# Sound Security Protocol Transformations*

Binh Thanh Nguyen and Christoph Sprenger

Institute of Information Security, ETH Zurich, Switzerland
{thannguy,sprenger}@inf.ethz.ch

**Abstract.** We propose a class of protocol transformations, which can be used to (1) develop (families of) security protocols by refinement and (2) abstract existing protocols to increase the efficiency of verification tools. We prove the soundness of these transformations with respect to an expressive security property specification language covering secrecy and authentication properties. Our work clarifies and significantly extends the scope of earlier work in this area. We illustrate the usefulness of our approach on a family of key establishment protocols.

## 1 Introduction

It is well-known that security protocols are notoriously hard to get right. This motivates the use of formal methods for their design and development. The last decade has witnessed substantial progress in the formal verification of security protocols' properties such as secrecy and authentication. However, methods for transforming protocols have received much less attention.

Protocol transformations are interesting for at least two applications: we can use them (1) to develop (families of) protocols by refinement [17,9,16,7,6,4] and (2) to abstract existing protocols for the more efficient tool-based verification of their properties [11]. Abstraction and refinement correspond bottom-up and top-down views on (the same) protocol transformations. To be useful, protocol transformations must be sound with respect to a relevant class of security properties, i.e., refinement must be property-preserving, or equivalently, abstraction must be attack-preserving.

In this work, we propose a class of syntactic protocol transformations covering a wide range of protocols and security properties. Following Hui and Lowe [11], we support both message-based transformations, which we lift to protocol roles, and structural transformations, which directly operate on protocol roles. Message-based transformations include the removal of hashes or encryptions, pulling cleartext fields out of an encryption, and rearranging pair components. To guarantee the uniform transformation (e.g., removal) of variables and the messages they are supposed to receive, we work with typed messages. We use the type system of Arapinis and Duflot [2], which enables a fine-grained control over the message transformations. We establish the soundness of our typed

* This work is partially supported by the EU FP7-ICT-2009.1.4 Project No. 256980, NESSoS: Network of Excellence on Engineering Secure Future Internet Software Services and Systems.

transformations with respect to an expressive property specification language based on [14].

We make the following contributions. First, our work provides a sound formal underpinning for protocol transformations, which can serve as a foundation for rigorous security protocol development by refinement as well as for the abstraction of existing protocols. As an example of the latter, our approach helps to improve the performance of security protocol verifiers that are sensitive to message sizes such as SATMC [3]. Second, we extend existing work with respect to the expressiveness of the protocol specifications, the protocol transformations, and the preserved properties. In particular, we extend [11] in several ways: (1) we clarify and formally justify the application of transformations to protocol specifications, which contain variables not only ground terms as in [11]; (2) we support composed keys under a mild restriction; (3) we cover additional transformations (e.g., splitting encryptions) including many of those in [5,7,6]; and (4) we extend soundness to a more expressive property language based on predicates expressing event occurrence and ordering, intruder knowledge, and including quantification over thread identifiers.

The full version of this paper [15] includes the proofs of all our results and the treatment of structural transformations.

**A motivating example** We discuss the abstraction and refinement of key establishment protocols. We first take the abstraction view and defer a brief discussion of the refinement view to the end of this section. We start from a core version of Kerberos IV, called $\mathsf{K4}$, which we simplify in several steps with the aim of optimizing the performance of verification tools. In Alice&Bob notation, the protocol $\mathsf{K4}$ reads as follows.

$$\begin{aligned}
&\mathsf{K4}(1).\ A \to S : A, B, n_A\\
&\mathsf{K4}(2).\ S \to A : \{|B, t_S, n_A, k_{AB}, \{|A, t_S, k_{AB}|\}_{\mathsf{sh}(B,S)}|\}_{\mathsf{sh}(A,S)}\\
&\mathsf{K4}(3).\ A \to B : \{|A, t_S, k_{AB}|\}_{\mathsf{sh}(B,S)}, \{|c, t_A|\}_{k_{AB}}\\
&\mathsf{K4}(4).\ B \to A : \{|t_A|\}_{k_{AB}}
\end{aligned}$$

The security properties we are interested in include: (P1) the secrecy of the session key $k_{AB}$, (P2) $A$ authenticates $S$ on $k_{AB}$, $n_A$, and $t_S$, and (P3) $A$ and $B$ authenticate each other on $k_{AB}$ and $t_A$. To improve the performance of verification tools, we remove protocol elements that we deem unnecessary for a given property to hold and verify that property on the simplified protocol. If there is no attack then the soundness of our abstractions allows us to conclude that the original protocol also satisfies the property.

In the first abstraction step, we pull $B$'s ticket out of the encryption in message $\mathsf{K4}(2)$. The result is the core of Kerberos V, called $\mathsf{K5}$, which differs from $\mathsf{K4}$ as follows.

$$\mathsf{K5}(2).\ S \to A : \{|B, t_S, n_A, k_{AB}|\}_{\mathsf{sh}(A,S)}, \{|A, t_S, k_{AB}|\}_{\mathsf{sh}(B,S)}$$

In the second step, we eliminate the forwarding of $B$'s ticket by $A$ by applying structural transformations. This yields protocol $\mathsf{K3}$, on which we verify mutual

authentication of $A$ and $B$ (P3). We omit the message $\mathsf{K3}(1)$ which equals $\mathsf{K5}(1)$.

$$\begin{array}{ll}
\mathsf{K3}(2). & S \to A : \{\!|B, t_S, n_A, k_{AB}|\!\}_{\mathsf{sh}(A,S)} \\
\mathsf{K3}(3). & S \to B : \{\!|A, t_S, k_{AB}|\!\}_{\mathsf{sh}(B,S)} \\
\mathsf{K3}(4). & A \to B : \{\!|c, t_A|\!\}_{k_{AB}} \\
\mathsf{K3}(5). & B \to A : \{\!|t_A|\!\}_{k_{AB}}
\end{array}$$

In the third step, we remove the key confirmation phase, i.e., messages $\mathsf{K3}(4)$ and $\mathsf{K3}(5)$. For the resulting protocol, $\mathsf{K2}$, which we omit here, we verify the authentication property (P2).

In a final transformation, we remove the server timestamp $t_S$ and the initiator nonce $n_A$. The result is protocol $\mathsf{K1}$ for which we verify secrecy (P1).

$$\begin{array}{ll}
\mathsf{K1}(1). & A \to S : A, B \\
\mathsf{K1}(2). & S \to A : \{\!|B, k_{AB}|\!\}_{\mathsf{sh}(A,S)} \\
\mathsf{K1}(3). & S \to B : \{\!|A, k_{AB}|\!\}_{\mathsf{sh}(B,S)}
\end{array}$$

The protocols and transformations above will serve as running examples throughout the paper. We will report on experiments with SATMC in Section 4.

We can also view these transformations in the other direction, as a development of $\mathsf{K4}$ by refinement. We start from the abstract protocol $\mathsf{K1}$ satisfying session key secrecy (P1) and add new properties or modify the protocol structure with each refinement step. We verify properties (P2) and (P3) for $\mathsf{K2}$ and $\mathsf{K3}$, respectively, knowing that they are preserved by further refinements. By refining given protocols in different ways, we can develop entire protocol families, whose members share structure and properties. For example, most server-based key establishment protocols can be derived from $\mathsf{K1}$.

## 2 Security protocol model

### 2.1 Term algebra

We define a generic set of terms $\mathcal{T}(V, U, F, C)$ parametrized by four sets $V$, $U$, $F$, and $C$. We will instantiate these parameters to generate different sets of terms including those in protocol descriptions, network messages, and types, i.e., $V$ to variables, $U$ to roles or agents, $F$ to fresh values, and $C$ to constants, as well as to their associated types.

| | |
|---|---|
| $\mathcal{T} ::= V$ | variables |
| $\mid U \mid F \mid C$ | atoms: agents/roles, fresh values, constants |
| $\mid \mathsf{pk}(U) \mid \mathsf{pri}(U) \mid \mathsf{sh}(U,U)$ | atoms: long-term keys (public, private, shared) |
| $\mid \mathsf{h}(\mathcal{T}) \mid \langle \mathcal{T}, \mathcal{T} \rangle \mid \{\!|\mathcal{T}|\!\}_{\mathcal{T}}$ | composed terms: hashing, pairing, encryption |

We use $\mathcal{T}$ as a shorthand for $\mathcal{T}(V, U, F, C)$ in the generic case where the concrete parameters do not matter. We denote by $|t|$ the size of a term $t$. The set $St(t)$ denotes the set of subterms of $t$. For $T \subseteq \mathcal{T}$, $vars(T)$ and $atoms(T)$ denote the sets of variables and atoms in $St(T)$. A term without variables is called *ground*.

The terms $\mathsf{pk}(A)$, $\mathsf{pri}(A)$, and $\mathsf{sh}(A, B)$ for $A, B \in U$ denote the public key of $A$, the private key of $A$, and a symmetric key shared by $A$ and $B$. We define the function $(\cdot)^{-1}$ on ground terms $t$ as follows: $\mathsf{pk}(A)^{-1} = \mathsf{pri}(A)$, $\mathsf{pri}(A)^{-1} = \mathsf{pk}(A)$, and $t^{-1} = t$ otherwise. Next, we define a number of functions on terms in $\mathcal{T}$.

