Synthesizing secure protocols*

Véronique Cortier[1], Bogdan Warinschi[2], and Eugen Zălinescu[1]

[1] Loria UMR 7503 & CNRS & INRIA Lorraine, Project Cassis.
{Veronique.Cortier,Eugen.Zalinescu}@loria.fr
[2] Computer Science Department, University of Bristol. bogdan@cs.bris.ac.uk

Abstract. We propose a general transformation that maps a cryptographic protocol that is secure in an extremely weak sense (essentially in a model where no adversary is present) into a protocol that is secure against a fully active adversary which interacts with an *unbounded* number of protocol sessions, and has absolute control over the network. The transformation works for arbitrary protocols with any number of participants, written with usual cryptographic primitives. Our transformation provably preserves a large class of security properties that contains secrecy and authenticity.
An important byproduct contribution of this paper is a modular protocol development paradigm where designers focus their effort on an extremely simple execution setting – security in more complex settings being ensured by our generic transformation. Conceptually, the transformation is very simple, and has a clean, well motivated design. Each message is tied to the session for which it is intended via digital signatures and on-the-fly generated session identifiers, and prevents replay attacks by encrypting the messages under the recipient's public key.

1 Introduction

Cryptographic protocols are small programs designed to ensure secure communications over an untrusted network. Their security is of crucial importance due to their widespread use in critical systems and in day-to-day life. Unfortunately, designing and analyzing such protocols is a notoriously difficult and error-prone task, largely due to the potentially unbounded behavior of malicious agents.

In this paper we contribute to a popular technique that has been developed to cope with this problem. Under the paradigm that we study, one can start with the design of a simple version of a system intended to work in restricted environments (*i.e.* with restricted adversaries) and then obtain, via a generic transformation, a more robust system intended to work in arbitrary environments.

* The work described in this paper has been supported in part by the European Commission through the IST Programme under Contract IST-2002-507932 ECRYPT, and by the French ACI Satin and the French ACI Jeunes Chercheurs JC9005. The information in this document reflects only the author's views, is provided as is and no guarantee or warranty is given that the information is fit for any particular purpose. The user thereof uses the information at its sole risk and liability.

More specifically, we introduce one such transformation that takes as input a protocol that is secure (in a sense that we discuss below) in a *single* execution of the protocol, with *no adversary* present (not even a passive eavesdropper). The output of the transformation is a protocol that withstands a realistic adversary with absolute control of the communication between an unbounded number of protocol sessions. The details of our transformation are useful to understand how security is transfered from the simple to the more complex setting.

Our transformation. At a high level, the transformation works by first dynamically generating a unique session identifier for the session, bounding messages to sessions by signing their concatenation with the identifier of the session for which they are intended, and finally, hiding the message from the adversary using encryption under the public key of the recipient. More specifically, the transformation is as follows. Consider a protocol with k participants $A_1, \ldots, A_k$ and n exchanges of messages.

$$\begin{aligned} A_{i_1} &\rightarrow A_{j_1} : \ m_1 \\ &\vdots \\ A_{i_n} &\rightarrow A_{j_n} : \ m_n \end{aligned}$$

The transformed protocol starts with a preliminary phase, where each participant A_i broadcasts a fresh nonce N_i to all others participants. The concatenation of the nonces with the identities of the participants forms a session identifier $\mathsf{sessionID} = \langle A_1, A_2, \ldots, A_k, N_1, N_2, \ldots, N_k \rangle$. Note that the adversary may interfere with this preliminary phase and may, for instance, intercept and replace some of the nonces. Such a behavior would however be detected in the next phase. The remainder of the protocol works roughly as the original one except that each message is sent together with a signature on the message concatenated with the session identifier, and the whole construct is encrypted under the recipient's public key:

$$\begin{aligned} A_{i_1} &\rightarrow A_{j_1} : \ \{\!|m_1, [\![m_1, p_1, \mathsf{sessionID}]\!]_{\mathsf{sk}(A_{i_1})}|\!\}_{\mathsf{pk}(A_{j_1})} \\ &\vdots \\ A_{i_n} &\rightarrow A_{j_n} : \ \{\!|m_n, [\![m_n, p_n, \mathsf{sessionID}]\!]_{\mathsf{sk}(A_{i_n})}|\!\}_{\mathsf{pk}(A_{j_n})} \end{aligned}$$

where the p_i's are the current control points in the participant's programs. We write $[\![m]\!]_{\mathsf{sk}(A)}$ for the message m tied with its signature with the signing key of A and $\{\!|m|\!\}_{\mathsf{pk}(A)}$ for the encryption of m under the public encryption key of A.

Security preservation. Intuitively, our transformation ensures that an active adversary cannot tamper with the messages sent during an execution of the (transformed) protocol between honest participants. The transformation also ensures that the adversary cannot learn these messages. In turn, these properties imply that many security statements that hold about a single execution of the protocol, in the absence of an adversary, are inherited by the transformed protocol, even if executed in the presence of an active adversary. Clearly, the transformation does not preserve all imaginable security properties (for example, any anonymity

that the original protocol might enjoy is lost due to the use of public key encryption). We identify a class of logic formulas which if satisfied in single executions of the original protocol are also satisfied by the transformed protocol in the presence of active adversaries. The class that we consider is interesting in that it includes standard formulations for secrecy and authentication (for example injective agreement and several other weaker variants [14]).

Simple protocol design. Our transformation enables more modular and manageable protocol development. One can start by building a protocol with the desirable properties built-in, and bearing in mind that no adversary is actually present. Then, the final protocol is obtained using the transformation that we propose. We remark that designers can easily deal with the case of single session and it is usually the more involved setting (multi-party, many-session) that causes the real problems. Our transformation can be applied to any kind of protocols, with any number of participants (although, the number of participants in each session should not be too large for efficiency).

Simple protocol verification. In standard protocol design, the potentially unbounded behavior of malicious agents makes verification of protocols an extremely difficult task. Even apparently simple security properties like secrecy are undecidable in general [9]. One obvious approach to enable verifiability is to consider restrictions to smaller protocol classes. For example, it can be shown that for finite number of parallel sessions secrecy preservation is co-NP-complete [17]. Most automatic tools are based on this assumption, which is often sufficient to discover new attacks but does not allow in general to prove security properties. It is also possible to ensure verifiability of protocols even for unbounded number of sessions by restricting the form of messages, and/or the ability to generate new nonces, e.g. [9, 3, 4], but only a few such results do not make this unreasonably strong assumption [15, 16].

For the class of protocols obtained via our transformation, security verification is significantly trivialised. Indeed, for a *single, honest* execution, security is in fact closer to correctness, and should be easily carried out automatically.

Related work. The kind of modular design paradigm that we propose is rather pervasive in cryptographic design. For example, Goldreich, Micali, and Wigderson show how to compile arbitrary protocols secure against participants that honestly follow the protocol (but may try to learn information they are not entitled to) into protocols secure against participants that may arbitrarily deviate from the protocol [10]. Bellare, Canetti, and Krawczyk have shown how to transform a protocol that is secure when the communication between parties is authenticated into one that remains secure when this assumption is not met [2]. All of the above transformations have a different goal, apply to protocols that need to satisfy stronger requirements than ours, and are also different in their design.

