\ q'(\sigma) = (i, b_1, \ldots, b_k, \emptyset, 1). \end{cases}$
2. *Execute next send-action of an existing session* $\sigma \in SId$ ($e = \mathtt{send}(\sigma, p)$):
   - Event e is enabled at state Q whenever the control point of session σ is p and the next action to perform in σ is a send action.
   - The effect of firing this transition is $update(Q, e) = (SId, q', I \cup \{m\})$ with $m = \theta(||t||_{(\sigma,b_1,\ldots,b_k)})$ and
     $\begin{cases} q'(\sigma') = q(\sigma'),\ \forall \sigma' \in SId,\ \sigma' \neq \sigma \\ (q'(\sigma) = (i, b_1, \ldots, b_k, \theta, p + 1)). \end{cases}$
3. *Execute next receive-action of an existing session* $\sigma \in SId$ ($e = \mathtt{receive}(\sigma, p, m)$):
   - Event e is enabled at state Q whenever the control point of session σ is p and the next action to perform in σ is a receive action.
     - $m \in \mathfrak{M}$ is a message that can be computed by the intruder from I,
     - $q(\sigma) = (i, b_1, \ldots, b_k, \theta, p)$ (the control point of σ is p),
     - $\Pi_i(p) = P_i?_j : t$ (the next action is a receive),
     - $\psi \neq \bot$, where $\psi = mgu(m, \theta(||t||_{(\sigma,b_1,\ldots,b_k)}))$ (m and the expected message are unifiable).
   - The effect of firing this transition is $update(Q, e) = (SId, q', I)$ with
     $\begin{cases} q'(\sigma') = q(\sigma'),\ \forall \sigma' \in SId,\ \sigma' \neq \sigma \\ q'(\sigma) = (i, b_1, \ldots, b_k, \theta \cup \psi, p + 1). \end{cases}$

The adversary intercepts messages between honest participants and computes new messages using the deduction rule $\vdash$ defined in Fig.1. Intuitively $M \vdash m$ means that the adversary is able to compute the message m from the set of messages M. The notation m^{-1} stands for $\texttt{pbk}(a)$ if m is of the type $\texttt{pvk}(a)$, $\texttt{pvk}(a)$ if m is of the type $\texttt{pbk}(a)$, and $m^{-1} = m$ otherwise.

$$\frac{}{M \vdash m}\, m \in M$$

$$\frac{M \vdash m_1 \quad M \vdash m_2}{M \vdash \langle m_1, m_2\rangle} \qquad \frac{M \vdash \langle m_1, m_2\rangle}{M \vdash m_i}\, 1 \leq i \leq 2$$

$$\frac{M \vdash m_1 \quad M \vdash m_2}{M \vdash \{m_1\}_{m_2}} \qquad \frac{M \vdash \{m_1\}_{m_2} \quad M \vdash m_2^{-1}}{M \vdash m_1}$$

$$\frac{M \vdash m_1 \quad M \vdash m_2}{M \vdash \texttt{sig}_{m_1}(m_2)} \qquad \frac{M \vdash \texttt{sig}_{m_1}(m_2) \quad M \vdash m_1^{-1}}{M \vdash m_2}$$

Fig. 1. Deduction rules

Example 2. In order to better understand the effect of firing transition, in Fig.4 we have detailed a valid trace of length 7 for Π^{NS}. Due to space restrictions, the trace is given in appendix.

2.3 The secrecy problem

Let Π be an arbitrary k-party protocol. We say that Π guarantees the secrecy of nonce $N_j^i \in \mathcal{N}$ (*resp.* session-key $K_j^i \in \mathcal{K}$) if, in all possible executions, each honest instantiation of N_j^i (*resp.* K_j^i) remains unknown to the adversary.
More formally, we say that Π preserves secrecy of nonce $N_j^i \in \mathcal{N}$ (of session key *resp* $K_j^i \in \mathcal{K}$) if for every valid trace $(SId_0, s_0, I_0) \rightarrow^* (SId_n, s_n, I_n)$ of the protocol and for every $(b_1, \ldots, b_k) \in (\mathcal{A} \setminus \{\epsilon\})^k$ (*i.e.* k honest agents), we have $I_n \not\vdash N_j^i(\sigma, b_1, \ldots, b_k)$ (*resp.* $I_n \not\vdash K_j^i(\sigma, b_1, \ldots, b_k)$) for some $\sigma \in SId_n$.
We say that Π admits an attack on nonce $N_j^i \in \mathcal{N}$ (*resp.* session-key $K_j^i \in \mathcal{K}$) if there exists (SId_n, s_n, I_n) *s.t.* $(SId_0, s_0, I_0) \rightarrow^* (SId_n, s_n, I_n)$ and $b_1, \ldots, b_k \in \mathcal{A} \setminus \epsilon$, and we have $I_n \vdash N_j^i(\sigma, b_1, \ldots, b_k)$ (*resp.* $I_n \vdash N_j^i(\sigma, b_1, \ldots, b_k)$) for some $\sigma \in SId_n$

Example 3. An attack on Π^{NS}
The trace of Π^{NS} detailed in Fig.4 is an attack on the nonce N_1^2. Indeed, from I_7 the intruder can deduce $N_1^2(\sigma_2, a, b)$ which is encrypted with its own public key.

3 Well-formed protocols and well-typed attacks

In this section, we state the main result of the paper. We prove that for a *well-formed* protocol (*i.e.* with non unifiable subterms), for verification of secrecy properties we only need to consider well-typed runs of the protocol, *i.e.* for well-formed protocols the typing abstraction, with respect to the following type system, is correct.

3.1 Types

We introduce in this section a very strong typing on messages, that will allow us to restrict significantly the set of traces to consider in order to detect an attack. For example, nonces may have different types, depending on the role that generated them and the moment of the protocol when they were generated.

We first use a single type agent α for every principal name $P \in \mathcal{P}$. In particular, the intruder has the same type as any other agent.
To each session-key K_j^i in $\mathcal{K}$ (*resp.* nonce N_j^i in $\mathcal{N}$, constant c_i in $\mathcal{C}$), we associate a different type κ_j^i (*resp.* ν_j^i, γ_i). The notations κ, ν and γ denote respectively the set of session-key types, nonce types and constant types.