A multiset $m$ over a set $S$ is a function $m\colon S \to \mathbb{N}$, where $m(x)$ denotes the multiplicity of $x$ in $m$. The relations $\sqcap, \sqcup, \sqsubseteq$ denote multiset intersection, union, and inclusion, respectively, and $set(m) = \{x \in S \mid m(x) > 0\}$.

**Definition 1.** *We define the* pair splitting function *on terms as follows.*

$$\begin{aligned} split(u) &= \{u\} && \text{if } u \text{ is not a pair} \\ split(\langle u_1, u_2\rangle) &= split(u_1) \sqcup split(u_2) \end{aligned}$$

*We also define* $split(U) = \bigsqcup_{u \in U} split(u)$ *for a set* $U$ *of terms.*

**Definition 2.** *We define the set* $acc(t)$ *of* accessible subterms *of a term* $t$ *by*

$$\begin{aligned} acc(u) &= \{u\} && \text{if } u \text{ is a variable, atom, or hash} \\ acc(\langle u_1, u_2\rangle) &= acc(u_1) \cup acc(u_2) \\ acc(\{\!|u|\!\}_k) &= acc(u) \end{aligned}$$

### 2.2 Protocols

Let $\mathcal{V}$, $\mathcal{R}$, $\mathcal{F}$, and $\mathcal{C}$ be infinite and pairwise disjoint sets of variables, role names, fresh names, and constants. We define the set of *messages* by $\mathcal{M} = \mathcal{T}(\mathcal{V}, \mathcal{R}, \mathcal{F}, \mathcal{C})$.

We specify protocols in terms of roles. A *role* is a sequence of send and receive events of the form $\mathsf{snd}(t)$ or $\mathsf{rcv}(t)$ for a term $t \in \mathcal{M}$. We denote the set of all events by *Event*. We write $term(e)$ for the term contained in the event $e$. Let $mgu(t, u)$ denote the most general unifier of the terms $t$ and $u$.

**Definition 3 (Protocol).** *A protocol* role *is a sequence of events. We define* $Role = Event^*$. *A* protocol $P : \mathcal{R} \rightharpoonup Role$ *is a partial function from role names to roles such that*

1. *the sets of variables and fresh values in different roles are pairwise disjoint,*
2. *variables first occur in accessible positions of receive events, i.e., for all events* $e$ *in a role* $P(R)$ *and all variables* $X \in vars(term(e))$ *there is an event* $\mathsf{rcv}(t)$ *in* $P(R)$ *such that* $\mathsf{rcv}(t)$ *equals or precedes* $e$ *in* $P(R)$ *and* $X \in acc(t)$.
3. *the events in* $P$*'s roles can be exhaustively enumerated in a list of pairs of send and receive events* $[(s_1, r_1), \ldots, (s_m, r_m)]$. *We require that, for each* $i \in \{1, \ldots, m\}$, *there exist a substitution* $\delta_i$ *such that*
   - $\delta_1 = mgu(term(s_1), term(r_1))$, *and, for* $1 < k \leq m$,
   - $\delta_k = mgu(term(s_k)(\delta_{k-1} \circ \cdots \circ \delta_1), term(r_k)(\delta_{k-1} \circ \cdots \circ \delta_1))$.

   *We define* $\delta_P = \delta_m \circ \ldots \circ \delta_1$ *and call it the* honest substitution *of* $P$.

The second condition of this definition ensures that $\delta_P$ is a ground substitution.

Given a protocol $P$, let $\mathcal{V}_P$, $\mathcal{R}_P$, $\mathcal{F}_P$, and $\mathcal{C}_P$ be the sets of variables, role names, fresh values, and constants appearing in the roles of $P$ (i.e., $\mathcal{R}_P = dom(P)$). We assume a constant $\mathsf{nil} \in \mathcal{C} \setminus \mathcal{C}_P$ and define $\mathcal{C}_P^{\mathsf{nil}} = \mathcal{C}_P \cup \{\mathsf{nil}\}$. We denote by $Event_P$ the set of all events in the protocol $P$ and $Rt_P = term(Event_P)$. Moreover, we define the set of protocol messages (over the atomic messages of the protocol $P$) by $\mathcal{M}_P = \mathcal{T}(\mathcal{V}_P, \mathcal{R}_P, \mathcal{F}_P, \mathcal{C}_P^{\mathsf{nil}})$.

$$\frac{u \in T}{T \vdash u}\ \text{Axiom} \qquad \frac{T \vdash t \quad T \vdash u}{T \vdash \langle t, u \rangle}\ \text{Pair} \qquad \frac{T \vdash \langle t_1, t_2 \rangle}{T \vdash t_i}\ \text{Proj}_i$$

$$\frac{T \vdash u}{T \vdash \mathsf{h}(u)}\ \text{Hash} \qquad \frac{T \vdash t \quad T \vdash u}{T \vdash \{\!|t|\!\}_u}\ \text{Enc} \qquad \frac{T \vdash \{\!|t|\!\}_u \quad T \vdash u^{-1}}{T \vdash t}\ \text{Dec}$$

**Fig. 1.** Intruder deduction rules

### 2.3 Attacker model and operational semantics

Let $\mathcal{A}$ and *TID* be infinite sets of agents and thread identifiers. We partition $\mathcal{A}$ into non-empty sets of honest and compromised agents: $\mathcal{A} = \mathcal{A}_H \cup \mathcal{A}_C$.

When we instantiate a role into a thread for execution, we mark variables, role names, and fresh values of the respective role script with the thread identifier to distinguish them from those of other threads. Given a thread identifier $tid \in TID$, we define the instantiation function $inst_{tid}$ as the homomorphic extension of the following definition to all messages:

$$\begin{array}{lll} inst_{tid}(w) & = w^{tid} & \text{if } w \in \mathcal{V} \cup \mathcal{F} \cup \mathcal{R} \\ inst_{tid}(c) & = c & \text{if } c \in \mathcal{C} \\ inst_{tid}(k(R)) & = k(R^{tid}) & \text{if } R \in \mathcal{R}, k \in \{\mathsf{pk}, \mathsf{pri}\} \\ inst_{tid}(\mathsf{sh}(R,S)) & = \mathsf{sh}(R^{tid}, S^{tid}) & \text{if } R, S \in \mathcal{R} \end{array}$$

We define by $T^\sharp = \{inst_i(t) \mid t \in T \wedge i \in TID\}$ the set of instantiations of terms in a set $T$. Hence, the set of *instantiated messages* of protocol $P$ is $\mathcal{M}_P^\sharp$. We lift this to sets of events by instantiating the terms they contain, e.g., to define $Event_P^\sharp$. We also define the set of *network messages*, i.e., the ground messages transmitted over the network, by $\mathcal{N}_P = \mathcal{T}(\emptyset, \mathcal{A}, \mathcal{F}_P^\sharp \cup \mathcal{F}_P^\bullet, \mathcal{C}_P^{\mathsf{nil}})$, where $F^\bullet = \{f^\bullet \mid f \in F\}$ for $F \subseteq \mathcal{F}$ are attacker-generated fresh values. Furthermore, we abbreviate $\mathcal{M}_P^\square = \mathcal{M}_P \cup \mathcal{M}_P^\sharp \cup \mathcal{N}_P$.

*Attacker model* We use a standard Dolev-Yao attacker model. The intruder's capabilities for network messages are described by the deduction rules in Figure 1.

*Operational semantics* We define a transition system with states $(tr, th, \sigma)$, where

- $tr$ is a trace consisting of a sequence of pairs of thread identifiers and events,
- $th : TID \rightharpoonup \mathcal{R} \times Role$ is a thread pool denoting executing role instances, and
- $\sigma : \mathcal{V}^\sharp \cup \mathcal{R}^\sharp \to \mathcal{N}_P$ is a substitution with network messages as its range.

The trace $tr$ as well as the executing role instance are symbolic (with terms in $\mathcal{M}_P^\sharp$). The separate substitution $\sigma$ instantiates these messages to (ground) network messages. The ground trace associated with such a state is $tr\sigma$.

We define the (symbolic) intruder knowledge $IK(tr)$ derived from a trace $tr$ as the set of terms in the send events on $tr$, i.e., $IK(tr) = \{t \mid \exists i.\ (i, \mathsf{snd}(t)) \in tr\}$.

$$\frac{th(i) = (R, \mathsf{snd}(t) \cdot tl)}{(tr, th, \sigma) \to (tr \cdot (i, \mathsf{snd}(t)), th[i \mapsto (R, tl)], \sigma)} \; \mathit{SEND}$$

$$\frac{th(i) = (R, \mathsf{rcv}(t) \cdot tl) \quad IK(tr)\sigma \cup IK_0 \vdash t\sigma}{(tr, th, \sigma) \to (tr \cdot (i, \mathsf{rcv}(t)), th[i \mapsto (R, tl)], \sigma)} \; \mathit{RECV}$$

**Fig. 2.** Operational semantics

We associate with each protocol $P$ a fixed ground initial knowledge $IK_0$ and assume that $\mathcal{C} \cup \mathcal{A} \cup \mathcal{F}^\bullet \subseteq IK_0$. In particular, $\mathsf{nil} \in IK_0$.