Our work is inspired by a recent compiler introduced by Katz and Yung [12] which transforms any group key exchange protocol secure against a passive ad-

versary into one secure against an active adversary. Their transformation is, in some sense, simpler since they do not require that the messages in the transformed protocol are encrypted. However, their transformation is also weaker since although it requires that the protocol be secure against passive adversaries, these adversaries still can corrupt parties adaptively (even after the execution has finished). Furthermore, while their transformation is sufficient for the case of group key exchange, it fails to guarantee the transfer of more general security properties. The reason for the failure is that an adversary can obtain a message (e.g. a ciphertext) from a session with only honest participants, and get information about the message (e.g. the underlying plaintext) by replaying it in some other sessions for which he can produce the necessary digital signatures.

Our transformation might be viewed as a way of transforming protocols into fail-stop protocols, introduced by Gong and Syverson [11], where any interference of an attacker is immediately observed and causes the execution to stop. But for fail-stop protocols, it is still necessary to consider the security issues related to the presence of passive adversaries. Here we achieve more since we obtain directly secure protocols. Moreover, a major difference is that we provide formal proof of the security of the resulting protocols while the approach of [11] is rather a methodology for prudent engineering. In particular, there are no proved guarantees on the security of the resulting protocols.

Datta, Derek, Mitchell, and Pavlovic [8] propose a methodology for modular development of protocols where security properties are *added* to a protocol through generic transformations. In contrast, our transformation starts from protocols where the security property is built-in. Abadi, Gonthier, and Fournet give a compiler for programs written in a language with abstractions for secure channels into an implementation that uses cryptography [1] and is similar to ours in the sense that it aims to eliminate cryptographic security analysis in involved settings. However the overall goal is different.

2 Protocols

In this section we give a language for specifying protocols and define their execution in the presence of passive and active adversaries. For simplicity of presentation, we use a model that does not directly capture probabilistic primitives. Nevertheless, our theorems and proofs easily extend to a model that models randomness explicitly (e.g. through the use of labels as in [6]).

2.1 Syntax

We consider protocols specified in a language similar to the one of [6] allowing parties to exchange messages built from identities and randomly generated nonces using asymmetric and symmetric encryption and digital signatures.

Consider the algebraic signature Σ with the following sorts. A sort ID for agent identities, sorts $\mathsf{SigKey}, \mathsf{VerKey}, \mathsf{AsymEKey}, \mathsf{AsymDKey}, \mathsf{SymKey}$ containing keys for signing, verifying, public-key encryption, public-key decryption, and

symmetric encryption algorithms. The algebraic signature also contains sorts Nonce, $\mathsf{Ciphertext}$, $\mathsf{Signature}$, and Pair for nonces, ciphertexts, signatures, and pairs, respectively. The sort Term is a supersort containing, besides all other sorts enumerated above, a sort Int for integers having $\mathbb{Z}$ as the support set. There are eight operations: the four operations $\mathsf{ek}, \mathsf{dk}, \mathsf{sk}, \mathsf{vk}$ are defined on the sort ID and return the asymmetric encryption key, asymmetric decryption key, signing key, and verification key associated to the input identity. The other operations that we consider are pairing, public and symmetric key encryption, and signing. Their ranges and domains are as follows.

- $\langle \_ , \_ \rangle : \mathsf{Term} \times \mathsf{Term} \to \mathsf{Pair}$
- $\{\!|\_|\!\}\_ : \mathsf{AsymEKey} \times \mathsf{Term} \to \mathsf{Ciphertext}$
- $\{\!\{\_\}\!\}\_ : \mathsf{SymKey} \times \mathsf{Term} \to \mathsf{Ciphertext}$
- $[\![\_]\!]\_ : \mathsf{SigKey} \times \mathsf{Term} \to \mathsf{Signature}$

Let $\mathsf{X.a}, \mathsf{X.n}, \mathsf{X.k}, \mathsf{X.c}, \mathsf{X.s}, \mathsf{X.t}$ be sets of variables of sort agent, nonce, (symmetric) key, ciphertext, signature and term respectively, and $\mathsf{X} = \mathsf{X.a} \cup \mathsf{X.n} \cup \mathsf{X.k} \cup \mathsf{X.c} \cup \mathsf{X.s} \cup \mathsf{X.t}$. Protocols are specified using the terms in $T_\Sigma(\mathsf{X})$ of the free algebra generated by X over the signature Σ. We suppose that pairing is left associative and write $\langle m_1, m_2, \ldots, m_l \rangle$ for $\langle\langle\langle m_1, m_2\rangle, m_3\rangle \ldots, m_l\rangle$. When unambiguous, we may omit the brackets.

Throughout the paper we fix a constant $k \in \mathbb{N}$ that represents the number of protocol participants and we write $[k]$ for the set $\{1, 2, \ldots, k\}$. Furthermore, without loss of generality, we fix the set of agent variables to be $\mathsf{X.a} = \{ A_1, A_2, \ldots, A_k \}$, and partition the set of nonce (and key) variables, according to the party that generates them. Formally:

$$\mathsf{X.n} = \cup_{A \in \mathsf{X.a}} \mathsf{X_n}(A) \text{ where } \mathsf{X_n}(A) = \{N_A^j \mid j \in \mathbb{N}\}$$
$$\mathsf{X.k} = \cup_{A \in \mathsf{X.a}} \mathsf{X_k}(A) \text{ where } \mathsf{X_k}(A) = \{K_A^j \mid j \in \mathbb{N}\}$$

This partition avoids to have to specify later which of the nonces (symmetric keys) are generated by the party executing the protocol, or are expected to be received from other parties.

Roles and protocols. The messages that are sent by participants are specified using terms in$\mathsf{T}_\Sigma(\mathsf{X})$. The individual behavior of each protocol participant is defined by a *role* describing a sequence of message reception/transmission which we call *steps* or *rules*. A k-party protocol consists of k such roles together with an association that maps each step of a role that expects some message m to the step of the role where the message m is produced. Notice that this association essentially defines how the execution of a protocol should proceed in the absence of an adversary.

Definition 1 (Roles and protocols). *The set of* roles *is defined by* $\mathsf{Roles} = \big((\{\,\mathsf{init}\,\} \cup \mathsf{T}_\Sigma(\mathsf{X})) \times (\mathsf{T}_\Sigma(\mathsf{X}) \cup \{\,\mathsf{stop}\,\})\big)^*$. *A* k-party protocol *is a pair* $\Pi = (\mathcal{R}, \mathcal{S})$ *where* $\mathcal{R}$ *is a mapping* $\mathcal{R} : [k] \to \mathsf{Roles}$ *that maps* $i \in [k]$ *to the role executed by the* i*'th protocol participant and* $\mathcal{S} : [k] \times \mathbb{Z} \hookrightarrow [k] \times \mathbb{Z}$ *is a partial mapping that returns for each role/control-point pair* (r, p)*, the role/control-point pair* $(r', p') = \mathcal{S}(r, p)$ *which emits the message to be processed by role* r *at step* p*.*

We assume that a protocol specification is such that the r'th role of the protocol $\mathcal{R}(r) = ((\mathsf{rcv}_r^1, \mathsf{snt}_r^1), (\mathsf{rcv}_r^2, \mathsf{snt}_r^2), \ldots)$, is executed by player A_r. Informally, the above definition says that at step p, A_r expects to receive a message of the form specified by rcv_r^p and returns the message snt_r^p. It is important to notice that the terms rcv_r^p and snt_r^p are not actual messages but specify how the message that is received and the message that is output should look like. The messages init and stop are used to initiate and signal successful termination of role executions. We note that for technical reasons we sometimes use negative control points (with negatively indexed role rules).