We thus obtain inductively the following type set for terms:
$\tau ::= \alpha \mid \kappa \mid \nu \mid \gamma \mid \mathtt{pbk}(\alpha) \mid \mathtt{pvk}(\alpha) \mid \mathtt{shk}(\alpha,\alpha) \mid \langle \tau,\tau \rangle \mid \{\tau\}_\tau \mid \mathtt{sig}_\tau(\tau)$

The typing rules are given in Fig.2

$$\begin{array}{ccc}
\dfrac{P \in \mathcal{P}}{P : \alpha} & \dfrac{c_i \in \mathcal{C}}{c_i : \gamma_i} & \dfrac{K_j^i \in \mathcal{K}}{K_j^i : \kappa_j^i} \\[2ex]
\dfrac{N_j^i \in \mathcal{N}}{N_j^i : \nu_j^i} & \dfrac{P \in \mathcal{P}}{\mathtt{pbk}(P) : \mathtt{pbk}(\alpha)} & \dfrac{P \in \mathcal{P}}{\mathtt{pvk}(P) : \mathtt{pvk}(\alpha)} \\[2ex]
\dfrac{P, P' \in \mathcal{P}}{\mathtt{shk}(P,P') : \mathtt{shk}(\alpha,\alpha)} & \dfrac{t_1 : \tau_1 \quad t_2 : \tau_2}{f(t_1,t_2) : f(\tau_1,\tau_2)} & \dfrac{X \in \mathcal{X} \quad \delta(X) : \tau}{X : \tau} \\[2ex]
\dfrac{t : \tau}{||t||_{(\sigma,b_1,\ldots,b_k)} : \tau} & \dfrac{\mathfrak{k}_j^i \in \mathfrak{K}_\epsilon}{\mathfrak{k}_j^i : \kappa_j^i} & \dfrac{\mathfrak{n}_j^i \in \mathfrak{N}_\epsilon}{\mathfrak{n}_j^i : \nu_j^i}
\end{array}$$

Fig. 2. Typing rules

Definition 2. *A* well-typed *run is a valid trace* $(SId_0, q_0, I_0) \rightarrow^* (SId_n, q_n, I_n)$ *such that for every session id* $\sigma \in SId_n$ *with* $q_n(\sigma) = (i, b_1, \ldots, b_k, \theta, p)$*, for every variable* $X \in dom(\theta)$*,* $X : \tau \Rightarrow \theta(X) : \tau$.

This definition states that each variable used in the specification is always instanciated (using substitution θ) by a message with the expected type.

Example 4. The run detailed in Fig.4 is well-typed. Indeed,
$X_1^1(\sigma_1, a, \epsilon) : \nu_1^2$ and $\theta(X_1^1(\sigma_1, a, \epsilon)) = N_1^2(\sigma_2, a, b) : \nu_1^2$
$X_1^2(\sigma_2, a, b) : \nu_1^1$ and $\theta(X_1^2(\sigma_2, a, b)) = N_1^1(\sigma_1, a, \epsilon) : \nu_1^1$

The following definition constrains unifiability between subterms of different type.

Definition 3. *A protocol Π (as defined in the previous section) is said to be* well-formed *when the following condition holds:*
$\forall t, t' \in ESt(\Pi)$*, if there exist* $(\sigma, b_1, \ldots, b_k)$*,* $(\sigma', b'_1, \ldots, b'_k) \in \Sigma \times \mathcal{A}^k$ *and a substitution* θ *such that* $\theta(||t||_{(\sigma,b_1,\ldots,b_k)}) = \theta(||t'||_{(\sigma',b'_1,\ldots,b'_k)})$*, then* $\delta(t) = \delta(t')$.

As claimed in the introduction, this condition is often met in practice in the literature (see [CJ97]) and even when the protocol isn't well-formed, a light tagging scheme as used in [BP05] in which a different label is introduced at every encryption step of the specification, ensures well-formedness. We present such a tagging scheme in definition 7 in the context of the decidability results of [RS03a].

3.2 Considering only well-typed runs for well-formed protocols

We now state the main result of this paper. Due to a lack of space its proof is given in appendix B. We will give here the main ideas of the proof.

Theorem 1. *Let Π be a well-formed protocol. If Π admits an attack, then Π admits a well-typed attack.*

The proof is based on the fact that if a protocol admits an attack then it admits an attack of bounded length n, which can thus be found. The proof of theorem 1 is done by induction on a procedure searching for this attack. Indeed, we show that the considered procedure from [CZ06] instantiates variables only with terms of the expected type. We will first detail this procedure and then come back to explanations about well-typedness of computed substitutions.

The secrecy problem for security protocols can be translated into a constraint satisfaction problem [MS01,CZ06,RT01]. In [CZ06] it is shown that using some simplification rules, solving general constraints can be reduced to solving simpler constaint systems that are called *solved*.

Definition 4. *[CZ06] A constraint system C is a finite set of expressions $T_i \Vdash tt$ or $T_i \Vdash u_i$ where $T_i \subseteq \mathfrak{T}$, $T_i \neq \emptyset$, tt is a special symbol that represents an always deducible term, and $u_i \in \mathfrak{T}$, $1 \leq i \leq n$, such that:*

- $T_i \subseteq T_{i+1},\ \forall i,\ 1 \leq i \leq n-1$;
- *if $X \in \mathcal{V}(T_i)$, then $\exists j<i$ such that $T_j = \min\{T \mid T \Vdash u \in C, X \in \mathcal{V}(u)\}$ (for the inclusion relation) and $T_j \subsetneq T_i$*

$\perp$ denotes the unsatisfiable system. A constraint system is said to be solved if it is different from $\perp$ and each of its constraints are of the form $T \Vdash tt$ or $T \Vdash X$, where $X \in \mathfrak{X}$.

The left-hand side of the constraint $T \Vdash u$ is T and u is its right-hand side. The left-hand side $lhs(C)$ of the constraint system C is the maximal left-hand side of its constraints, and the right-hand side $rhs(C)$ of C is the set of messages in the right-hand side of its constraints. We consider the following sets over C defined as expected: $\mathcal{V}(C) = \mathcal{V}(lhs(C)) \cup \mathcal{V}(rhs(C))$, $terms(C) = lhs(C) \cup rhs(C)$, $St(C) = St(terms(C))$ and $ESt(C) = ESt(terms(C))$.

The simplification rules we consider are defined in Fig.3. They have been proven correct, complete and terminating in polynomial time [CZ06][1].