In our model, the substitution $\sigma$ is chosen non-deterministically in the initial state. The set of initial states $Init_P$ of protocol $P$ contains all $(\epsilon, th, \sigma)$ satisfying

$$\forall i \in dom(th).\ \exists R \in \mathcal{R}_P.\ th(i) = (R, inst_i(P(R))) \wedge \sigma(\mathcal{R}^\sharp) \subseteq \mathcal{A},$$

where $inst_i$ is applied to all terms in the respective protocol role.

The state transitions are defined by the rules in Figure 2. In both the *SEND* and *RECV* rules, the first premise states that a send or receive event is in the first position of the role script of thread $i$. The executed event is removed from the role script and added together with the thread identifier to the trace $tr$. Transitions do not change the substitution $\sigma$, which is fixed in the initial state. The second premise of the *RECV* rule requires that the network message $t\sigma$ matching the term $t$ in the receive event is derivable from $IK(tr)\sigma \cup IK_0$, i.e., the intruder's (ground) knowledge derived from $tr$ and his initial knowledge $IK_0$.

### 2.4 Type system

We introduce a type system that extends Arapinis and Duflot's [2] with type variables, but is equivalent to theirs for ground types. In this type system, all roles and agent names have the same type $\alpha$ and similarly with each kind of long-term key (e.g., $\mathsf{pk}(\alpha)$ is the type of public keys). Each fresh value $f \in \mathcal{F}$ and constant $c \in \mathcal{C}$ has its own type: $\beta_f$ and $\gamma_c$. This enables a fine-grained control in our message transformations. The types of composed terms follow the structure of the terms.

Let $\mathcal{V}_{ty}$ be an infinite set of type variables disjoint from $\mathcal{V}$. We define the set of types by $\mathcal{Y} = \mathcal{T}(\mathcal{V}_{ty}, \{\alpha\}, \{\beta_f \mid f \in \mathcal{F}\}, \{\gamma_c \mid c \in \mathcal{C}\})$. A *typing environment* is a partial function $\Gamma : \mathcal{V} \rightharpoonup \mathcal{Y}$ assigning types to (message) variables. Typing judgements are of the form $\Gamma \vdash t : \tau$, where $\Gamma$ is a typing environment, $t$ is a term, and $\tau$ is a type. The derivable typing judgements are determined by the inference rules in Figure 3. The first row displays the rules for variables, fresh values, and constants. The first two rules assign the types given by the typing environment to plain and instantiated variables. The third and fourth rules give a unique type to each fresh value or constant. In the second row, the

$$\frac{(X,\tau)\in\Gamma}{\Gamma\vdash X:\tau}\qquad \frac{(X,\tau)\in\Gamma\quad i\in \mathit{TID}}{\Gamma\vdash X^i:\tau}\qquad \frac{f\in\mathcal{F}\cup\mathcal{F}^\sharp\cup\mathcal{F}^\bullet}{\Gamma\vdash f:\beta_f}\qquad \frac{c\in\mathcal{C}}{\Gamma\vdash c:\gamma_c}$$

$$\frac{U\in\mathcal{R}\cup\mathcal{R}^\sharp\cup\mathcal{A}}{\Gamma\vdash U:\alpha}\qquad \frac{U\in\mathcal{R}\cup\mathcal{R}^\sharp\cup\mathcal{A}}{\Gamma\vdash \mathsf{pk}(U):\mathsf{pk}(\alpha)}\qquad \frac{U\in\mathcal{R}\cup\mathcal{R}^\sharp\cup\mathcal{A}}{\Gamma\vdash \mathsf{pri}(U):\mathsf{pri}(\alpha)}\qquad \frac{U,V\in\mathcal{R}\cup\mathcal{R}^\sharp\cup\mathcal{A}}{\Gamma\vdash \mathsf{sh}(U,V):\mathsf{sh}(\alpha,\alpha)}$$

$$\frac{\Gamma\vdash t:\tau}{\Gamma\vdash h(t):h(\tau)}\qquad \frac{\Gamma\vdash t_1:\tau_1\quad\Gamma\vdash t_2:\tau_2}{\Gamma\vdash\langle t_1,t_2\rangle:\langle\tau_1,\tau_2\rangle}\qquad \frac{\Gamma\vdash t_1:\tau_1\quad\Gamma\vdash t_2:\tau_2}{\Gamma\vdash \{\!|t_1|\!\}_{t_2}:\{\!|\tau_1|\!\}_{\tau_2}}$$

**Fig. 3.** Type system

first rule assigns the agent type $\alpha$ to role names and agents and the remaining rules assign types to long-term keys. The third row shows the typing rules for composed terms, i.e., hashes, pairs, and encryptions.

The abbreviation $\mathcal{Y}_P = \mathcal{T}(\mathcal{V}_{ty}, \{\alpha\}, \{\beta_f \mid f \in \mathcal{F}_P\}, \{\gamma_c \mid c \in \mathcal{C}_P^{\mathsf{nil}}\})$, defines the set of types of a protocol $P$. We derive the canonical typing environment $\Gamma_P : \mathcal{V}_P \rightharpoonup \mathcal{Y}_P$ for the protocol $P$ from the honest substitution $\delta_P$ as $\Gamma_P = \{(X,\tau) \mid X \in \mathit{dom}(\delta_P) \wedge \emptyset \vdash X\delta_P : \tau\}$. Note that $\Gamma_P$ ranges over ground types.

**Proposition 1 (Type inference).** *Let $P$ be a protocol and $t \in \mathcal{M}_P^{\sqsubset}$. Then there is unique ground type $\tau \in \mathcal{Y}_P$ such that $\Gamma_P \vdash t : \tau$.*

By this proposition, we can extend $\Gamma_P$ to all terms $t \in \mathcal{M}_P^{\sqsubset}$, i.e., we have $\Gamma_P(t) = \tau$ if and only if $\Gamma_P \vdash t : \tau$. We say that a substitution is well-typed if the terms in its range respect the types of the variables in its domain.

**Definition 4 (Well-typed substitutions).** *A substitution $\theta$ is* well-typed *with respect to a typing environment $\Gamma$ iff $\Gamma \vdash X : \tau$ implies that $\Gamma \vdash X\theta : \tau$ for all $X \in \mathit{dom}(\theta)$.*

In this paper, we assume that it is sufficient to consider attacks with well-typed substitutions. There are multiple ways to achieve this. For example, tagging can be used in protocols that can fully decrypt all messages, in which case tag checking is sufficient to prevent all ill-typed attacks. Alternatively, we can use a result along the lines of [10,12,2,1] stating that there is a well-typed attack for any ill-typed one under certain conditions (e.g., sufficient tagging or well-formedness, which prevents the confusion of ciphertexts with different types).

## 3 Protocol transformations

Following Hui and Lowe [11], we distinguish two kinds of protocol transformations: *message-based* and *structural* transformations. Message-based transformations are functions on protocol messages, which we lift to events and protocol roles. In contrast, structural transformations apply directly to protocol roles.

We cover essentially the same structural transformations for splitting and relaying as [11]. The splitting transformation splits selected events with pairs into two events and the relaying transformation removes a $\mathsf{rcv}(X)$ and a subsequent $\mathsf{snd}(X)$ event from a protocol role. In Section 1, they together justify the step from protocol K5 to K3. The other abstractions, from K4 to K5, from K3 to K2, and from K2 to K1 are obtained by message-based transformations. Here, we mainly focus on message-based protocol transformations. However, structural transformations are discussed in the full version of this paper [15].

In Section 3.1, we introduce a class of message transformations, which includes the following operations on messages: (1) remove encryptions and hashes, (2) remove fields from an encrypted message, (3) pull fields outside of an encryption, (4) split encryption into several ones, and (5) project and reorder pairs.

Consider a logical language $\mathcal{L}$ to express security properties.We will define such a language in Section 4. We want to achieve three main properties for our transformations $f$ (both message-based and structural) and formulas $\phi$.

**Well-definedness** If $P$ is a protocol then so is $f(P)$, i.e., the three conditions of Definition 3 are preserved by $f$.

**Simulation** $f$ preserves reachability, i.e., if the state $(tr, th, \sigma)$ is reachable in $P$ then the transformed state $(f(tr), f(th), f(\sigma))$ is reachable in $f(P)$.[1]

**Attack preservation** For a state $(tr, th, \sigma)$ reachable in $P$ such that $(tr, th, \sigma) \not\models \phi$ we have $(f(tr), f(th), f(\sigma)) \not\models f(\phi)$.