Example 1. The Needham-Schroeder-Lowe protocol [13]

$$\begin{aligned} A \to B:\ & \{\!|N_a, A|\!\}_{\mathsf{ek}(B)} \\ B \to A:\ & \{\!|N_a, N_b, B|\!\}_{\mathsf{ek}(A)} \\ A \to B:\ & \{\!|N_b|\!\}_{\mathsf{ek}(B)} \end{aligned}$$

is specified as follows: there are two roles $\mathcal{R}(1)$ and $\mathcal{R}(2)$ corresponding to the sender's role and the receiver's role.

$$\begin{aligned} \mathcal{R}(1): & \big(\mathsf{init},\ \{\!|N^1_{A_1}, A_1|\!\}_{\mathsf{ek}(A_2)}\big) && \mathcal{S}(1,1) = (0,0) \\ & \big(\{\!|N^1_{A_1}, N^1_{A_2}, A_2|\!\}_{\mathsf{ek}(A_1)},\ \{\!|N^1_{A_2}|\!\}_{\mathsf{ek}(A_2)}\big) && \mathcal{S}(1,2) = (2,1) \\ \mathcal{R}(2): & \big(\{\!|N^1_{A_1}, A_1|\!\}_{\mathsf{ek}(A_2)},\ \{\!|N^1_{A_1}, N^1_{A_2}, A_2|\!\}_{\mathsf{ek}(A_1)}\big) && \mathcal{S}(2,1) = (1,1) \\ & \big(\{\!|N^1_{A_2}|\!\}_{\mathsf{ek}(A_2)},\ \mathsf{stop}\big) && \mathcal{S}(2,2) = (1,2) \end{aligned}$$

Executable protocols. Clearly, not all protocols written using the syntax above are meaningful. We only consider the class of *executable protocols*, *i.e.* protocols for which each role can be implemented in an executable program, using only the local knowledge of the corresponding agent. This requires in particular that any sent message (corresponding to some snt_r^p) is always deducible from the previously received messages (corresponding to $\mathsf{rcv}_r^1, \ldots, \mathsf{rcv}_r^p$). Also we demand that $\mathcal{S}$ is consistent. In particular, this means that for a fixed role r, $\mathcal{S}(r,p)$ is defined on exactly $|\mathcal{R}(r)|$ consecutive integers, where $|S|$ denotes the cardinality of the set S.

2.2 Formal Execution Model

We start with the description of the execution model of the protocol in the presence of an active attacker. The model that we consider is rather standard. The parties in the system execute a (potentially unbounded) number of protocol sessions with each other. The communication is under the complete control of the adversary who can intercept, drop, or modify the messages on the network.

The messages transmitted between parties are terms of the algebra T^f freely generated over the signature Σ by an arbitrary fixed set of identities $\mathsf{T}^f_{\mathsf{ID}}$ together with the sets for types SymKey and Nonce defined by:

$$\begin{aligned} \mathsf{T}^f_{\mathsf{SymKey}} &= \{\mathsf{k}^{a,j,s} \mid a \in \mathsf{T}^f_{\mathsf{ID}}, j \in \mathbb{N}, s \in \mathbb{N}\} \cup \{\mathsf{k}^j \mid j \in \mathbb{N}\} \\ \mathsf{T}^f_{\mathsf{Nonce}} &= \{\mathsf{n}^{a,j,s} \mid a \in \mathsf{T}^f_{\mathsf{ID}}, j \in \mathbb{N}, s \in \mathbb{N}\} \cup \{\mathsf{n}^j \mid j \in \mathbb{N}\} \end{aligned}$$

Informally, one should think of the constant $k^{a,j,s}$ (respectively $n^{a,j,s}$) as the j'th key (respectively nonce) generated by party a in session s. Constants k^j and n^j represent keys and nonces produced by the adversary.

To each protocol we associate the set of its valid execution traces. First we clarify what execution traces are and then present the association.

A *global state* of an execution is given by a triple (SId, f, H). Here, SId is the set of role session ids currently executed by protocol participants, f is a global assignment function that keeps track of the local state of each existing session and H is the set of messages that have been sent on the network so far.

More precisely, each role session has an associated unique session identifier $s \in \mathbb{N}$. However, by language abuse (and overloading the notion), we say that a *session id* is a tuple of the form $(s, r, (a_1, a_2, \ldots, a_k))$, where s is the unique identifier for the session, r is the index of the role that is executed in the session and $a_1, a_2, \ldots, a_k \in \mathsf{T}^f_{\mathsf{ID}}$ are the identities of the parties that are involved in the session. We write SID for the set $(\mathbb{N} \times \mathbb{N} \times (\mathsf{T}^f_{\mathsf{ID}})^k)$ of all session ids.

Mathematically, the global assignment f is a function $f : \mathsf{SId} \to ([\mathsf{X} \hookrightarrow \mathsf{T}^f] \times \mathbb{N} \times \mathbb{Z})$, where $\mathsf{SId} \subseteq \mathsf{SID}$ represents the session ids initialized in the execution. For each such session id $\mathsf{sid} \in \mathsf{SId}$, $f(\mathsf{sid}) = (\sigma, r, p)$ returns its local state. Here, r is the same role index as in sid, the function σ is a substitution, that is a partial instantiation of the variables of the role $\mathcal{R}(r)$ and $p \in \mathbb{Z}$ is the control point of the program. We sometimes write $X\sigma$ for $\sigma(X)$. We denote by GA the set $[\mathsf{SID} \hookrightarrow ([\mathsf{X} \hookrightarrow \mathsf{T}^f] \times \mathbb{N} \times \mathbb{Z})]$ of all possible global assignments.

Finally, the messages that may be sent on the network can be essentially any element of T^f, so we write Msgs for the set 2^{T^f} (where 2^S is the power set of S).

An *execution trace* is a sequence

$$(S_0, f_0, H_0) \xrightarrow{\alpha_1} (S_1, f_1, H_1) \xrightarrow{\alpha_2} \ldots \xrightarrow{\alpha_n} (S_n, f_n, H_n)$$

such that for each $0 \leq i \leq n$, $(S_i, f_i, H_i) \in (2^{\mathsf{SID}} \times \mathsf{GA} \times \mathsf{Msgs})$ and α_i is one of the *actions* **corrupt**, **new**, and **send** with appropriate parameters that we clarify below. This corresponds to the intuition that transitions between two global states are caused by actions of the adversary who can corrupt users, initiate new sessions of the protocol between users that he chooses, and send messages to existing sessions.

For a k-party protocol Π, the transitions between global states are as follows:

- The adversary corrupts agents: $(\mathsf{SId}, f, H) \xrightarrow{\mathbf{corrupt}(a_1,\ldots,a_l)} (\mathsf{SId}, f, H')$ where $a_1, \ldots, a_l \in \mathsf{T}^f_{\mathsf{ID}}$ and $H' = \cup_{1 \leq j \leq l} \mathbf{kn}(a_j) \cup H$. Here, $\mathbf{kn}(a_j)$ denotes the knowledge of a_j: if A is a variable, or a constant of sort agent, we define its knowledge by $\mathbf{kn}(A) = \{\mathsf{dk}(A), \mathsf{sk}(A)\}$ *i.e.* an agent knows its secret decryption and signing key. The adversary corrupts parties by outputting a set of identities. In return, the adversary receives the secret keys corresponding to the identities. In this paper we are only concerned with the case of static corruption so this transition only occurs at the beginning of an execution trace.