R_1	$C \wedge T \Vdash u \rightsquigarrow_{\emptyset} C \wedge T \Vdash tt$	if $T \cup \{X \mid T' \Vdash X \in C, T' \subsetneq T\} \vdash u$
R_2	$C \wedge T \Vdash u \rightsquigarrow_{\psi} C\psi \wedge T\psi \Vdash u\psi$	if $\psi = \text{mgu}(t, u), t \in ESt(T)$ $t \neq u, u$ not variable
R_3	$C \wedge T \Vdash u \rightsquigarrow_{\psi} C\psi \wedge T\psi \Vdash u\psi$	if $\psi = \text{mgu}(t_1, t_2), t_1, t_2 \in ESt(T)$ $t_1 \neq t_2$
R_4	$C \wedge T \Vdash u \rightsquigarrow_{\emptyset} \bot$	if $\mathcal{V}(T, u) = \emptyset$ and $T \not\vdash u$
R_f	$C \wedge T \Vdash f(u, v) \rightsquigarrow_{\emptyset} C \wedge T \Vdash u \wedge T \Vdash v$	for $f \in \{\langle \_, \_ \rangle, \{\_\}_{\_}, \mathtt{sig}_{\_}(\_)\}$

Fig. 3. Simplification rules

We now give a proof sketch of theorem 1.

The proof is done by induction, as said before, on the length n of a sequence of simplifications leading to an attack:

$$C_0 \rightsquigarrow_{\theta_1}^{R_{i_1}} C_1 \rightsquigarrow_{\theta_2}^{R_{i_2}} \cdots \rightsquigarrow_{\theta_n}^{R_{i_n}} C_n$$

where C_0 is the initial constraint system corresponding to the interleaving of the considered attack, and C_n is a solved constraint system. Such a C_n exists since the protocol admits an attack with this interleaving, and the simplification rules are correct, complete and terminating.
It is easy to see that rules R_1, R_4 and R_f preserve well-typedness since they do not instantiate any variable.
For rules R_2 and R_3 we show, by means of some lemmas given in appendix A (again due to a lack of space), that the selected substerms t and t', such that $\psi_j = \text{mgu}(t, t')$ $1 \leq j \leq n$ when $R_{i_j} \in \{R_2, R_3\}$, are of the same type and that computing the mgu of two terms of the same type results in a well-typed substitution. This allows us to conclude that, when applying these rules, variables are instantiated only with terms of the expected type, and thus they preserve well-typedness.

The following corollary is an immediate consequence of the previous theorem and of the fact that function application (pairing, encrypting and signing) is embedded in the type of a term.

Corollary 1. *Let Π be a well-formed protocol. If Π admits an attack, then Π admits an attack with messages of bounded length.*

We have thus proved in this section that the encryption abstraction is correct for well-formed protocols. And that this holds for a much more fine-grained type notion than the one considered in [LYH04,HLS03], where all nonces and session keys are of the same type. This severely restricts the search space to consider for verification purposes. We now define a tagging scheme that ensures well-formedness.

[1] Actually, the rules presented in [CZ06] allow unification between pairs in rules R_2 and R_3. But since then, the authors have let us know that we could just consider unification between encrypted terms. Thus, the rules presented in Fig.3 slightly differ from, but are equivalent to, the original rules in [CZ06].

Definition 5. *A tagged protocol is a protocol (as defined in the previous section)* s. t.*:*
$\forall t \in ESt(\Pi),\ \exists c \in \mathcal{C}$ *and* $t_1, t_2 \in \mathfrak{T}$ s.t. $t = \{c, t_1\}_{t_2}$ *or* $t = \mathtt{sig}_{t_1}(c, t_2)$*,*
$\forall t, u \in ESt(\Pi)$ s.t. $t = f(l, t_1, t_2)$ *and* $u = g(l, u_1, u_2)$*, then* $f = g$ *and* $\delta(t) = \delta(u)$*.*

It immediately follows that such tagged protocols verify well-formedness. This tagging scheme is extremely lighter than the ones in [LYH04,HLS03] where the whole type of an encrypted subterm is used for tagging it. We have thus obtained a more refined type abstraction with a very simple tagging scheme.

4 Application to decidability results

As claimed in the introduction, the type assumption is often necessary in order to obtain decidability and in particular in the presence of an unbounded number of sessions and nonces. Indeed, Lowe in [Low99] as well as Ramanujam and Suresh in [RS03a] prove the decidability of a class of protocols but assume that messages are of bounded length.

In [RS03a], the authors prove decidability of the secrecy problem (for a stronger definition of secrecy than the one given in section 2.3) in the framework of messages of bounded length, for a class of protocols they call structured. Since structured protocols do not admit blind copies, we can slightly strengthen their definition in order to ensure well-formedness, and we claim that this does not restrict the class of protocols from a semantic point of view. Indeed, any structured protocol in the sence of [RS03a] can easiliy be transformed in a well-structured protocol in the sense of definition 6 (because of the absence of blind copies) without changing its purported “meaning”. Hence we can combine the decidability result in [RS03a] and theorem 1.

Definition 6. *A protocol* $\Pi = s_1 r_1 \ldots s_l r_l$ *is said to be well-structured if the following conditions hold:*

- Π *doesn’t have blind copies, each variable is of atomic type,*
- *keys are atomic,*
- *encrypted subterms are textually distinct, an encrypted subterm* t *of a protocol in the described class can be unified only with its matching send or receive* t'*.*

The above definition constrains unifiability of different subterms (even of the same message) whereas the one of [RS03a] only constrains unifiability of subterms of different messages.
As already argued above, the additional restriction is not severe and acceptable as it yields decidability for unbounded messages. Moreover, since two encrypted subterms of the protocol $t, t' \in ESt(\Pi)$ are unifiable *iff* the one is the send or receive message of the other, it is the case that $\delta(t) = \delta(t')$. Thus well-structured protocols as defined in definition 6 are well-formed, which permits us to conclude to the decidability of well-structured protocols in the frame of unbounded message length.

One way of ensuring well-structuredness may be by means of tags/labels.

Definition 7. *A protocol* $\Pi = s_1 r_1 \ldots s_l r_l$ *is a tagged protocol if it satisfies the folowing conditions:*

– *no blind copies,*
– *keys are atomic,*
– $\forall t \in ESt(\Pi),\ \exists c \in \mathcal{C},\ \exists t_1, t_2 \in \mathfrak{T}$ *such that* $t = \{c, t_1\}_{t_2}$ *or* $t = \mathtt{sig}_{t_2}(c, t_1)$
– $\forall c, c' \in \mathcal{C}, \forall i \neq j,\ 1 \leq i, j \leq l,$ *if* $f(c, t_1, t_2) \in ESt(s_i)$ *then* $\forall f(c', u_1, u_2) \in ESt(s_j),\ c \neq c'$
– $\forall c \in \mathcal{C}, \forall i\ 1 \leq i, j \leq l,\ \forall p \in \mathbb{N}^*$, *if* $term(s_i)|_p = f(c, t_1, t_2)$ *then* $\forall q \mathbb{N}^*$ s.t. $q \neq p\ \wedge\ s_i|_q = f(c', u_1, u_2),\ c \neq c'$.