The proofs of these three properties hinge on two more basic properties: the preservation of unifiers and of message deducibility. Unifier preservation is needed for well-definedness (the existence of an honest substitution) and attack preservation (for message equalities). Formally, this is expressed as follows.

$$t\theta = u\theta \;\Rightarrow\; f(t)f(\theta) = f(u)f(\theta) \tag{1}$$

Deducibility preservation is required for the simulation of receive events (see second premise of *RECV* rule) and attack preservation (for formulas expressing the intruder's knowledge). Formally, this property is stated as follows.

$$T\theta \cup IK_0 \vdash u\theta \;\Rightarrow\; f(T)f(\theta) \cup f(IK_0) \vdash f(u)f(\theta) \tag{2}$$

We further reduce the properties (1) and (2) to two simpler properties. First, we show in Section 3.2 deducibility preservation for ground terms: $T \vdash u$ implies $f(T) \vdash f(u)$ if all terms in $T \cup \{u\}$ are ground and the set $T$ satisfies an additional mild condition. Second, we establish the substitution property:

$$f(t\theta) = f(t)f(\theta). \tag{3}$$

This property (as well as (1) and (2)) does not hold for all transformations. The problem stems from the application of $f$ to terms with variables: a term $t$ and its instantiation $t\theta$ may be transformed in different ways (see Example 1 below).

[1] For now, you can read $f(\theta)$ as the composed substitution $f \circ \theta$.

We solve this problem by typing variables and restricting $\theta$ to well-typed substitutions. In Section 3.3, we thus introduce a restricted class of type-based message transformations, where a message's type uniquely determines how it is transformed. We use the type system from Section 2.4, which enables a fine-grained control over the transformations. In Section 3.4, we show that the substitution property (3) holds for type-based transformations $f$ and well-typed substitutions. Then we lift these transformations to protocols and establish well-definedness and the simulation property. Section 4 treats attack preservation.

### 3.1 Message transformations

We now introduce a class of message-based transformations. In these transformations, the constant $\mathsf{nil}$ plays a special role for the removal of (sub)terms. We remove variables and atoms by mapping them to $\mathsf{nil}$ and we rely on the following normalization function to remove the resulting $\mathsf{nil}$-subterms and eliminate trivial encryptions (with key $\mathsf{nil}$).

**Definition 5 (Normalization).**

$$\begin{aligned}
nf(t) &= t \qquad \textit{if } t \textit{ is a variable or an atom}\\
nf(h(t)) &= \mathsf{if}\ nf(t) = \mathsf{nil}\ \mathsf{then}\ \mathsf{nil}\ \mathsf{else}\ h(nf(t))\\
nf(\langle t_1, t_2\rangle) &= \mathsf{if}\ nf(t_1) = \mathsf{nil}\ \mathsf{then}\ nf(t_2)\\
&\quad\ \mathsf{else\ if}\ nf(t_2) = \mathsf{nil}\ \mathsf{then}\ nf(t_1)\\
&\quad\ \mathsf{else}\ \langle nf(t_1), nf(t_2)\rangle\\
nf(\{\!|t|\!\}_u) &= \mathsf{if}\ nf(t) = \mathsf{nil}\ \mathsf{then}\ \mathsf{nil}\\
&\quad\ \mathsf{else\ if}\ nf(u) = \mathsf{nil}\ \mathsf{then}\ nf(t)\\
&\quad\ \mathsf{else}\ \{\!|nf(t)|\!\}_{nf(u)}
\end{aligned}$$

*We say that a term $t$ is in* normal form *iff $nf(t) = t$.*

Note that $\mathsf{nil}$ can only occur in a normal-form term $t$ if $t$ equals $\mathsf{nil}$. We now formally define message transformations.

**Definition 6 (Message transformation).** *A function $f : \mathcal{T} \to \mathcal{T}$ is a* message transformation on $\mathcal{T}$ *if the following conditions hold:*

1. *for all non-normal form terms $t \in \mathcal{T}$, $f(t) = f(nf(t))$,*
2. *if $t \in nf(\mathcal{T})$ is a variable or an atom, then $f(t) = t$ or $f(t) = \mathsf{nil}$. Moreover, if $t$ is an asymmetric key then $f(t) = \mathsf{nil}$ if and only if $f(t^{-1}) = \mathsf{nil}$,*
3. *if $h(u) \in nf(\mathcal{T})$, then $f(h(u)) \in \{nf(h^a(f(u))) \mid a \geq 0\} \cup \{\mathsf{nil}\}$.*
4. *if $\langle u_1, u_2\rangle \in nf(\mathcal{T})$, then $f(\langle u_1, u_2\rangle) = nf(\langle f(t_1), \ldots, f(t_n)\rangle)$ for some terms $t_i$, $1 \leq i \leq n$, such that $\mathcal{P}(\langle u_1, u_2\rangle, \langle t_1, \ldots, t_n\rangle)$ and $|t_i| < |\langle u_1, u_2\rangle|$.*
5. *if $\{\!|u|\!\}_k \in nf(\mathcal{T})$, then for some $t_i$, $1 \leq i \leq n$ s.t. $\mathcal{P}(u, \langle t_1, \ldots, t_n\rangle)$, $|t_i| \leq |u|$, $a_i \geq 0$, and $b \geq 0$, $f(\{\!|u|\!\}_k) = nf(\{\!|\langle \{\!|f(t_1)|\!\}_{f(k)^{a_1}}, \ldots, \{\!|f(t_n)|\!\}_{f(k)^{a_n}}\rangle|\!\}_{f(k)^b})$.*

*where $\mathcal{P}(u, t) = split(t) \sqsubseteq split(u) \wedge set(split(t)) = set(split(u))$ and $\{\!|m|\!\}_{k^a}$ denotes the $a$-fold encryption of message $m$ with the key $k$.*

Condition 1 ensures that we only transform normal-form terms. Conditions 2-5 put restrictions on the transformation of the different kinds of messages. Note that we normalize the result of each transformation step. By Condition 2 we can either remove variables and atoms or keep them unchanged. Moreover, an asymmetric key and its inverse must be both removed or kept. This is necessary to achieve that *$f$ respects key inversion*, i.e., $f(t^{-1}) = f(t)^{-1}$ for all terms $t$. We need this property to prove deducibility preservation. Condition 3 enables two types of transformations for hashes: we can (a) add or remove hash function applications or (b) map it to $\mathsf{nil}$ (i.e., remove it completely).

Condition 4 allows us to arbitrarily rearrange the components of a pair provided that (a) every component of $\langle t_1, \ldots, t_n \rangle$ is also in $\langle u_1, u_2 \rangle$ but possibly with a smaller number of occurrences (expressed using $\mathcal{P}$) and (b) each term $t_i$ is smaller than the pair $\langle u_1, u_2 \rangle$. This ensures the well-foundedness of our definition and enables inductive proofs on term sizes. Similarly, Condition 5 describes the transformation of encryptions by splitting its plaintext into an arbitrary number of smaller terms $t_i$ (compared to the size of the plaintext). The terms $f(t_i)$ may be encrypted zero or more times with $f(k)$. This enables splitting and selective removal of encryptions.

### 3.2 Deducibility preservation

As mentioned above, our proof of deducibility preservation requires that $f$ respects key inversion. However, the conditions discussed above are not sufficient. For instance, we may have $f(\mathsf{h}(\mathsf{pk}(a))) = \mathsf{pk}(a)$ and therefore $f(\mathsf{h}(\mathsf{pk}(a))^{-1}) = \mathsf{pk}(a) \neq \mathsf{pri}(a) = f(\mathsf{h}(\mathsf{pk}(a)))^{-1}$. This shows that we must restrict the transformation of arbitrary terms into asymmetric keys. Therefore, we now introduce the notion of simple terms and we show that message transformations respect key inversion on simple ground terms.

**Definition 7 (Simple terms and simple-keyed term sets).** *A ground term $t \in \mathcal{T}$ is* simple *if it is an atom or it contains asymmetric keys only in key positions of encryptions. A set of ground terms $T$ is* simple-keyed *if $k$ is simple for all $\{\!|u|\!\}_k \in St(T)$.*

**Lemma 1.** *Let $f\colon \mathcal{T} \to \mathcal{T}$ be a message transformation and $t \in \mathcal{T}$ be a simple ground term. Then $f$ respects key inversion, i.e., $f(t^{-1}) = f(t)^{-1}$.*

Using this lemma, we establish deducibility preservation for simple-keyed sets of network messages.

**Theorem 1 (Deducibility preservation).** *Let $f$ be a message transformation on $\mathcal{N}_P$, $T \subseteq \mathcal{N}_P$ be a simple-keyed set of network messages and let $u \in \mathcal{N}_P$. Then $T \vdash u$ implies $f(T) \cup \{\mathsf{nil}\} \vdash f(u)$.*

We next present a more syntactic, type-based definition of message transformations for which the substitution property holds.

### 3.3 Type-based protocol transformations

We want to extend our message transformations to protocols. However, a simple lifting from messages to events, roles, and protocols will not work, since protocol roles contain variables and we cannot guarantee that a pair of matching send and receive events still matches after the transformation. Technically, this problem manifests itself as a failure of the substitution property (3) and unifier preservation (1) for some message transformations. Before giving an example, we extend message transformations to substitutions.