– The adversary initiates new sessions: $(\mathsf{SId},f,H) \xrightarrow{\mathbf{new}(r,a_1,\ldots,a_k)} (\mathsf{SId}',f',H)$ where $1 \leq r \leq k$, $a_1,\ldots,a_k \in \mathsf{T}^f_{\mathsf{ID}}$, and f' and SId' are defined as follows. Let $s = |\mathsf{SId}| + 1$, be the session identifier of the new session. Then $\mathsf{SId}' = \mathsf{SId} \cup \{(s, r, (a_1,\ldots,a_k))\}$ and the function f' is defined by:
  - $f'(\mathsf{sid}) = f(\mathsf{sid})$ for every $\mathsf{sid} \in \mathsf{SId}$.
  - $f'(s, r, (a_1,\ldots,a_k)) = (\sigma, r, p_0)$ where p_0 is the initial control point of role r and σ is a partial function $\sigma : \mathsf{X} \hookrightarrow \mathsf{T}^f$ defined by $\sigma(A_j) = a_j$ for $1 \leq j \leq k$, $\sigma(N^j_{A_r}) = \mathsf{n}^{a_r,j,s}$ for $j \in \mathbb{N}$ and $\sigma(K^j_{A_r}) = \mathsf{k}^{a_r,j,s}$ for $j \in \mathbb{N}$. We recall that the principal that executes role $\mathcal{R}(r)$ is represented by variable A_r thus, in that role, every variable of the form $X^j_{A_r}$ represents a nonce or a symmetric key generated by A_r.
– The adversary sends messages to sessions: $(\mathsf{SId},f,H) \xrightarrow{\mathbf{send}(\mathsf{sid},m)} (\mathsf{SId}, f', H')$ where $\mathsf{sid} \in \mathsf{SId}$ and $m \in \mathsf{T}^f$. H' and f' are defined as follows. We define $f'(\mathsf{sid}') = f(\mathsf{sid}')$ for every $\mathsf{sid}' \in \mathsf{SId} \setminus \{\mathsf{sid}\}$. Let $f(\mathsf{sid}) = (\sigma, r, p)$ for some σ, r and p, and let $\mathcal{R}(r) = \big((\mathsf{rcv}^1_r, \mathsf{snt}^1_r), \ldots, (\mathsf{rcv}^{k_r}_r, \mathsf{snt}^{k_r}_r)\big)$ be the role executed in this session. There are two cases:
  - Either there exists a substitution σ' such that $m = \mathsf{rcv}^p_r\sigma'$ and σ' extends σ, that is, $X\sigma' = X\sigma$ whenever σ is defined on X. Then $f'(\mathsf{sid}) = (\sigma', r, p+1)$ and $H' = H \cup \{\mathsf{snt}^p_r\sigma'\}$. We say that m is *accepted*.
  - Or we let $f'(\mathsf{sid}) = f(\mathsf{sid})$ and $H' = H$ (the state remains unchanged).

As usual, we are only interested in *valid* execution traces – those traces where the adversary only sends messages that he can compute out of his knowledge and the messages it had seen on the network. The adversary can derive new information using the relation $\vdash$. Intuitively, $S \vdash m$ means that the adversary is able to compute the message m from the set of messages S; the adversarial abilities are captured by the definition in Figure 1. The only rule that is perhaps less standard is the last one. It essentially states that out of a signature an adversary could compute the message that is signed, which is theoretically possible for any secure digital signature scheme.

The set of valid execution traces is described by the following definition.

Definition 2 (Valid execution traces). *An execution trace* $(\mathsf{SId}_0, f_0, H_0) \rightarrow \ldots \rightarrow (\mathsf{SId}_n, f_n, H_n)$ *is* valid *if*

– $H_0 = \mathsf{SId}_0 = \emptyset$, $(\mathsf{SId}_0, f_0, H_0) \rightarrow (\mathsf{SId}_1, f_1, H_1)$ *for one of the three transitions described above and for every* $1 \leq i \leq n$, $(\mathsf{SId}_i, f_i, H_i) \rightarrow (\mathsf{SId}_{i+1}, f_{i+1}, H_{i+1})$ *for one of the last two transitions described above;*
– *moreover, the messages sent by the adversary can be computed using* $\vdash$, *i.e. whenever* $(\mathsf{SId}_i, f_i, H_i) \xrightarrow{\mathbf{send}(\mathsf{sid},m)} (\mathsf{SId}_{i+1}, f_{i+1}, H_{i+1})$ *then* $H_i \vdash m$.

Given a protocol Π, *we write* $\mathsf{Exec}(\Pi)$ *for the set of valid execution traces of* Π.

Example 2. Playing with the Needham-Schroeder-Lowe protocol described in Example 1, an adversary can corrupt an agent a_3, start a new session for the

$\dfrac{}{S \vdash m}\, m \in S$	$\dfrac{}{S \vdash a, \mathsf{ek}(a), \mathsf{vk}(a), \mathsf{k}^j, \mathsf{n}^j}\, j \in \mathbb{N}$	Initial knowledge
$\dfrac{S \vdash m_1 \quad S \vdash m_2}{S \vdash \langle m_1\, , m_2 \rangle}$	$\dfrac{S \vdash \langle m_1\, , m_2 \rangle}{S \vdash m_i}\, i \in \{1,2\}$	Pairing and unpairing
$\dfrac{S \vdash k \quad S \vdash m}{S \vdash \{\!\!\{m\}\!\!\}_k}$	$\dfrac{S \vdash \{\!\!\{m\}\!\!\}_k \quad S \vdash k}{S \vdash m}$	Sym. encryption/decryption
$\dfrac{S \vdash m}{S \vdash \{\!\!\{m\}\!\!\}_{\mathsf{ek}(a)}}$	$\dfrac{S \vdash \{\!\!\{m\}\!\!\}_{\mathsf{ek}(a)} \quad S \vdash \mathsf{dk}(a)}{S \vdash m}$	Asym. encryption/decryption
$\dfrac{S \vdash \mathsf{sk}(a) \quad S \vdash m}{S \vdash [\![m]\!]_{\mathsf{sk}(a)}}$	$\dfrac{S \vdash [\![m]\!]_{\mathsf{sk}(a)}}{S \vdash m}$	Signature

Fig. 1. Deduction rules for the formal adversary. In the above, $a \in \mathsf{T}^f_{\mathsf{ID}}$.

second role with players a_1, a_2 and send the message $\{\!\!\{\mathsf{n}(a_3,1,1), a_1\}\!\!\}_{\mathsf{ek}(a_2)}$ to the player of the second role. The corresponding valid trace execution is:

$$(\emptyset, f_1, \emptyset) \xrightarrow{\mathbf{corrupt}(a_3)} (\emptyset, f_1, \mathbf{kn}(a_3)) \xrightarrow{\mathbf{new}(2,a_1,a_2)} (\{\mathsf{sid}_1\}, f_2, \mathbf{kn}(a_3))$$
$$\xrightarrow{\mathbf{send}(\mathsf{sid}_1, \{\!\!\{n_3,a_1\}\!\!\}_{\mathsf{ek}(a_2)})} \left(\{\mathsf{sid}_1\}, f_3, \mathbf{kn}(a_3) \cup \{\{\!\!\{n_3, n_2, a_2\}\!\!\}_{\mathsf{ek}(a_1)}\}\right),$$

where $\mathsf{sid}_1 = (1, 2, (a_1, a_2))$, $n_2 = \mathsf{n}(a_2, 1, 1)$, $n_3 = \mathsf{n}(a_3, 1, 1)$, and f_2, f_3 are defined as follows: $f_2(\mathsf{sid}_1) = (\sigma_1, 2, 1)$, $f_3(\mathsf{sid}_1) = (\sigma_2, 2, 2)$ where $\sigma_1(A_1) = a_1$, $\sigma_1(A_2) = a_2$, $\sigma_1(N^1_{A_2}) = n_2$, and σ_2 extends σ_1 by $\sigma_2(N^1_{A_1}) = n_3$.