The third condition is similar to all definitions of tagged protocols [BP05,RS03b]. The fourth condition stipulates that each tag is use at most in one send event; and the fifth, that a tagged use in a send event is used in at moste one position. We have thus constraind to be used exactly once in the protocol. Thus any encrypted subterm is unifiable with and only with its matching send or receive action. Therefore tagged protocols are well-structured, and we can hence conclude to decidability of the secrecy problem for tagged protocols. The secrecy problem, for the class of tagged protocols, is shown to be decidable in [RS03b], but the considered tagging scheme is heavier. Indeed, in the above definition a few bits are sufficient to tag messages, whereas in [RS03b] each encrypted subterm of the protocol is tagged with a pair (c, N) where c is a constant and N is a different nonce making the tagging scheme heavier.

5 Conclusion

The result presented in this paper is a first step towards decidability. We have proven that for a well known and wide class of protocols (that we call well-formed) the type abstraction is correct. Therefore there is no need to check for badly typed attacks, and this restricts the search space to consider in order to prove secrecy for a protocol. This was achieved in a much more economic way than in [LYH04,HLS03]. Further, the type abstraction is here significantly refined.
In addition, we have shown how this result can improve existing decidability results. We also believe it could improve the efficiency of existing tools by restricting their search space to well-typed executions.
Our next goal is to use our theorem to get decidability for (at least a large subclass of) our well-formed protocols. The idea is that protocols at stake in existing undecidability proofs lie outside our framework. They are either not executable (*i.e.* in an honest execution, some action of the specification can never occur) or not well-formed (*i.e.* they allow, for example, to replay a message generated in a session at step m in another session at step $n < m$ without the agents noticing it). We expect that the restriction to well-formed and executable protocols will lead to a larger decidable subclass.

References

[Bla01] B. Blanchet. An efficient cryptographic protocol verifier based on prolog rules. In *Proc. 14th IEEE work. on Computer Security Foundations (CSFW'01)*, page 82. IEEE, 2001.

[BP05] B. Blanchet and A. Podelski. Verification of cryptographic protocols: Tagging enforces termination. *TCS: Theoretical Computer Science*, 333, 2005.

[CC01] H. Comon and V. Cortier. Tree automata with one memory, set constraints and cryptographic protocols. Research Report LSV-01-13, Laboratoire Spécification et Vérification, ENS Cachan, France, 2001. 98 pages.

[CC03] H. Comon-Lundh and V. Cortier. New decidability results for fragments of first-order logic and application to cryptographic protocols. In *Proc. 14th International Conference on Rewriting Techniques and Applications (RTA'03)*, volume 2706 of *LNCS*, pages 148–164. Springer, 2003.

[CDL06] V. Cortier, S. Delaune, and P. Lafourcade. A survey of algebraic properties used in cryptographic protocols. *Journal of Computer Security*, 14(1):1–43, 2006.

[CJ97] J.A. Clark and J.L. Jacob. A survey of authentication protocol literature, 1997.

[CJM00] E. M. Clarke, S. Jha, and W. Marrero. Verifying security protocols with brutus. *ACM Trans. Softw. Eng. Methodol.*, 9(4):443–487, 2000.

[CKR$^+$03] Y. Chevalier, R. Küsters, M. Rusinowitch, M. Turuani, and L. Vigneron. Extending the Dolev-Yao intruder for analyzing an unbounded number of sessions. In *Proc. 17th Int. Work. on Computer Science Logic (CSL03)*, volume 2803 of *LNCS*, pages 128–141. Springer, 2003.

[CMR01] V. Cortier, J. Millen, and H. Rueß. Proving secrecy is easy enough. In *Proc. 14th IEEE work. on Computer Security Foundations (CSFW'01)*, page 97. IEEE, 2001.

[CZ06] V. Cortier and E. Zalinescu. Deciding key cycles for security protocols. In *LPAR*, pages 317–331, 2006.

[DLMS99] N. Durgin, P. Lincoln, J. Mitchell, and A. Scedrov. Undecidability of bounded security protocols. In *Proc. Work. on Formal Methods and Security Protocols (FMSP)*, 1999.

[HLS03] J. Heather, G. Lowe, and S. Schneider. How to prevent type flaw attacks on security protocols. *Journal of Computer Security*, 11(2):217–244, 2003.

[Low99] G. Lowe. Towards a completeness result for model checking of security protocols. *Journal of Computer Security*, 7(1), 1999.

[LYH04] Y. Li, W. Yang, and C-W. Huang. Preventing type flaw attacks on security protocols with a simplified tagging scheme. In *Proc. Int. symp. on Information and communication technologies (ISICT'04)*, pages 244–249. Trinity College Dublin, 2004.

[Mea96] C. Meadows. The NRL protocol analyzer: An overview. *Journal of Logic Programming*, 26(2):113–131, 1996.

[MM82] A. Martelli and U. Montanari. An efficient unification algorithm. *ACM Trans. Program. Lang. Syst.*, 4(2):258–282, 1982.

[MS01] J.K. Millen and V. Shmatikov. Constraint solving for bounded-process cryptographic protocol analysis. In *Proc. 8th ACM Conf. on Computer and Communications Security*, pages 166–175. ACM Press, 2001.

[RS03a] R. Ramanujam and S. P. Suresh. A decidable subclass of unbounded security protocols. In *Proc. Work. on Issues in the Theory of Security (WITS'03)*, 2003.

[RS03b] R. Ramanujam and S. P. Suresh. Tagging makes secrecy decidable with unbounded nonces as well. *FSTTCS: Foundations of Software Technology and Theoretical Computer Science*, 2914, 2003.

[RT01] M. Rusinowitch and M. Turuani. Protocol insecurity with finite number of sessions is NP-complete. In *Proc. 14th IEEE work. on Computer Security Foundations (CSFW'01)*, page 174. IEEE, 2001.

[SBP01] D.X. Song, S. Berezin, and A. Perrig. Athena: a novel approach to efficient automatic security protocol analysis. *Journal of Computer Security*, 9(1-2):47–74, 2001.

A Some lemmas

As said before, the proof of the main theorem necessitates some lemmas that we give here.

The first lemma stipulates that the most general unifier of two terms of the same type is well-typed.