**Definition 8.** *$f\colon \mathcal{T} \to \mathcal{T}$ be a message transformation on $\mathcal{T}$ and $\theta : \mathcal{V} \rightharpoonup \mathcal{T}$. Then we define the substitution $f(\theta) = \{(x, f(\theta(x))) \mid x \in dom(\theta) \wedge f(x) = x\}$.*

Note that $dom(f(\theta)) \subseteq dom(\theta)$ as $f$ may map some variables in $dom(\theta)$ to $\mathsf{nil}$.

**Example 1.** Let $X$ be a variable and $\theta$ a ground substitution such that $f(X) = \mathsf{nil}$. For the substitution property to hold for $X$ and $\theta$, i.e., $f(X)f(\theta) = f(X\theta)$, we need $f(X\theta) = \mathsf{nil}$. Since $\theta$ is arbitrary so is $\theta(X)$. Hence, $f$ would have to map all terms to $\mathsf{nil}$, thus reducing $f$ to a trivial transformation. Similarly, $f(X)$ and $f(X\theta)$ are unifiable only if $f(X\theta) = \mathsf{nil}$. Hence, unifier preservation also fails.

In order to solve this problem we introduce type-based message transformations and restrict our attention to well-typed substitutions. Intuitively, in the typed setting, we can ensure that (1) a term and its (well-typed) instances have the same type and (2) all terms with the same type are transformed in a uniform way. We will guarantee this by having the type of a term alone determine how the term is transformed. This excludes situations like in Example 1 and enables us to establish the substitution property for well-typed substitutions (Section 3.4). Moreover, since the terms in matching send and receive events will have the same type, the typing ensures that the transformed events also match. This enables the lifting of type-based transformations to protocols.

The type system from Section 2.4 is well-suited for our purposes because it gives us a fine-grained control over the transformation of messages. More precisely, since each fresh value and constant has a different type, we can transform messages of similar shapes, but with different types in different ways. For example, we can remove the nonce $n_A$ from $\langle A, n_A\rangle$, while $\langle A, n_B\rangle$ remains unchanged.

**Specifying type-based transformations** In order to guarantee the uniform transformation of messages with the same type, our definition of type-based message transformations consists of two parts. The first part determines which terms are mapped to $\mathsf{nil}$ and therefore removed. It is specified as a set of types. The second part determines how composed messages are transformed and is specified using pattern matching on terms and types. In both cases, we have to ensure that it is only the type of a term, which determines how it is transformed. We define the semantics of these transformations as a functional program.

To avoid the need to introduce fresh variables in transformations, we now restrict our attention to protocols without variables of pair types. This is not a limitation, since we assume that protocol roles can always decompose pairs.

**Definition 9.** *A protocol $P$ is* splitting *iff, for all $X \in \mathcal{V}_P$, $X\delta_P$ is not a pair.*

*Function specifications* Let $\mathcal{V}_{pt}$ be an infinite set of *pattern variables* distinct from $\mathcal{V}$ and $\mathcal{V}_{ty}$. We construct term patterns from pattern variables using hashing, pairing and, encryption. Type patterns are types which contain (type) variables.

**Definition 10.** *The set of term patterns is defined by* $\mathcal{P} = \mathcal{T}(\mathcal{V}_{pt}, \emptyset, \emptyset, \emptyset)$. *A term pattern* $p \in \mathcal{P}$ *is* linear *if each pattern variable occurs at most once in* $p$.

We introduce a simple generic form of recursive function specifications. Based on these we will then define type-based transformations. Below, we use typing environments of the form $\Gamma : \mathcal{V}_{pt} \rightharpoonup \mathcal{Y}_P$ with pattern variables rather than message variables in the domain. Otherwise, the type system remains the same.

**Definition 11.** *Let* $f$ *be an unary function symbol.* A function specification for $f$ with respect to a typing environment $\Gamma\colon \mathcal{V}_{pt} \rightharpoonup \mathcal{Y}_P$ *is a list of equations*

$$E_f = [f(p_1 : \pi_1) = u_1, \ldots, f(p_n : \pi_n) = u_n],$$

*where each* $p_i \in \mathcal{P}$ *is a linear term pattern and* $\pi_i \in \mathcal{Y}_P$ *is a type pattern such that* $\Gamma \vdash p_i : \pi_i$. *The* $u_i$ *are terms, built from the pattern variables in* $p_i$, *cryptographic operations, and the function symbol* $f$.

We introduce the notion of a complete set of type patterns to ensure that each term's type matches some type pattern of a function specification. The use of type variables is essential to achieve this.

**Definition 12.** *A set of type patterns* $S \subseteq \mathcal{Y}_P$ *is* complete *w.r.t. a set of ground types* $T$ *if, for all* $\tau \in T$, *there is* $\pi \in S$ *such that* $\tau = \pi\theta$ *for some* $\theta : \mathcal{V}_{ty} \rightharpoonup \mathcal{Y}_P$.

**Example 2.** We define $E_0(f)$, the "homomorphic" function specification for $f$ with respect to $\Gamma_0(f) = \{(X, \mathcal{X}), (X', \mathcal{X}')\}$ below. Clearly, any set of patterns including the set $\{h(\mathcal{X}), \{\!|\mathcal{X}|\!\}_{\mathcal{X}'}, \langle \mathcal{X}, \mathcal{X}' \rangle\}$ is complete with respect to composed ground types in $\mathcal{Y}_P$.

$$\begin{aligned} E_0(f) = [&f(h(X) : h(\mathcal{X})) = h(f(X)), \quad f(\{\!|X|\!\}_{X'} : \{\!|\mathcal{X}|\!\}_{\mathcal{X}'}) = \{\!|f(X)|\!\}_{f(X')}, \\ &f(\langle X, X' \rangle : \langle \mathcal{X}, \mathcal{X}' \rangle) = \langle f(X), f(X') \rangle] \end{aligned}$$

*Transformation specifications* We can now make the two parts of the specification of a type-based transformation for a function symbol $f$ more precise. The first part is given by a set $T_f$ of atomic and ground hash types. The intention is that all terms composed from terms of these types by hashing, pairing, and encryption map to $\mathsf{nil}$ and are therefore removed. The second part handles composed terms and is given as a function specification $E_f$ for $f$ with respect to a $\Gamma_f$. By posing conditions on the term and type patterns, we ensure that the matching clause only depends on the term's type and that the restrictions on message shapes required for protocol transformations are satisfied.

**Definition 13 (Type-based message transformation).** *A* type-based message transformation *for a splitting protocol* $P$ *and function symbol* $f$ *is a triple* $S_f = (T_f, \Gamma_f, E_f)$ *satisfying the following conditions:*

1. $T_f \subseteq \mathcal{Y}_P \setminus \{\alpha\}$ *is a set of atomic and ground hash types such that* $\mathsf{pk}(\alpha) \in T_f$ *if and only if* $\mathsf{pri}(\alpha) \in T_f$,
2. $E_f = [f(p_1 : \pi_1) = u_1, \ldots, f(p_n : \pi_n) = u_n]$ *is a function specification for* $f$ *with respect to* $\Gamma_f : \mathcal{V}_{pt} \rightharpoonup \mathcal{Y}_P$ *such that*
   (a) $\{\pi_1, \ldots, \pi_n\}$ *is a complete set of patterns with respect to composed ground types, i.e., the ground types in the set* $\mathcal{Y}_P \setminus atoms(\mathcal{Y}_P)$, *and*
   (b) $p_i$ *is not deeper than* $\pi_i$ *for each* $1 \leq i \leq n$, *i.e., each term position in* $p_i$ *is also a position in* $\pi_i$.

   *Moreover, for all* $(f(p : \pi) = u) \in E_f$ *one of the following holds:*
   - $p = \mathsf{h}(q)$ *and* $u = \mathsf{h}^a(f(q))$, *where* $q \in \mathcal{V}_{pt}$ *and* $a \geq 0$,
   - $p = \langle q, r \rangle$ *and* $u = \langle f(t_1), \ldots, f(t_m) \rangle$, *where* $set(split(\langle q, r \rangle)) \subseteq \mathcal{V}_{pt}$, $split(\langle t_1, \ldots, t_m \rangle) = split(\langle q, r \rangle)$ *and* $|t_i| < |\langle q, r \rangle|$ *for* $1 \leq i \leq m$, *or*
   - $p = \{\!|q|\!\}_r$ *and* $u = \{\!|\langle \{\!|f(t_1)|\!\}_{f(r)^{a_1}}, \ldots, \{\!|f(t_m)|\!\}_{f(r)^{a_m}} \rangle|\!\}_{f(r)^b}$, *where* $set(split(q)) \cup \{r\} \subseteq \mathcal{V}_{pt}$, $a_i, b \geq 0$, $split(\langle t_1, \ldots, t_m \rangle) = split(q)$, *and* $|t_i| \leq |q|$ *for* $1 \leq i \leq m$.

We forbid $\alpha \in T_f$, since this would result in the removal of all role names from a protocol, which does not make much sense. The type of public and private keys can only be included together in $T_f$. For the case of pairs and encryptions, the linearity of the patterns $p_i$ implies that the subsumption relation $\mathcal{P}$ between two term tuples from Definition 6 reduces to an equality here.

*Transformation semantics* Before defining the semantics of type-based transformations, we formalize the set of types of those terms that we want to remove.