Given an arbitrary trace $\mathsf{tr} = (\mathsf{SId}_0, f_0, H_0) \xrightarrow{\alpha_1} \ldots \xrightarrow{\alpha_n} (\mathsf{SId}_n, f_n, H_n)$ with $n \in \mathbb{N}$, we define the set of *corrupted agents* of a trace tr by $\mathsf{CA}(\mathsf{tr}) = \{a_1, \ldots, a_l\}$ if $\alpha_1 = \mathbf{corrupt}(a_1, \ldots, a_l)$ and $\mathsf{CA}(\mathsf{tr}) = \emptyset$ otherwise. The set $\mathsf{SId}^{\mathsf{h}}(\mathsf{tr})$ of honest session identifiers is the set of session identifiers that correspond to sessions between non-corrupt agents:

$$\mathsf{SId}^{\mathsf{h}}(\mathsf{tr}) = \{\mathsf{sid} \in \mathsf{SId}_n \mid \mathsf{sid} = (s, r, (a_1, \ldots, a_k)), \mathsf{CA}(\mathsf{tr}) \cap \{a_1, \ldots, a_k\} = \emptyset\}.$$

Also, for a trace tr we denote by $\mathsf{I}(\mathsf{tr})$ the set of indexes i of the transitions and global states of tr. For example the above trace has $\mathsf{I}(\mathsf{tr}) = \{0, 1, \ldots, n\}$. If sid is a session id then we denote by $\mathsf{Ag}(\mathsf{sid})$ the set of agents involved in this session, that is $\mathsf{Ag}(\mathsf{sid}) = \{a_1, \ldots, a_k\}$ when $\mathsf{sid} = (\cdot, \cdot, (a_1, \ldots, a_k))$.

3 Security properties

We use a simple logic introduced in [5] to express security properties for protocols specified in the language given in the previous section. We recall this logic, define

its semantics and provide several examples of security properties that can be expressed within.

3.1 A logic for security properties

The formulas of logic capture trace properties and, in particular, they allow quantification over the local states of agents. We define the set of local states of a trace $\mathsf{tr} = (\mathsf{SId}_i, f_i, H_i)_{1 \leq i \leq n}$ for role r at step p by $\mathcal{LS}_{r,p}(\mathsf{tr}) \triangleq \{(\sigma, r, p) \mid \exists i \in [n], \exists \mathsf{sid} \in \mathsf{SId}_i, \text{ s.t. } f_i(\mathsf{sid}) = (\sigma, r, p)\}$. Note that the cardinality of this set equals the number of agents that are playing role r and that have reached control point p.

We assume an infinite set XSub of meta-variables for substitutions. The logic contains tests between terms where variables are substituted by variable substitutions. More formally, let $\mathsf{T}_{\mathsf{Sub}}$ be the algebra defined by:

$$\mathsf{T}_{\mathsf{Sub}} ::= \varsigma(X) \mid g(\mathsf{T}_{\mathsf{Sub}}) \mid h(\mathsf{T}_{\mathsf{Sub}}, \mathsf{T}_{\mathsf{Sub}})$$

where $\varsigma \in \mathsf{XSub}$, $X \in \mathsf{X}$, and $g, h \in \Sigma$ of arity 1 and 2 respectively.

Besides standard propositional connectors, the logic has a predicate to specify honest agents, equality and inequality tests between terms, and existential and universal quantifiers over the local states of agents.

Definition 3. *The formulas of the logic $\mathcal{L}$ are defined by induction as follows:*

$$\begin{aligned}\phi(\mathsf{tr}) ::= {} & \mathsf{NC}(\mathsf{tr}, \varsigma(A)) \mid t_1 = t_2 \mid \neg\phi(\mathsf{tr}) \mid \phi(\mathsf{tr}) \wedge \phi(\mathsf{tr}) \mid \phi(\mathsf{tr}) \vee \phi(\mathsf{tr}) \\ & \mid \forall \mathcal{LS}_{r,p}(\mathsf{tr}).\varsigma\ \phi(\mathsf{tr}) \mid \exists \mathcal{LS}_{r,p}(\mathsf{tr}).\varsigma\ \phi(\mathsf{tr}) \mid \exists! \mathcal{LS}_{r,p}(\mathsf{tr}).\varsigma\ \phi(\mathsf{tr})\end{aligned}$$

where tr is a parameter of the formula, $A \in \mathsf{X}.a$, $\varsigma \in \mathsf{XSub}$, $t_1, t_2 \in \mathsf{T}_{\mathsf{Sub}}$ and $r, p \in \mathbb{N}$. As usual, we may use $\phi_1 \Rightarrow \phi_2$ as a shortcut for $\neg\phi_1 \vee \phi_2$.

Here the predicate $\mathsf{NC}(\mathsf{tr}, \varsigma(A))$ is used to specify non corrupted agents. The quantifications $\forall \mathcal{LS}_{r,p}(\mathsf{tr}).\varsigma$ and $\exists \mathcal{LS}_{r,p}(\mathsf{tr}).\varsigma$ are over local states of agent r at step p in trace tr, and they bound the variable substitution ς. The semantics of our logic is defined for *closed* formula as follows: standard propositional connectors and negation are interpreted as usual. Equality is syntactic equality. The interpretation of quantifiers and the predicate NC is shown in Figure 2.

A *security property* ϕ should be seen in this paper as an abstraction of the form $\phi \triangleq \lambda\mathsf{tr}.\phi(\mathsf{tr})$, where the tr parameter is used only to define the semantics of such formulas. By abuse of notation we therefore ignore this parameter and write $\phi \in \mathcal{L}$ for a security property. Informally, a protocol Π satisfies ϕ if $\phi(\mathsf{tr})$ is true for all traces of Π. Formally:

Definition 4 (Satisfiability). *Let Π be a protocol and $\phi \in \mathcal{L}$ be a security property. We say that Π* satisfies *the security property ϕ, and write $\Pi \models \phi$ if for any trace $\mathsf{tr} \in \mathsf{Exec}(\Pi)$, $[\![\phi(\mathsf{tr})]\!] = 1$.*

$$[\![\mathsf{NC}(\mathsf{tr},a)]\!] = \begin{cases} 1 & \text{if } a \text{ does not appear in a corrupt action,} \\ & \textit{i.e.} \text{ if } e_1 \xrightarrow{\mathbf{corrupt}(a_1,\ldots,a_l)} e_2,\ \mathsf{intr} = (e_1,e_2,\ldots,e_n), \\ & \text{we have } a \neq a_i, \forall 1 \leq i \leq l, \\ 0 & \text{otherwise} \end{cases}$$

$$[\![\forall \mathcal{LS}_{r,p}(\mathsf{tr}).\varsigma\ \phi(\mathsf{tr})]\!] = \begin{cases} 1 & \text{if } \forall(\sigma,r,p) \in \mathcal{LS}_{r,p}(\mathsf{tr}), \text{ we have } [\![\phi(\mathsf{tr})[\%_\varsigma]]\!] = 1, \\ 0 & \text{otherwise.} \end{cases}$$

$$[\![\exists \mathcal{LS}_{r,p}(\mathsf{tr}).\varsigma\ \phi(\mathsf{tr})]\!] = \begin{cases} 1 & \text{if } \exists(\sigma,r,p) \in \mathcal{LS}_{r,p}(\mathsf{tr}), \text{ s.t. } [\![\phi(\mathsf{tr})[\%_\varsigma]]\!] = 1, \\ 0 & \text{otherwise.} \end{cases}$$

$$[\![\exists! \mathcal{LS}_{r,p}(\mathsf{tr}).\varsigma\ \phi(\mathsf{tr})]\!] = \begin{cases} 1 & \text{if } \exists!\, \mathsf{sid} \in \mathsf{SId}(\mathsf{tr}), \exists i \in \mathsf{I}(\mathsf{tr}) \text{ s.t.} \\ & f_i(\mathsf{sid}) = (\sigma,r,p) \text{ and } [\![\phi(\mathsf{tr})[\%_\varsigma]]\!] = 1, \\ 0 & \text{otherwise.} \end{cases}$$

Fig. 2. Interpretation of formulas in $\mathcal{L}$.