Lemma 1. *Let $t, u \in \mathfrak{T}$ and* $\mathrm{mgu}(t, u) \neq \bot$ *(*i.e. *t and u are unifiable), if $t : \tau$ and $u : \tau$ (*i.e. *t and u are of the same type), then* $\mathrm{mgu}(t, u)$ *is well typed.*

The proof of lemma 1 is done by induction on the algorithm that computes the most general unifier, the one we have chosen here is the one in [MM82]. Before proceeding with the proof, we need to recall some notions introduced in [MM82] as well as the unification procedure.

The unification problem $\mathrm{mgu}(t, u)$ can be written as the equation $t = u$. We will need to consider sets of equations $\{t_i = u_i | t_i, u_i \in \mathfrak{T}\}_{1 \leq i \leq k}$. A set of equations is said to be *in solved form iff* the two following conditions hold:

1. the equations are $X_i = t_i,\ X_i \in \mathfrak{X},\ 1 \leq i \leq k$,
2. every variable which is the left member of some equation occurs only there.

The following algorithm computes a solved set of equations equivalent to the one given in argument. In [MM82] it is shown that the resulting solved set of equations is the most general unifier of the initial one. Thus in order to compute $\mathrm{mgu}(t, u)$ for some terms t and u, we apply this algorithm to the set of equations $\{t = u\}$.

Algorithm *[MM82] Given a set of equations, repeatedly perform any of the following transformations. If no transformation applies, stop with success.*

(a) Select any equation of the form $t = X$, where $t \notin \mathfrak{X}$, and $X \in \mathfrak{X}$ and rewrite it as $X = t$.
(b) Select any equation of the form $X = X$, where $X \in \mathfrak{X}$, and erase it.
(c) Select any equation of the form $s = v$, where $s, v \notin \mathfrak{X}$. If the two function symbols are different, stop with failure; otherwise (i.e. $s = f(s_1, \ldots, s_n)$ and $v = f(v_1, \ldots, v_n)$), replace this equation by the following ones: $s_1 = v_1, \ldots, s_n = v_n$. In particular if f is a constant (i.e. $n = 0$) then just erase the selected equation.
(d) Select any equation of the form $X = t$ (i.e. the equation set E is of the form $E' \cup \{X = t\}$ for some equation set E'), where $X \in \mathfrak{X},\ t \neq X$. If X occurs in t, then stop with failure; otherwise, replace E with the set ($E'[X \leftarrow t] \cup \{X = t\}$)

We say that an equation $t = t'$ is *well-typed* whenever t and t' are of the same type. A set of equations is well-typed when all the equations in the set are well-typed.

We can now proceed with the proof of lemma 1.

Proof (lemma 1). Let $t, u \in \mathfrak{T}$ such that $t : \tau$, $u : \tau$, and $\theta = \text{mgu}(t, u)$. The substitution θ can be computed following the above algorithm. The initial set of equations is $\{t = u\}$, and since t and u are by hypothesis of the same type, then the initial set of equations is well-typed. It is thus sufficient to prove that an iteration of the algorithm preserves well-typedness, that is when an iteration of this algorithm is applied to a well-typed set of equations E, then the resulting set E' of equations is well-typed.

– If the applied rule is (a) then the resulting set is clearly well-typed since we only replace a well-typed equation $s = X$ by $X = s$.
– If the applied rule is (b) then we replace E well-typed by $E' \subsetneq E$ which is trivially well-typed.
– If the applied rule is (c) and since $\theta \neq \perp$, then E is either of the form $E'' \cup \{f(t_1, t_2) = f(u_1, u_2)\}$, with $f \in \{\langle \_, \_ \rangle, \{\_\}_\_, \mathtt{sig}_\_(\_)\}$ and $t_1, t_2, u_1, u_2 \in \mathfrak{T}$ or of the form $E'' \cup \{s = v\}$, with $s, v \in \mathfrak{T}_0$ for some set of equations E''. In the first case the resulting set is $E' = E'' \cup \{t_1 = u_1, t_2 = u_2\}$. From the typing rules in Fig.2 and since $f(t_1, t_2)$ and $f(u_1, u_2)$ are of the same type $f(\tau_1, \tau_2)$, then $\{t_1 = u_1, t_2 = u_2\}$ is well-typed as well as $E'' \cup \{t_1 = u_1, t_2 = u_2\}$. Thus, E' is well-typed. In the second case $E' = E''$ and it is thus immediate that E' is well-typed.
– If the applied rule is (d), then E is of the form $E'' \cup \{X = s\}$ for some set of equations E'' and some $s \in \mathfrak{T}$ and $X \in \mathfrak{X}$, and since $\theta \neq \perp$, X does not occur in s. The resulting equation set is $E''[X \leftarrow s] \cup \{X = s\}$. Now, for every equation $v = w$ in E'' since X and s are of the same type ($X = s$ is well typed), then $v[X \leftarrow s]$ and v (*resp.* $w[X \leftarrow s]$ and w) are of the same type , thus $v[X \leftarrow s] = w[X \leftarrow s]$ is well-typed, and finally $E' = E''[X \leftarrow s] \cup \{X = s\}$ is well-typed. □

Because lemma 4, needed for the proof of the main theorem, is rather technical and not intuitive at all, we separate the steps of the proof by the means of the two following lemmas, and that in order to make it more comfortable to the reader.

Lemma 2. *For every pair of terms $t, u \in \mathfrak{T}$ and every substitution θ, if $u \in ESt(t\theta)$, then $(u \in \theta(ESt(t)))$ or $(\exists X \in \mathcal{V}(t)\ s.t.\ u \in ESt(X\theta))$.*

Proof. We prove this by induction on the structure of t:

– If $t \in \mathfrak{T}_0$, then $t\theta = t$ and $ESt(t\theta) = \emptyset$, thus $u \notin ESt(t\theta)$, and the implication trivially holds.
– If $t = X \in \mathfrak{X}$, then $t\theta = X\theta$ and since $u \in ESt(t\theta)$, then $u \in ESt(X\theta)$, and thus the second part of the disjunction holds.
– If $t = \langle t_1, t_2 \rangle$ for some $t_1, t_2 \in \mathfrak{T}$, then $t\theta = \langle t_1\theta, t_2\theta \rangle$, and thus $u \in ESt(t_1\theta) \vee u \in ESt(t_2\theta)$. By induction we know that $u \in \theta(ESt(t_i))$ or $(\exists X \in \mathcal{V}(t_i)\ s.t.\ u \in ESt(\theta(X)))$, $1 \leq i \leq 2$. Finally, for i, $1 \leq i \leq 2$, if $u \in \theta(ESt(t_i))$, since $ESt(t_i) \subseteq ESt(t)$, then $\theta(ESt(t_i)) \subseteq \theta(ESt(t))$ and thus $u \in \theta(ESt(t))$; else $\exists X \in \mathcal{V}(t_i)\ s.t.\ u \in ESt(\theta(X))$ and since $\mathcal{V}(t_i) \subseteq \mathcal{V}(t)$ then we trivially have that $\exists X \in \mathcal{V}(t)\ s.t.\ u \in ESt(\theta(X))$.
– If $t = f(t_1, t_2)$ with $t_1, t_2 \in \mathfrak{T}$ and $f \in \{\{\_\}_\_, \mathtt{sig}_\_(\_)\}$. We need to distinguish two cases:

1. $u = t\theta$
   By definition of ESt, we have that $t \in ESt(t)$ and thus, $t\theta \in \theta(ESt(t))$, so the first part of the disjunction holds.
2. $u \neq t\theta$
   As in the previous case with the pairing function $u \in ESt(t_1) \vee u \in ESt(t_2)$ and thus we can conclude with the same argument as we did above. $\square$

For the following lemma we introduce a bit of notation. Let $E = \{t_1 = u_1, \ldots, t_n = u_n\}$ be a set of equation, we define the set of terms of E and the set of encrypted subterms of E as expected:

$$terms(E) = \bigcup_{i=1}^{i=n}\{t_i, u_i\} \qquad \text{and} \qquad ESt(E) = ESt(terms(E)).$$

Furthermore, we will denote $E \rightarrow_{\psi}^{R} E'$ an iteration of the algorithm computing the mgu and presented above, where E' is the resulting equation set when rule $R \in \{a, b, c, d\}$ is applied to an equation $t_i = u_i$ of E for some $i,\ 1 \leq i \leq n$ and $\psi = \emptyset$ if $R \in \{a, b, c\}$ else $\psi = [t_i \leftarrow u_i]$.

Lemma 3. *For every tuple of terms $s, t, u \in \mathfrak{T}$ and $X \in \mathcal{V}(\{t, u\})$* s.t. $\text{mgu}(t, u) = \psi \neq \bot$, *if $s \in ESt(\psi(X))$, then there exists $s' \in ESt(\{t, u\})$* s.t. $s = \psi(s')$.

Proof. Let $E_0 = \{t = u\} \rightarrow_{\psi_1}^{R_1} E_1 \rightarrow_{\psi_2}^{R_2} \cdots \rightarrow_{\psi_n}^{R_n} E_n$ be a sequence of sets of equations computed by the unifying algorithm. We show by induction on the length n of the computation, that $\forall s \in ESt(E_n),\ \exists s' \in ESt(\{t, u\})\ s.t.\ s = \psi_n \circ \cdots \circ \psi_1(s')$. If $n = 0$, then it is immediate that the statement holds. Suppose now that for $0 \leq n \leq p$ it is true, and let $n = p + 1$, then we have a sequence of computations of the form:
$E_0 = \{t = u\} \rightarrow_{\psi_1}^{R_1} E_1 \rightarrow_{\psi_2}^{R_2} \cdots \rightarrow_{\psi_p}^{R_p} E_p \rightarrow_{\psi_{p+1}}^{R_{p+1}} E_{p+1}$

– If $R_{p+1} \in \{a, b, c\}$, then $\psi_{p+1} = \emptyset$ and $ESt(E_{p+1}) \subseteq ESt(E_p)$, thus by induction we have that $\forall s \in ESt(E_{p+1}),\ \exists s' \in ESt(\{t, u\})\ s.t.\ s = \psi_p \circ \cdots \circ \psi_1(s')$, and since $\psi_{p+1} = \emptyset$ we have that $\psi_{p+1} \circ \psi_p \circ \cdots \circ \psi_1 = \psi_p \circ \cdots \circ \psi_1$. Thus $s = \psi_{p+1} \circ \psi_p \circ \cdots \circ \psi_1(s')$ for some $s' \in ESt(\{t, u\})$
– If $R_{p+1} = d$, then the selected equation is of the form $X = v$ for some $X \in \mathfrak{X}$ and some $v \in \mathfrak{T}$, E_p is of the form $E \cup \{X = v\}$ and $E_{p+1} = E[X \leftarrow v] \cup \{X = v\}$. Let $s \in ESt(E_{p+1})$. If $s \in ESt(\{X = v\})$, then $s \in ESt(v)$ and since $X = v \in E_p$ by induction we know that $\exists s' \in ESt(\{t, u\})$ such that $s = \psi_p \circ \cdots \circ \psi_1(s')$ and finally we know that if rule (d) is applied then X does not occur in v, thus $s = \psi_{p+1} \circ \psi_p \circ \cdots \circ \psi_1(s')$.
  Now we need to consider the case where $s \in ESt(E\psi_{p+1})$. That means that there exists some $w \in terms(E\psi_{p+1})$ such that $s \in ESt(w)$ and thus there exists some $w' \in terms(E) \subseteq E_p$ such that $w = w'\psi_{p+1}$, *i.e.* $s \in ESt(w'\psi_{p+1})$ for some $w' \in terms(E_p)$. From lemma 2 we know that $s \in (\psi_{p+1}(ESt(w'))) \vee (\exists Y \in \mathcal{V}(w'),\ s \in ESt(\psi_{p+1}(Y)))$. In the first case we can conclude by induction that $\exists s' \in ESt(\{t, u\})$ such that $s = \psi_{p+1} \circ \psi_p \circ \cdots \circ \psi_1(s')$; and in the second it is obvious that it is X that is in $\mathcal{V}(w')$ such that $s \in ESt(\psi_{p+1}(X)) = ESt(v)$ since X is the only variable in $dom(\psi_{p+1})$. And as we showed before, there exists $s' \in ESt(\{t, u\})$ such that $s = \psi_{p+1} \circ \cdots \circ \psi_1(s')$.

We have finally proved that for any sequence of computations $E_0 = \{t = u\} \rightarrow^{R_1}_{\psi_1} E_1 \rightarrow^{R_2}_{\psi_2} \cdots \rightarrow^{R_n}_{\psi_n} E_n$, we have that $\forall s \in ESt(E_n),\ \exists s' \in ESt(\{t, u\})\ s.t.\ s = \psi_n \circ \cdots \circ \psi_n(s')$ and in particular for E_n in solved form (*i.e.* $E_n = \text{mgu}(t, u)$)). Now it is easy to see that $\psi = \psi_n \circ \ldots \psi_1$, thus we can conclude that if $s \in ESt(\psi(X))$, then there exists $s' \in ESt(\{t, u\})$ *s.t.* $s = \psi(s')$. $\square$

This fourth lemma necessitates introducing some more notations. In what follows, we will denote the following simplifications

$$C_0 \rightsquigarrow_{\psi_1} C_1 \rightsquigarrow_{\psi_2} \cdots \rightsquigarrow_{\psi_n} C_n$$

by $C_0 \rightsquigarrow^n_\theta C_n$, where $\theta = \psi_n \circ \cdots \circ \psi_2 \circ \psi_1$.