**Definition 14.** *For a set of ground types* $G$, *we define the* removable types $rem(G)$ *as the least set closed under the following rules.*

- *if* $\tau \in G$ *then* $\tau \in rem(G)$,
- *if* $\tau \in rem(G)$ *then* $h(\tau) \in rem(G)$,
- *if* $\tau_1, \tau_2 \in rem(G)$ *then* $\langle \tau_1, \tau_2 \rangle \in rem(G)$, *and*
- *if* $\tau \in rem(G)$ *then* $\{\!|\tau|\!\}_{\tau'} \in rem(G)$ *for all ground types* $\tau'$.

**Definition 15 (Semantics of typed-based transformations).** *The semantics of a type-based transformation* $S_f$ *for a splitting protocol* $P$ *and function symbol* $f$ *is given by Program 1.*

As said earlier, the main motivation for type-based setting is to achieve uniform transformations based on types, i.e., the type $\tau = \Gamma_P(t)$ of a term $t$ uniquely determines how $t$ is transformed ($\tau$ is well-defined by Proposition 1). We achieve this by ensuring that both (1) term removal and (2) pattern matching for composed types only depend on the type $\tau$. The program ensures point (1) by removing terms with types in $rem(T_f)$ (line 3). The lemma below guarantees that $rem(T_f)$ describes precisely these terms.

**Lemma 2.** *Let* $P$ *be a splitting protocol and* $S_f = (T_f, \Gamma_f, E_f)$ *be a type-based message transformation. Suppose* $t \in nf(\mathcal{M}_P^{\square}) \setminus \{\mathsf{nil}\}$ *and* $\Gamma_P \vdash t : \tau$. *Then* $\tau \in rem(T_f)$ *iff* $f(t) = \mathsf{nil}$.

```
1   fun f^rec(t) =
2     let τ = Γ_P(t) in
3       if τ ∈ rem(T_f) then nil
4       else if t ∈ vars(M_P^□) ∪ atoms(M_P^□) then t
5       else case (t, τ) of
6         (p_1, π_1) ⇒ nf(u_1[f^rec/f])
7       | ...
8       | (p_n, π_n) ⇒ nf(u_n[f^rec/f])
9
10  fun f(t) = f^rec(nf(t))
```

**Program 1.** Functional program resulting from specification $S_f = (T_f, \Gamma_f, E_f)$

Point (2) is guaranteed by Conditions (2a) and (2b) of Definition 13. A composed term's type uniquely determines a non-empty set of matching term-type patterns of $E_f$. This is expressed in the following lemma, which together with Lemma 2 will allow us to establish the substitution property.

**Lemma 3.** *Let $P$ be a splitting protocol and $S_f = (T_f, \Gamma_f, E_f)$ be a type-based message transformation for $P$, where $E_f = [f(p_1 : \pi_1) = u_1, \ldots, f(p_n : \pi_n) = u_n]$, and let $S(t, \tau) = \{i \mid \exists\theta.\ (p_i, \pi_i)\theta = (t, \tau)\}$. Then $S(t_1, \tau) = S(t_2, \tau) \neq \emptyset$ for all composed terms $t_1$, $t_2 \in \mathcal{M}_P^{\square}$ of ground type $\tau$ in environment $\Gamma_P$.*

As expected, type-based message transformations are indeed message transformation, as stated in the following proposition.

**Proposition 2.** *Let $P$ be a splitting protocol and $S_f$ be a type-based message transformation. Then $f$ is a message transformation on $\mathcal{M}_P^{\square}$ and also on $\mathcal{N}_P$.*

*Transforming protocols* We extend type-based transformations to events, roles and protocols. Transformed events with $\mathsf{nil}$ arguments are removed from roles.

**Definition 16 (Protocol transformations).** *Let $S_f$ be a type-based message transformation. We define $f(s(m)) = s(f(m))$ for events $s(m) \in Event$ and, for event sequences,*

$$f(\epsilon) = \epsilon \qquad f(e \cdot tl) = \mathsf{if}\ term(f(e)) = \mathsf{nil}\ \mathsf{then}\ f(tl)\ \mathsf{else}\ f(e) \cdot f(tl)$$

*For protocols $P$, $f(P)(R) = f(P(R))$ for $R \in dom(P)$ and undefined otherwise.*

Next, we present two examples of type-based message transformations formalizing some transformations from Section 1. The first one pulls a message out of an encryption and the second one removes some atoms from messages.

**Example 3 (K4 to K5).** We formalize the protocol $\mathsf{K4}$ as follows (where $c \in \mathcal{C}$).

$$\begin{aligned}
\mathsf{K4}(A) &= \mathsf{snd}(A, B, n_A) \cdot \mathsf{rcv}(\{\!|B, T_S, n_A, K_{AB}, X|\!\}_{\mathsf{sh}(A,S)}) \cdot \\
&\quad\ \mathsf{snd}(X, \{\!|c, t_A|\!\}_{K_{AB}}) \cdot \mathsf{rcv}(\{\!|t_A|\!\}_{K_{AB}}) \\
\mathsf{K4}(S) &= \mathsf{rcv}(A, B, N_A) \cdot \mathsf{snd}(\{\!|B, t_S, N_A, k_{AB}, \{\!|A, t_S, k_{AB}|\!\}_{\mathsf{sh}(B,S)}|\!\}_{\mathsf{sh}(A,S)}) \\
\mathsf{K4}(B) &= \mathsf{rcv}(\{\!|A, T'_S, K'_{AB}|\!\}_{\mathsf{sh}(B,S)}, \{\!|c, T_A|\!\}_{K'_{AB}}) \cdot \mathsf{snd}(\{\!|T_A|\!\}_{K'_{AB}})
\end{aligned}$$

The type-based message transformation $S_{f_4} = (T_{f_4}, \Gamma_{f_4}, E_{f_4})$, where $T_{f_4} = \emptyset$ and $E_{f_4}$ is defined using list concatenation @ and $E_0(f)$ from Example 2 as follows.

$$\begin{aligned} E_{f_4} &= [f_4(\{\!|X_1, X_2, X_3, X_4, X_5|\!\}_K : \{\!|\mathcal{X}_1, \mathcal{X}_2, \mathcal{X}_3, \mathcal{X}_4, \mathcal{X}_5|\!\}_{\mathsf{sh}(\alpha,\alpha)}) \\ &= \langle \{\!|f_4(X_1, X_2, X_3, X_4)|\!\}_K, f_4(X_5)\rangle] \mathbin{@} E_0(f_4) \end{aligned}$$

Applying $f_4$ to $\mathsf{K4}$ yields $\mathsf{K5} = f_4(\mathsf{K4})$ as follows. In this and the next example, we omit roles that are unchanged by the respective transformations.

$$\begin{aligned} \mathsf{K5}(A) &= \mathsf{snd}(A, B, n_A) \cdot \mathsf{rcv}(\{\!|B, T_S, n_A, K_{AB}|\!\}_{\mathsf{sh}(A,S)}, X) \cdot \\ &\quad\ \mathsf{snd}(X, \{\!|c, t_A|\!\}_{K_{AB}}) \cdot \mathsf{rcv}(\{\!|t_A|\!\}_{K_{AB}}) \\ \mathsf{K5}(S) &= \mathsf{rcv}(A, B, N_A) \cdot \mathsf{snd}(\{\!|B, t_S, N_A, k_{AB}|\!\}_{\mathsf{sh}(A,S)}, \{\!|A, t_S, k_{AB}|\!\}_{\mathsf{sh}(B,S)}) \end{aligned}$$

**Example 4 ($\mathsf{K3}$ to $\mathsf{K2}$).** Recall that $\mathsf{K3}$ results from $\mathsf{K5}$ by structural transformations $f_5$ eliminating the forwarding of $B$'s ticket by $A$. In $\mathsf{K3}$, defined below, there are therefore separate events for the server sending $A$ and $B$'s ticket and for $B$ receiving his ticket (from $S$) and the authenticator (from $A$).

$$\begin{aligned} \mathsf{K3}(A) &= \mathsf{snd}(A, B, n_A) \cdot \mathsf{rcv}(\{\!|B, T_S, n_A, K_{AB}|\!\}_{\mathsf{sh}(A,S)}) \cdot \\ &\quad\ \mathsf{snd}(\{\!|c, t_A|\!\}_{K_{AB}}) \cdot \mathsf{rcv}(\{\!|t_A|\!\}_{K_{AB}}) \\ \mathsf{K3}(S) &= \mathsf{rcv}(A, B, N_A) \cdot \mathsf{snd}(\{\!|B, t_S, N_A, k_{AB}|\!\}_{\mathsf{sh}(A,S)}) \cdot \mathsf{snd}(\{\!|A, t_S, k_{AB}|\!\}_{\mathsf{sh}(B,S)}) \\ \mathsf{K3}(B) &= \mathsf{rcv}(\{\!|A, T'_S, K'_{AB}|\!\}_{\mathsf{sh}(B,S)}) \cdot \mathsf{rcv}(\{\!|c, T_A|\!\}_{K'_{AB}}) \cdot \mathsf{snd}(\{\!|T_A|\!\}_{K'_{AB}}) \end{aligned}$$

The type-based message transformation $S_{f_3} = (T_{f_3}, \Gamma_{f_3}, E_{f_3})$ is defined by $T_{f_3} = \{\beta_{t_A}, \gamma_c\}$ and $E_{f_3} = E_0(f_3)$. Applying $f_3$ to $\mathsf{K3}$ yields protocol $\mathsf{K2} = f_3(\mathsf{K3})$ where the key confirmation messages have been removed.

$$\begin{aligned} \mathsf{K2}(A) &= \mathsf{snd}(A, B, n_A) \cdot \mathsf{rcv}(\{\!|B, T_S, n_A, K_{AB}|\!\}_{\mathsf{sh}(A,S)}) \\ \mathsf{K2}(B) &= \mathsf{rcv}(\{\!|A, T'_S, K'_{AB}|\!\}_{\mathsf{sh}(B,S)}) \end{aligned}$$

A further abstraction, $f_2$, removes $t_S$ and $n_A$ from $\mathsf{K2}$, resulting in protocol $\mathsf{K1}$.