3.2 Examples of security properties

In this section we show how to specify secrecy and several variants of authentication, including those from Lowe's hierarchy [14], in the given security logic.

A secrecy property. Let Π be a k-party executable protocol. To specify our secrecy property we use a standard encoding. Namely, we add a role to the protocol, $\mathcal{R}(k+1) = (Y, \mathsf{stop})$, where Y is a new variable of sort Term. It can be seen as some sort of witness as it does nothing but waits for receiving a piece of public data.

Informally, the definition of the secrecy property ϕ_s states that, for any local state of an agent playing role r in which a nonce (or a key) X was created in an honest session, a witness (i.e. an agent playing role $k+1$) cannot gain any knowledge on X. Formally, the property is specified by the following formula:

$$\phi_s(\mathsf{tr}) = \forall \mathcal{LS}_{r,1}(\mathsf{tr}).\varsigma \Big(\bigwedge_{l \in [k]} NC(\mathsf{tr}, \varsigma(A_l)) \Rightarrow \forall \mathcal{LS}_{k+1,2}(\mathsf{tr}).\varsigma' \left(\varsigma(X) \neq \varsigma'(Y)\right) \Big)$$

Intuitively, the formula states that for all local states of an agent executing role r (in session ς) and being at his initial control point (i.e. 1), if only honest agents are playing in ς then for all local states of an agent executing role $k+1$ (in session ς') and being at his final control point (i.e. 2), the value of the secret (i.e. X) in ς is different from the value of the received message (i.e. Y) in ς'.

As a side remark, notice that it is possible to also model the secrecy of a data X received during an honest session: we would simply specify the control point p (instead of 1) at which the date is received by the role r. Moreover, in both cases (that is, X created or received) the formula is always true for honest, single session traces. It will follow that our transformation preserves secrecy of all nonces or keys used in sessions that involve only honest parties.

Authentication properties. We show how to use the logic defined above to specify the injective agreement [14] between two parties A and B. Informally, this

property states that whenever an A completes a run of the protocol, apparently with B, then there is unique run of B apparently with A such that two agents agree on the values of some fixed variables, provided that A and B are honest. As usual nothing is guaranteed in sessions involving corrupted agents.

Let p_1 be the length of A's role and p_2 be the control point at which B should have received all data items from A. Then, the above intuition is captured by the following formula:

$$\phi_a(\mathsf{tr}) \triangleq \forall \mathcal{LS}_{1,p_1}(\mathsf{tr}).\varsigma \ \Big(\mathsf{NC}(\mathsf{tr}, A\varsigma) \wedge \mathsf{NC}(\mathsf{tr}, B\varsigma) \Rightarrow \exists! \mathcal{LS}_{2,p_2}(\mathsf{tr}).\varsigma' \ \big((A\varsigma = A\varsigma') \wedge (B\varsigma = B\varsigma') \wedge \textstyle\bigwedge_{1\leq i\leq n}(X_i\varsigma = X_i\varsigma')\big)\Big)$$

Intuitively, the formula states that for all local states of an agent having finished executing the first role (in session ς), if the two agents (executing the two roles in ς) are honest in ς then there exists an unique local state of an agent having finished executing the second role (in session ς') and such that the agents agree on their identitites and on the values of the variables that are common to the two roles.

It is easy to modify this formula in order to obtain formulas corresponding to the other variants of authentication from Lowe's hierarchy.

4 Transformation of protocols

The core idea of the transformation is to have parties agree on some common, dynamically generated, session identifier s, and then transmit the encryption of a message m of the original protocol accompanied by a signature on $m||s$.

The modification of the source protocol is performed in two steps. We first introduce an initialization phase, where each agent generates a fresh nonce which is distributed to all other participants. The idea is that the concatenation of all these nonces and all the identities involved in the session plays the role of a unique session identifier. To avoid underspecification of the resulting protocol we fix a particular way in which the nonces are distributed. First, each agent generates a fresh nonce and then sends the nonces he received so far together with his nonce to the next agent. That is, in Alice-Bob notation, $A_i \rightarrow A_{i+1} : \ N_{A_1}, ..., N_{A_i}$, for all i in the sequence $1, \ldots, k-1$. Then, once the last agent received all nonces, each agent forwards the concatenation of all nonces to its predecesor. That is, $A_i \rightarrow A_{i-1} : \ N_{A_1}, \ldots, N_{A_k}$, for all i in the sequence $k, \ldots, 2$. In this way, at the end of this first phase all agents know each other's nonces.

We remark that the precise order in which participants send these nonces does not really matter, and we do not require that these nonces be authenticated in some way. In principle an active adversary is allowed to forward, block or modify the messages sent during the initialization phase, but behaviours that deviate from the intended execution of the protocol are detected in the next phase.

In the second phase of the transformed protocol, the execution proceeds as prescribed by Π with the difference that to each message m that needs to be sent, the sending parties also attaches a signature $[\![m, p, \mathsf{nonces}]\!]_{\mathsf{sk}'(a)}$ and encrypts the

whole construct with the intended receiver public key. p is the current control point and nonces is the concatenation of the nonces received during the first phase with the identities of the participants involved in the protocol.

Intuitively, the adversary cannot impersonate users in honest sessions (since in this case it would need to produce digital signatures on their behalf), and cannot learn secrets by replaying messages from one session to another (since messages are encrypted, and any blindly replayed message would be rejected due to un-matching session identifiers). The control-point p plays within a session the same role as nonces between sessions: messages not matching the correct control point are rejected.

To avoid confusion and unintended interactions between the signatures and the encryptions produced by the compiler and those used in the normal execution of the protocol, the former use fresh signatures and public keys. Formally, we extend the signature Σ with four new function symbols sk', vk', ek' and dk' which have exactly the same functionality (that is the same sort and similar deduction rules) with sk, vk, ek and dk respectively.

This formalizes the assumption that in the transformed version of Π each agent a has associated two pairs of verification/signing keys $((\mathsf{vk}(a), \mathsf{sk}(a))$ and $(\mathsf{vk}'(a), \mathsf{sk}'(a)))$ and two pairs of encryption/decryption keys $((\mathsf{ek}(a), \mathsf{dk}(a))$ and $(\mathsf{ek}'(a), \mathsf{dk}'(a)))$ and that these new pairs of keys were correctly distributed previously to any execution of the protocol. We assume that source protocols are constructed over Σ only.