Lemma 4. *Let Π be a protocol, C_0 a constraint system corresponding to some interleaving of Π's sessions and, $C_0 \rightsquigarrow^n_\theta C_n$ a sequence of simplifications according to the rules in Fig.3. The following statement holds:*

$$\forall t \in ESt(C_n),\ \exists t' \in ESt(\Pi)\ s.t.\ t = \theta(||t'||_{(\sigma, b_1, \ldots, b_k)})$$

for some $(\sigma, b_1, \ldots, b_k)$ in $\Sigma \times \mathcal{A}^k$.

Proof. We do this by induction on the length n of the simplification sequence.
If $n = 0$, then by construction of C_0 we have that for every term u that occurs in C_0, there exists a term u' of Π such that $u = ||u'||_{(\sigma, b_1, \ldots, b_k)}$ for some $(\sigma, b_1, \ldots, b_k) \in \Sigma \times \mathcal{A}^k$. Since $t \in ESt(C_0)$, then there exists some $u \in terms(C_0)$, and thus from what we just said, $\exists u' \in terms(\Pi)$ and some $(\sigma, b_1, \ldots, b_k)$ such that $t \in ESt(||u'||_{(\sigma, b_1, \ldots, b_k)})$ and thus $\exists t' \in ESt(u') \subseteq ESt(\Pi)$ such that $t = ||t'||_{(\sigma, b_1, \ldots, b_k)}$.
We suppose that for $0 \leq n \leq p$ the hypothesis holds.
Now, if $n = p + 1$, then the reduction sequence is of the form $C_0 \rightsquigarrow^n_\theta C_p \rightsquigarrow^{R_i}_\psi C_{p+1}$ for some substitution θ and some constraint system C_p.

- If $i \in \{1, 4, f\}$, $ESt(C_{p+1}) \subseteq ESt(C_p)$, and we can conclude immediately using the induction hypothesis.
- $i = 2 \Rightarrow C_p = C \wedge T \Vdash u$ for some C, T and u, and $\psi = \text{mgu}(t, u)$ for some $t \in ESt(T)$.
  Let $s \in ESt(C_{p+1})$. By ESt's definition it is the case that there exists $v \in terms(C_{p+1})$ such that $s \in ESt(v)$, and since $C_{p+1} = C_p\psi$, there exists $w \in terms(C_p)$ such that $v = w\psi$, and thus $s \in ESt(w\psi)$ for some w in C_p. From lemma 2 we have that either $s \in \psi(ESt(w))$, or $\exists X \in \mathcal{V}(w)$ and $s \in ESt(\psi(X))$. We consider these two cases separately:
  1. If $s \in \psi(ESt(w))$ and since $w \in terms(C_p)$, we have that $s \in \psi(ESt(C_p))$, *i.e.* $s = w'\psi$ for some $w' \in ESt(C_p)$. Now, by induction we know that there exists $w'' \in ESt(\Pi)$ such that $w' = \theta(||w''||_{(\sigma, b_1, \ldots, b_k)})$, and thus $s = \psi(\theta(||w''||_{(\sigma, b_1, \ldots, b_k)}))$.
  2. If $\exists X \in \mathcal{V}(w)$ and $s \in ESt(\psi(X))$, from lemma 3 we know that there exists $s' \in ESt(\{t, u\}) \subseteq ESt(C_p)$ *s.t.* $s = \psi(s')$, and by induction there exists $s'' \in ESt(\Pi)$ and some $(\sigma, b_1, \ldots, b_k) \in \Sigma \times \mathcal{A}^k$ *s.t.* $s' = \theta(||s''||_{(\sigma, b_1, \ldots, b_k)})$, and thus $s = \psi \circ \theta(||s''||_{(\sigma, b_1, \ldots, b_k)})$

– The case $i = 3$ is similar to the previous one, thus ommitted.

In every subcase, we can conclude that there exists $t' \in ESt(\mathit{\Pi}$ such that $t = \theta(||t'||_{(\sigma,b_1,\ldots,b_k)})$.
$\square$

B Proof of the main result

Theorem 1. *Let $\mathit{\Pi}$ be a well-formed protocol. If $\mathit{\Pi}$ admits an attack, then $\mathit{\Pi}$ admits a well-typed attack.*

Proof. Let $tr = (SId_0, q_0, I_0) \rightarrow \ldots \rightarrow (SId_n, q_n, I_n)$ be an attack and C_0 be the constraint set corresponding to tr's interleaving. Since the reduction procedure is terminating, sound and complete and $\mathit{\Pi}$ admits an attack with this interleaving, then C_0 admits a solution, *i.e.* there exist C, θ, m such that, $C_0 \rightsquigarrow^m_\theta C$ and C is solved.
We will first show that θ is well-typed. We do this by induction on the length m of $C_0 \rightsquigarrow^m_\theta C$:
If $m = 0$, then $\theta = \emptyset$, and thus θ is well-typed.
We suppose that $0 \leq m \leq p,\ C_0 \rightsquigarrow^m_\theta C,\ \theta$ is well-typed.
If $m = p + 1$, then $\exists C', \theta'$ such that:

$$C_0 \rightsquigarrow^p_{\theta'} C' \rightsquigarrow^{R_i}_\psi C,\ i \in \{1, 2, 3, 4, f\} \text{ and } \theta = \psi \circ \theta'$$

and by induction we know that θ' is well-typed.