### 3.4 Well-definedness and simulation

We are now in a position to establish the substitution property for splitting protocols and well-typed substitutions. Its proof uses Lemmas 2 and 3 above together the following lemma stating that well-typed substitutions preserve types.

**Lemma 4.** *Let $\theta$ be a well-typed substitution with respect to a typing environment $\Gamma$. Then for all terms $t \in \mathcal{T}$, $\Gamma \vdash t : \tau$ implies that $\Gamma \vdash t\theta : \tau$.*

**Theorem 2 (Substitution property).** *Let $P$ be a splitting protocol and $S_f$ be a type-based protocol transformation and $\theta$ be a well-typed substitution with respect to $\Gamma_P$. Then for all $t \in \mathcal{M}_P^{\square}$, we have $f(t\theta) = f(t)f(\theta)$.*

The first application of the substitution property is to establish well-definedness.

**Proposition 3 (Well-definedness).** *Let $P$ be a splitting protocol and $S_f$ be a type-based protocol transformation. Then $f(P)$ is a protocol with honest substitution $\delta_{f(P)} = f(\delta_P)$.*

Next, we lift deducibility preservation (Theorem 1) to non-ground terms and establish the simulation property. Since protocol descriptions contain non-ground terms, we restrict our attention to simple-keyed protocols, for which the set of (ground) *types* of the protocol's terms is simple-keyed. Hereafter, $IK_0$ and $IK'_0$ denote the intruder's initial knowledge associated with $P$ and $f(P)$, respectively.

**Definition 17.** *A protocol $P$ is* simple-keyed *if the set of types $\Gamma_P(Rt_P)$ is simple-keyed.*

**Lemma 5.** *If $P$ is a simple-keyed protocol, $T \subseteq Rt_P^\sharp$ and $\theta$ is well-typed ground substitution with respect to $\Gamma_P$, then $T\theta$ is a simple-keyed set of terms.*

**Proposition 4.** *Let $P$ be a simple-keyed, splitting protocol, $S_f$ a type-based message transformation, and $\theta$ a well-typed ground substitution with respect to $\Gamma_P$. Assume that $IK_0$ is simple-keyed and $f(IK_0) \subseteq IK'_0$. Then, for all $T \subseteq Rt_P^\sharp$ and $u \in \mathcal{M}_P^\sharp$, we have $T\theta \cup IK_0 \vdash u\theta$ implies $f(T)f(\theta) \cup IK'_0 \vdash f(u)f(\theta)$.*

**Theorem 3 (Simulation).** *Let $P$ be a simple-keyed, splitting protocol and let $S_f$ be a type-based message transformation. Assume that $IK_0$ is simple-keyed and $f(IK_0) \subseteq IK'_0$. Then for all states $(tr, th, \sigma)$ reachable in $P$ such that $\sigma$ is well-typed w.r.t. $\Gamma_P$, then $(f(tr), f(th), f(\sigma))$ is a reachable state of $f(P)$ and $f(\sigma)$ is well-typed w.r.t. $\Gamma_{f(P)}$.*

## 4 Property language and soundness

We introduce a specification language for security properties including secrecy and authentication. We extend our transformations to formulas of the property language and establish the preservation of well-typed attacks (and hence soundness) for protocols and formulas satisfying certain injectiveness conditions.

### 4.1 Security properties

Our property specification language is an instance of first-order logic with formulas in negation normal form (negation occurs only in front of atomic formulas).

$$\phi ::= A \mid \neg A \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \mid \forall i.\, \phi' \mid \exists i.\, \phi'$$

Here, $A$ are atomic predicates and the quantified variables $i$ represent thread identifiers. An atomic predicate or negated atomic predicate is called a *literal*. The atomic predicates and their meaning are as follows, where $m, m' \in \mathcal{M}_P^\sharp$ are messages, $e, e'$ are events, $i, j$ are thread-id variables, and $R$ is a role name.

| | | |
|---|---|---|
| $A ::=$ | $i = j$ | thread $i$ and thread $j$ are equal |
| $\mid$ | $m = m'$ | messages $m$ and $m'$ are equal |
| $\mid$ | $role(i, R)$ | thread $i$ executes role $R$ |
| $\mid$ | $honest(i, R)$ | the agent playing role $R$ in thread $i$'s view is honest |
| $\mid$ | $steps(i, e)$ | thread $i$ has executed event $e$ |
| $\mid$ | $(i, e) \prec (j, f)$ | thread $i$ has executed $e$ before thread $j$ has executed $f$ |
| $\mid$ | $secret(m)$ | the intruder does not know $m$ |

To achieve attack preservation, we focus on the fragment of this logic where the predicate $secret(m)$ only occurs positively. We call this language $\mathcal{L}_P$. A *property* is a closed formula of $\mathcal{L}_P$. In examples, we freely use standard abbreviations (e.g., for implication) if there is an equivalent negative normal form in $\mathcal{L}_P$.

Recall that $\mathcal{A}_H$ denotes the set of honest agents. Let $\vartheta$ be a substitution such that $range(\vartheta) \subseteq dom(th)$. We define formula satisfaction, $(tr, th, \sigma, \vartheta) \vDash \phi$, as follows (omitting the standard cases for the boolean operators and the dual existential quantifier):

$$
\begin{array}{ll}
(tr, th, \sigma, \vartheta) \vDash i = j & \text{iff } \vartheta(i) = \vartheta(j) \\
(tr, th, \sigma, \vartheta) \vDash m = m' & \text{iff } m\sigma = m'\sigma \\
(tr, th, \sigma, \vartheta) \vDash role(i, R) & \text{iff } \exists seq \in Event^*.\ th(\vartheta(i)) = (R, seq) \\
(tr, th, \sigma, \vartheta) \vDash honest(i, R) & \text{iff } R^{\vartheta(i)}\sigma \in \mathcal{A}_H \\
(tr, th, \sigma, \vartheta) \vDash steps(i, e) & \text{iff } (\vartheta(i), e) \in tr \\
(tr, th, \sigma, \vartheta) \vDash (i, e) \prec (j, e') & \text{iff } (\vartheta(i), e) \prec_{tr} (\vartheta(j), e') \\
(tr, th, \sigma, \vartheta) \vDash secret(m) & \text{iff } IK(tr)\sigma \cup IK_0 \vdash m\sigma \text{ is not derivable} \\
(tr, th, \sigma, \vartheta) \vDash \forall i.\, \phi' & \text{iff } (tr, th, \sigma, \vartheta[i \mapsto tid]) \vDash \phi' \text{ for all } tid \in dom(th)
\end{array}
$$

where $a \prec_{tr} b$ ("$a$ occurs before $b$ on $tr$") holds if $tr = tr_1 \cdot a \cdot tr_2 \cdot b \cdot tr_3$ for some $tr_1$, $tr_2$, $tr_3$. We write $(tr, th, \sigma, \vartheta) \nvDash \phi$ if $(tr, th, \sigma, \vartheta) \vDash \phi$ does not hold. If $\phi$ is a closed formula, we write $(tr, th, \sigma) \vDash \phi$ instead of $(tr, th, \sigma, \vartheta) \vDash \phi$.