Definition 5 (Transformed protocol). *Let $\Pi = (\mathcal{R}, \mathcal{S})$ be a k-party executable protocol such that the nonce variables $N^0_{A_k}$ do not appear in $\mathcal{R}$ (which can be ensured by renaming the nonce variables of Π) and all the initial control points are set to 1 (which can be ensured by rewriting the function $\mathcal{S}$).*

The transformed protocol $\widetilde{\Pi} = (\widetilde{\mathcal{R}}, \widetilde{\mathcal{S}})$ *is defined as follows:* $\widetilde{\mathcal{R}}(r) = \mathcal{R}^{\mathsf{init}}(r) \cdot \mathcal{R}'(r)$ *and* $\widetilde{\mathcal{S}} = \mathcal{S}^{\mathsf{init}} \cup \mathcal{S}$ *where* $\cdot$ *denotes the concatenation of sequences and* $\mathcal{R}^{\mathsf{init}}$, $\mathcal{R}'$ *and* $\mathcal{S}^{\mathsf{init}}$ *are defined as follows:*

$$\begin{array}{l}\mathcal{R}^{\mathsf{init}}(r) = \big((\mathsf{nonces}_{r-1}, \mathsf{nonces}_r), (\mathsf{nonces}_k, \mathsf{nonces}_k)\big), \quad \forall 1 \le r < k,\\ \mathcal{S}^{\mathsf{init}}(r,-1) = (r-1,-1), \quad \mathcal{S}^{\mathsf{init}}(r,0) = (r+1,0), \quad \forall 1 \le r < k,\\ \mathcal{R}^{\mathsf{init}}(k) = \big((\mathsf{nonces}_{k-1}, \mathsf{nonces}_k)\big) \qquad \mathcal{S}^{\mathsf{init}}(k,0) = (k-1,-1)\end{array}$$

with $\mathsf{nonces}_0 = \mathsf{init}$ *and* $\mathsf{nonces}_j = \langle N^0_{A_1}, N^0_{A_2}, \ldots, N^0_{A_j}\rangle$ *for* $1 \le j \le k$.
Let $\mathcal{R}(r) = \big((\mathsf{rcv}^p_r, \mathsf{snt}^p_r)\big)_{p\in[k_r]}$. *Then* $\mathcal{R}'(r) = \big((\widetilde{\mathsf{rcv}}^p_r, \widetilde{\mathsf{snt}}^p_r)\big)_{p\in[k_r]}$ *such that if* $\mathsf{rcv}^p_r = \mathsf{init}$ *then* $\widetilde{\mathsf{rcv}}^p_r = \mathsf{fake}$, *if* $\mathsf{snt}^p_r = \mathsf{stop}$ *then* $\widetilde{\mathsf{snt}}^p_r = \mathsf{stop}$ *and otherwise*

$$\begin{array}{l}\widetilde{\mathsf{rcv}}^p_r = \{\![\mathsf{rcv}^p_r, [\![\mathsf{rcv}^p_r, p', \mathsf{nonces}]\!]_{\mathsf{sk}'(A_{r'})}]\!\}_{\mathsf{ek}'(A_r)},\\ \widetilde{\mathsf{snt}}^p_r = \{\![\mathsf{snt}^p_r, [\![\mathsf{snt}^p_r, p, \mathsf{nonces}]\!]_{\mathsf{sk}'(A_r)}]\!\}_{\mathsf{ek}'(A_{r''})}\end{array}$$

where $(r', p') = \mathcal{S}(r, p)$, $(r, p) = \mathcal{S}(r'', p'')$ *and* $\mathsf{nonces} = \langle A_1, \ldots, A_k, \mathsf{nonces}_k\rangle$.

The initial control point is now set to -1 (or 0 for A_k) since actions have been added for the initialization stage. The special message fake is used to model for example the situation where an agent waits for more than one message in order to reply or when an agent sends more than one reply.

5 Main result

We identify a class of executions, which we call honest, single session executions which, intuitively, correspond to traces where just one session is executed, session in which all parties are honest and there is no adversary. Our only hypothesis will be that the initial protocol has to be secure in this very weak setting.

Definition 6 (Honest, single session trace). *Let $\Pi = (\mathcal{R}, \mathcal{S})$ be a k-party protocol and $\mathsf{tr} = (\mathsf{SId}_0, f_0, H_0) \xrightarrow{\alpha_1} \ldots \xrightarrow{\alpha_n} (\mathsf{SId}_n, f_n, H_n)$ be a valid trace of Π. The trace tr is an* honest, single session trace *if there are k agent identities $a_1, \ldots, a_k$ such that*

- *for $1 \leq i \leq k$, $\alpha_i = \mathbf{new}(i, a_1, \ldots, a_k)$,*
- *for $k+1 \leq i \leq n$, $\alpha_i = \mathbf{send}(\mathsf{sid}, m)$, $m = \mathsf{rcv}_r^p \sigma$ where $f_i(\mathsf{sid}) = (\sigma, r, p+1)$, and there exists $j < i$ such that $f_j(\mathsf{sid}') = (\sigma', r', p')$, $\mathcal{S}(r, p) = (r', p')$, and $m = \mathsf{snt}_{r'}^{p'} \sigma'$ for some sid'.*

Let $\mathsf{Exec}^{p,1}(\Pi)$ be the set of honest, single session traces of Π.

Definition 7 (Passive, single session satisfiability). *Let Π be a protocol and $\phi \in \mathcal{L}$ be a security property. We say that Π satisfies the security property ϕ for* passive adversaries and a single session*, and write $\Pi \models^{p,1} \phi$ if for any trace $\mathsf{tr} \in \mathsf{Exec}^{p,1}(\Pi)$, $[\![\phi(\mathsf{tr})]\!] = 1$.*

Transferable security properties. We identify a fragment $\mathcal{L}'$ of the logic $\mathcal{L}$ defined in Section 3 whose formulas specify the properties that can be transferred from the honest, single session case to the full active adversary case.

Definition 8. *The set $\mathcal{L}'$ consists of those formulas $\phi(\mathsf{tr})$ with*

$$\phi(\mathsf{tr}) = \forall \mathcal{LS}_{r,p}(\mathsf{tr}).\varsigma \Big(\bigwedge_{l \in [k]} \mathsf{NC}(\mathsf{tr}, \varsigma(A_l)) \Rightarrow \bigwedge_{i \in I} \big(\mathcal{Q}_i \, \mathcal{LS}_{r_i,p_i}(\mathsf{tr}).\varsigma_i \bigwedge_{j \in J_i} \tau_j^i(u_j^i, v_j^i) \big) \Big)$$

where $\mathcal{Q}_i \in \{\forall, \exists, \exists!\}$, and for all $i \in I$, for all $j \in J_i$, if $\mathcal{Q}_i = \forall$ then $\tau_j^i \in \{\neq\}$ and if $\mathcal{Q}_i \in \{\exists, \exists!\}$ then $\tau_j^i \in \{=, \neq\}$; moreover, for each $i \in I$, if $\mathcal{Q}_i = \forall$ (respectively $\mathcal{Q}_i = \exists!$) then for all (there is) $j \in J_i$ we have that (such that $\tau_j^i \in \{=\}$ and) there exists at least a subterm $\varsigma(X)$ in u_j^i or v_j^i with X a nonce or key variable.

As usual, we require security properties to hold in sessions between honest agents. This means that no guarantee is provided in a session where a corrupted agent is involved. But this does not prevent honest agents from contacting corrupted agents in parallel sessions. Properties that can be expressed in our fragment $\mathcal{L}'$ are correspondence relations between (data in) particular local states of agents in different sessions. It is a non-trivial class since e.g. the logical formulas given in Section 3 for expressing secrecy and authentication are captured by the above definition.