– If $i \in \{1, f\}$ then $\psi = \emptyset$, thus $\theta = \theta'$ and by induction we can conclude that θ is well-typed.
– If $i = 4$ then $C = \bot$ and this contradicts the fact that C is solved.
– If $i = 2$ then there exist C'', T, u such that $C' = C'' \wedge T \Vdash u$ and $C = C''\psi \wedge T\psi \Vdash u\psi$, with $\psi = \text{mgu}(t, u),\ t \in ESt(T),\ t \neq u$ and u not variable.
Since, θ' is well-typed and $dom(\theta') \cap dom(\psi) = \emptyset$, we only need to show that ψ is well-typed.
Since u is not a variable and unifiable with an ancrypted subterm of T, u is also an encrypted subterm (thus $t, u \in ESt(C')$). And from lemma 4 we know that $\exists t', u' \in ESt(\mathit{\Pi})$ and $\exists (\sigma, b_1, \ldots, b_k), (\sigma', b'_1, \ldots, b'_k) \in \mathit{\Sigma} \times \mathcal{A}^k$ such that $t = \theta'(||t'||_{(\sigma,b_1,\ldots,b_k)})$ and $u = \theta'(||u'||_{(\sigma',b'_1,\ldots,b'_k)})$, and thus $\psi \circ \theta'(||t'||_{(\sigma,b_1,\ldots,b_k)}) = \psi \circ \theta'(||u'||_{(\sigma',b'_1,\ldots,b'_k)})$. Now by well-formed protocols' definition we have that $\delta(t') = \delta(u')$ and thus $t' : \tau \Rightarrow u' : \tau$. Now since we know by induction that θ' is well-typed, we also have that $t : \tau$ and $u : \tau$, and with lemma 1 we can conclude that ψ is well-typed.
– The case $i = 3$ is similar to the previous one, thus ommitted.

In every subcase we can conclude that θ is well typed and C is solved.

To end the proof we now need to show that there exists a well-typed substitution of variables remaining uninstantiated by θ, *i.e.* variables in right hand side of constraints in C. We can note that if these variables remain uninstanciated in C, it means that there exists an attack (well-typed) no matter the actual value taken by these variables. In order

to keep the well-typedness of the attack we just need to instanciate them by an object of the correct type. Since the intruder can create an object of each type, this instanciation is possible.
For every $X \in rhs(C)$ of type τ, we define $\xi(X) = fake(\tau)$ with $fake$ defined recursively as follows:

$$\begin{array}{ll}
fake(\alpha) = \epsilon & fake(\gamma_i) = c_i \\
fake(\kappa^i_j) = \mathfrak{k}^i_j & fake(\nu^i_j) = \mathfrak{n}^i_j \\
fake(\mathtt{pbk}(\alpha)) = \mathtt{pbk}(\epsilon) & fake(\mathtt{pvk}(\alpha)) = \mathtt{pvk}(\epsilon) \\
fake(\mathtt{shk}(\alpha, \alpha)) = \mathtt{shk}(\epsilon, \epsilon) & fake(f(\tau_1, \tau_2)) = f(fake(\tau_1), fake(\tau_2))
\end{array}$$

$\Rightarrow$ $\xi \cup \theta$ is well-typed.

Thus the interleaving of C_0 and $\xi \cup \theta$ describe an attack of the protocol, and since $\xi \cup \theta$, they describe a well-typed attack. We can therefore conclude that if Π admits an attack, then Π admits a well-typed attack. $\Box$

C A valid trace for the Needham-Schroeder protocol

$$\begin{tikzcd}[column sep=large, row sep=large]
\boxed{(SId_0, q_0, I_0)} \arrow[r, "\text{new }(\sigma_1, 1, a, \epsilon)"] & \boxed{\begin{array}{l}(SId_1, q_1, I_1)\\ SId_1 = \{\sigma_1\}\\ q_1(\sigma_1) = (1, a, \epsilon, \emptyset, 1)\\ I_1 = I_0\end{array}} \arrow[r, "\text{send }(\sigma_1, 1)"] & \boxed{\begin{array}{l}(SId_2, q_2, I_2)\\ SId_2 = \{\sigma_1\}\\ q_2(\sigma_1) = (1, a, \epsilon, \emptyset, 2)\\ I_2 = I_1 \cup \{\{a, N_1^1(\sigma_1, a, \epsilon)\}_{\mathtt{pbk}(\epsilon)}\}\end{array}} \arrow[dl, "\text{new}(\sigma_2, 2, a, b)"] \\
& \boxed{\begin{array}{l}(SId_3, q_3, I_3)\\ SId_3 = \{\sigma_1, \sigma_2\}\\ q_3(\sigma_1) = q_2(\sigma_1)\\ q_3(\sigma_2) = (2, a, b, \emptyset, 1)\\ I_3 = I_2\end{array}} \arrow[dl, "\text{receive}(\sigma_2, 1, \{a, N_1^1(\sigma_1, a, \epsilon)\}_{\mathtt{pbk}(b)})"] & \\
\boxed{\begin{array}{l}(SId_4, q_4, I_4)\\ SId_4 = \{\sigma_1, \sigma_2\}\\ q_4(\sigma_1) = q_3(\sigma_1)\\ q_4(\sigma_2) = (2, a, b, [X_1^2(\sigma_2, a, b) \leftarrow N_1^1(\sigma_1, a, \epsilon)], 2)\\ I_4 = I_3\end{array}} \arrow[d, "\text{send}(\sigma_2, 2)"] & & \\
\boxed{\begin{array}{l}(SId_5, q_5, I_5)\\ SId_5 = \{\sigma_1, \sigma_2\}\\ q_5(\sigma_1) = q_4(\sigma_1)\\ q_5(\sigma_2) = (2, a, b, [X_1^2(\sigma_2, a, b) \leftarrow N_1^1(\sigma_1, a, \epsilon))], 3)\\ I_5 = I_4 \cup \{\{N_1^1(\sigma_1, a, \epsilon), N_1^2(\sigma_2, a, b)\}_{\mathtt{pbk}(a)}\}\end{array}} \arrow[d, "\text{receive}(\sigma_1, 2, \{N_1^1(\sigma_1, a, \epsilon), N_1^2(\sigma_2, a, b)\}_{\mathtt{pbk}(a)})"] & & \\
\boxed{\begin{array}{l}(SId_6, q_6, I_6)\\ SId_6 = \{\sigma_1, \sigma_2\}\\ q_6(\sigma_1) = (1, a, \epsilon, [X_1^1(\sigma_1, a, \epsilon) \leftarrow N_1^2(\sigma_2, a, b)], 3)\\ q_6(\sigma_2) = q_5(\sigma_2)\\ I_6 = I_5\end{array}} \arrow[d, "\text{send}(\sigma_1, 3)"] & & \\
\boxed{\begin{array}{l}(SId_7, q_7, I_7)\\ SId_7 = \{\sigma_1, \sigma_2\}\\ q_7(\sigma_1) = (1, a, \epsilon, [X_1^1(\sigma_1, a, \epsilon) \leftarrow N_1^2(\sigma_2, a, b)], 4)\\ q_7(\sigma_2) = q_6(\sigma_2)\\ I_7 = I_6 \cup \{\{N_1^2(\sigma_2, a, b)\}_{\mathtt{pbk}(\epsilon)}\}\end{array}} & &
\end{tikzcd}$$

Fig. 4. A valid trace of Π_{NS}