Transference result. The main result of this paper is the following transference theorem. It informally states that the formulas of $\mathcal{L}'$ that are satisfied in single, honest executions of a protocol are also satisfied by executions of the transformed protocol in the presence of a fully active adversary.

Theorem 1. *Let Π be a protocol and $\widetilde{\Pi}$ the corresponding transformed protocol. Let $\phi \in \mathcal{L}'$ be a security property. Then $\Pi \models^{p,1} \phi \Rightarrow \widetilde{\Pi} \models \phi$.*

The main intuition behind the proof is that any execution in the presence of an active adversary is closely mirrored by some *honest* execution (i.e. an execution with no adversarial interference plus some additional useless sessions). We define honest executions next.

Honest executions. Recall that we demand that protocols come with an intended execution order, in which the designer specifies the source of each message in an execution. Roughly, in an honest execution trace one can partition the set of session ids in sets of at most k role sessions (each corresponding to a different role of the protocol) such that messages are exchanged only within partitions, and the message transmission within each partition follows the intended execution specification. Since we cannot prevent an intruder to create new messages and sign them with corrupted signing keys, clearly the property can hold only for session identifiers corresponding to honest participants. The above ideas are captured by the following definition.

Definition 9 (Honest execution traces). *Let Π be an executable protocol. An execution trace $\mathsf{tr} = (\mathsf{SId}_0, f_0, H_0) \xrightarrow{\alpha_1} \ldots \xrightarrow{\alpha_n} (\mathsf{SId}_n, f_n, H_n)$ is* honest *if it is valid and there is a partition PrtSId of the honest role session identifiers $\mathsf{SId}^{\mathsf{h}}(\mathsf{tr})$ such that:*

1. *for all $S \in \mathsf{PrtSId}$, for all $\mathsf{sid}, \mathsf{sid}' \in S$ with $\mathsf{sid} \neq \mathsf{sid}'$ and $\mathsf{sid} = (s, r, (a_1, \ldots, a_k))$ and $\mathsf{sid}' = (s', r', (a'_1, \ldots a'_k))$, we have $r \neq r'$, and $a_j = a'_j$ for all $1 \leq j \leq k$; that is, in any protocol session each of the participants execute different roles[3] and the agents agree on their communication partners;*
2. *whenever $(\mathsf{SId}_{i-1}, f_{i-1}, H_{i-1}) \xrightarrow{\mathbf{send}(\mathsf{sid}, m)} (\mathsf{SId}_i, f_i, H_i)$ with $\mathsf{sid} \in \mathsf{SId}^{\mathsf{h}}(\mathsf{tr})$, m accepted, $m \neq \mathsf{fake}$ and $m = \mathsf{rcv}^p_r \sigma$, $p \geq 1$, we have that there are $\mathsf{sid}' \in [\mathsf{sid}]$ and $i' < i$ such that $m = \mathsf{snt}^{p'}_{r'} \sigma'$ and $\mathcal{S}(r, p) = (r', p')$ where $f_i(\mathsf{sid}) = (\sigma, r, p+1)$, $f_{i'}(\mathsf{sid}') = (\sigma', r', p')$, and $[\mathsf{sid}]$ is the partition to which sid belongs to.*

Notice that the above definition considers partial executions in which not all roles finish their execution, and where not all roles in a protocol session need to be initialized. The following lemma states that for any transformed protocol, an active intruder cannot interfere with the execution of honest sessions.

Lemma 1. *Let Π be a protocol and $\widetilde{\Pi}$ the corresponding transformed protocol. In $\widetilde{\Pi}$, any valid execution trace is an honest execution trace.*

Then it remains to show that any property expressed in $\mathcal{L}'$ that holds for one honest, single session trace also holds for any honest execution trace of the transformed protocol. It relies in particular on the fact that due to encryption, fresh values of honest sessions cannot occur in dishonest sessions. Moreover, honest execution traces actually correspond to the honest, single session trace of the initial protocol. More details and full proofs are available in [7].

[3] Consequently, each partition consists of at most k role sessions.

References

1. M. Abadi, C. Fournet, and G. Gonthier. Secure implementation of channel abstractions. *Inf. Comput.*, 174(1):37–83, 2002.
2. M. Bellare, R. Canetti, and H. Krawczyk. A modular approach to the design and analysis of authentication and key exchange protocols (extended abstract). In *STOC '98: Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 419–428, New York, NY, USA, 1998. ACM Press.
3. B. Blanchet and A. Podelski. Verification of cryptographic protocols: Tagging enforces termination. In A. Gordon, editor, *Foundations of Software Science and Computation Structures (FoSSaCS'03)*, volume 2620 of *LNCS*, April 2003.
4. H. Comon-Lundh and V. Cortier. New decidability results for fragments of first-order logic and application to cryptographic protocols. In *Proc. of the 14th Int. Conf. on Rewriting Techniques and Applications (RTA'2003)*, volume 2706 of *LNCS*, pages 148–164. Springer-Verlag, June 2003.
5. V. Cortier, H. Hördegen, and B. Warinschi. Explicit Randomness is not Necessary when Modeling Probabilistic Encryption. In *Workshop on Information and Computer Security (ICS 2006)*, Timisoara, Romania, September 2006.
6. V. Cortier and B. Warinschi. Computationally Sound, Automated Proofs for Security Protocols. In *Proc. 14th European Symposium on Programming (ESOP'05)*, volume 3444 of *LNCS*, pages 157–171, Edinburgh, U.K, April 2005. Springer.
7. V. Cortier, B. Warinschi, and E.Zălinescu. Synthesizing secure protocols. Inria research report, INRIA, Apr. 2007. Available at `http://www.loria.fr/~cortier/Papiers/compiler.pdf`.
8. A. Datta, A. Derek, J. C. Mitchell, and D. Pavlovic. A derivation system and compositional logic for security protocols. *J. Comput. Secur.*, 13(3):423–482, 2005.
9. N. Durgin, P. Lincoln, J. Mitchell, and A. Scedrov. Undecidability of bounded security protocols. In *Proc. of the Workshop on Formal Methods and Security Protocols*, 1999.
10. O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game. In *STOC '87: Proceedings of the nineteenth annual ACM conference on Theory of computing*, pages 218–229, New York, NY, USA, 1987. ACM Press.
11. L. Gong and P. Syverson. Fail-stop protocols: An approach to designing secure protocols. In *Proceedings of the 5th International Working Conference on Dependable Computing for Critical Applications (DCCA-5)*, pages 44–55, 1995.
12. J. Katz and M. Yung. Scalable protocols for authenticated group key exchange. In *Proceedings of Crypto'03*, pages 110–125. Springer-Verlag, 2003.
13. G. Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In Margaria and Steffen, editors, *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 1055 of *LNCS*, pages 147–166, 1996.
14. G. Lowe. A hierarchy of authentication specifications. In *CSFW 1997)*, Rockport, Massachusetts, USA, 1997. IEEE Computer Society Press.
15. G. Lowe. Towards a completeness result for model checking of security protocols. In *CSFW 1998)*. IEEE Computer Society Press, 1998.
16. R. Ramanujam and S. P. Suresh. A decidable subclass of unbounded security protocols. In *Proc. IFIP Workshop on Issues in the Theory of Security (WITS'03)*, pages 11–20, Warsaw (Poland), 2003.
17. M. Rusinowitch and M. Turuani. Protocol insecurity with finite number of sessions is NP-complete. In *CSFW 2001*, pages 174–190. IEEE Computer Society Press, 2001.