**Definition 18 (Attack).** *We say that a state $s = (tr, th, \sigma)$ is an* attack on $\phi$ *if $s \nvDash \phi$. The state (attack) $s$ is well-typed if $\sigma$ is well-typed.*

We extend transformations $f$ to formulas $\phi \in \mathcal{L}_P$ as follows:

$$
\begin{array}{rlrl}
f(i = i') &= i = i' & f(secret(m)) &= secret(f(m)) \\
f(m = m') &= f(m) = f(m') & f(\neg A) &= \neg f(A) \\
f(role(i, R)) &= role(i, f(R)) & f(\phi_1 \wedge \phi_2) &= f(\phi_1) \wedge f(\phi_2) \\
f(honest(i, R)) &= honest(i, f(R)) & f(\phi_1 \vee \phi_2) &= f(\phi_1) \vee f(\phi_2) \\
f(steps(i, e)) &= steps(i, f(e)) & f(\forall i.\, \phi') &= \forall i.\, f(\phi') \\
f((i, e) \prec (j, e')) &= (i, f(e)) \prec (j, f(e')) & f(\exists i.\, \phi') &= \exists i.\, f(\phi')
\end{array}
$$

**Example 5 (Secrecy and authentication).** Consider the initiator and responder roles of the core Kerberos IV protocol $\mathsf{K4}$ as specified in Example 3.

$$
\begin{array}{rl}
\mathsf{K4}(A) = & \mathsf{snd}(A, B, n_A) \cdot \mathsf{rcv}(\{\!|B, T_S, n_A, K_{AB}, X|\!\}_{\mathsf{sh}(A,S)}) \cdot \\
& \mathsf{snd}(X, \{\!|c, t_A|\!\}_{K_{AB}}) \cdot \mathsf{rcv}(\{\!|t_A|\!\}_{K_{AB}}) \\
\mathsf{K4}(B) = & \mathsf{rcv}(\{\!|A, T'_S, K'_{AB}|\!\}_{\mathsf{sh}(B,S)}, \{\!|c, T_A|\!\}_{K'_{AB}}) \cdot \mathsf{snd}(\{\!|T_A|\!\}_{K'_{AB}})
\end{array}
$$

We express the secrecy of the session key $k_{AB}$ for role $A$ by

$$\phi_s = \forall i.\ (role(i, A) \wedge honest(i, [A, B]) \wedge steps(i, \mathsf{rcv}(t_2))) \Rightarrow secret(K^i_{AB}).$$

where $t_2 = \{\!|B, T_S, n_A, K_{AB}, X|\!\}_{\mathsf{sh}(A,S)}$ and $honest(i, [A, B])$ abbreviates the obvious conjunction. We abstract this property to verify it on the simplified

protocol $\mathsf{K1} = g(\mathsf{K4})$, where $g = f_2 \circ f_3 \circ f_5 \circ f_4$ is the composition of all transformations in our running example. Hence, we derive $\phi'_s = g(\phi_s)$, yielding

$$\phi'_s = \forall i.\ (\mathit{role}(i, A) \wedge \mathit{honest}(i, [A, B]) \wedge \mathit{steps}(i, \mathsf{rcv}(t'_2))) \Rightarrow \mathit{secret}(K^i_{AB}).$$

where $t'_2 = \{\!|B, K_{AB}|\!\}_{\mathsf{sh}(A,S)}$. Next, we formalize non-injective agreement of $B$ with $A$ on the key $k_{AB}$ and the timestamp $t_A$. This property is based on the authenticator.

$$\begin{aligned}\phi_a = \forall i.\ &(\mathit{role}(i, B) \wedge \mathit{honest}(i, [A, B]) \wedge \mathit{steps}(i, \mathsf{rcv}(u_1, u_2)))\\ &\Rightarrow \exists j.\ \mathit{role}(j, A) \wedge \mathit{steps}(j, \mathsf{snd}(X, \{\!|c, t_A|\!\}_{K_{AB}}))\\ &\qquad \wedge \langle A^i, B^i, K'^i_{AB}, T^i_A\rangle = \langle A^j, B^j, K^j_{AB}, t^j_A\rangle\end{aligned}$$

where $u_1 = \{\!|A, T'_S, K'_{AB}|\!\}_{\mathsf{sh}(B,S)}$ and $u_2 = \{\!|c, T_A|\!\}_{K'_{AB}}$. For the simplified protocol $\mathsf{K3} = f_5 \circ f_4(\mathsf{K4})$, we check the abstracted formula $\phi'_a = f_5 \circ f_4(\phi_a)$, where $B$'s ticket and the associated variable $X$ of role $A$ have been removed.

$$\begin{aligned}\phi'_a = \forall i.\ &(\mathit{role}(i, B) \wedge \mathit{honest}(i, [A, B]) \wedge \mathit{steps}(i, \mathsf{rcv}(u_2)))\\ &\Rightarrow \exists j.\ \mathit{role}(j, A) \wedge \mathit{steps}(j, \mathsf{snd}(\{\!|c, t_A|\!\}_{K_{AB}}))\\ &\qquad \wedge \langle A^i, B^i, K'^i_{AB}, T^i_A\rangle = \langle A^j, B^j, K^j_{AB}, t^j_A\rangle\end{aligned}$$

### 4.2 Soundness

We now show that if there exists a well-typed attack on a property $\phi$ of a protocol $P$, then the transformed attack state constitutes an attack on property $f(\phi)$ of protocol $f(P)$. In other words, we can say that the protocol $P$ is at least as secure as $f(P)$.

However, attack preservation does not hold for all properties $\phi$ and type-based message transformations $f$. For example, attacks on properties involving protocol events may not be preserved if $f$ maps two different events of $P$ to a single one in $f(P)$. Similarly, if $f$ identifies messages then attacks on equality are not preserved. These atomic predicates typically appear in authentication properties.

Our soundness result is therefore restricted to a subset of $(P, f)$-*safe* formulas of $\mathcal{L}_P$. We first define some auxiliary notions. Let $T^+_{eq}(\phi)$ be the set of pairs $(m, m')$ such that the equation $m = m'$ occurs positively in $\phi$ and let $T_{evt}(\phi)$ ($T^+_{evt}(\phi)$) be the set of events $s(m)$, $s'(m')$ such that $(i, s(m)) \prec (j, s'(m'))$ or $\mathit{steps}(i, s(m))$ occurs (positively) in $\phi$.

**Definition 19 ($(P, f)$-safe formulas).** *Let $P$ be a protocol and $S_f$ be a type-based message transformation for $P$ and function symbol $f$. A formula $\phi \in \mathcal{L}_P$ is* $(P, f)$-safe *if*

1. *$m\sigma \neq m'\sigma$ implies $f(m\sigma) \neq f(m'\sigma)$ for all $(m, m') \in T^+_{eq}(\phi)$ and well-typed ground substitutions $\sigma$,*
2. *$m \neq m'$ implies $f(m) \neq f(m')$ for all $s(m) \in T^+_{evt}(\phi)$ and $s(m') \in \mathit{Event}^\sharp_P$, and*
3. *$f(m) \neq \mathsf{nil}$ for all $s(m) \in T_{evt}(\phi)$.*

| protocol | K4/6 | | | K5/6 | | | K3/6 | | | K2/6 | | K1/6 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| property | *sec* | *aut* | *kc* | *sec* | *aut* | *kc* | *sec* | *aut* | *kc*∗ | *sec* | *aut*∗ | *sec*∗ |
| time [sec] | 1.45 | 1.31 | 1.16 | 20.65 | 20.5 | 18.27 | 1.44 | 1.31 | 1.18 | 0.95 | 0.85 | 0.14 |
| #clauses/1000 | 15.4 | 13.8 | 12.1 | 188.9 | 486.2 | 165.6 | 15.9 | 14.2 | 12.6 | 10.0 | 8.8 | 1.1 |
| #atoms/1000 | 2.0 | 1.9 | 1.9 | 33.0 | 32.9 | 32.8 | 2.0 | 2.0 | 1.9 | 1.4 | 1.3 | 0.4 |

**Table 1.** Experimental verification results for Kerberos (times in seconds); the abstraction level increases from left to right columns; (*) denotes highest abstraction level for marked properties.

**Theorem 4 (Attack preservation).** *Let $P$ be a simple-keyed, splitting protocol, $S_f$ a type-based message transformation for $P$ and function symbol $f$, and $\phi \in \mathcal{L}_P$ a $(P, f)$-safe property. Assume that $IK_0$ is simple-keyed and $f(IK_0) \subseteq IK'_0$. Then, for all well-typed states $(tr, th, \sigma)$ reachable in $P$, we have that $(f(tr), f(th), f(\sigma))$ is a well-typed reachable state of $f(P)$, and if $(tr, th, \sigma) \not\models \phi$ then $(f(tr), f(th), f(\sigma)) \not\models f(\phi)$.*

**Example 6.** Consider the protocol K4 and the typed-based message transformation $S_{f_4}$ from Example 3. We check that $\phi_s$ and $\phi_a$ from Example 5 are $(\mathsf{K4}, f_4)$-safe, i.e., satisfy the three conditions of Definition 19. The first condition holds for $\phi_s$, since $T^+_{eq}(\phi_s) = \emptyset$. It also holds for $\phi_a$, since $f_4(t) = t$ for all $t$ of the form $\langle t_1, t_2, t_3, t_4 \rangle$ such that $\Gamma_{\mathsf{K4}} \vdash t : \langle \alpha, \alpha, \beta_{k_{AB}}, \beta_{t_A} \rangle$. The second condition holds trivially for $\phi_s$ and it holds for $\phi_a$, since $f_4$ does not identify the only term $\langle X, \{|c, t_A|\}_{K_{AB}} \rangle \in T^+_{evt}(\phi_a)$ in its conclusion is not identified with another protocol event term. The third condition holds, since $f_4$ does not map any term appearing in a *steps* predicate of $\phi_s$ or $\phi_a$ to nil. Hence, the properties $\phi_s$ and $\phi_a$ are both $(\mathsf{K4}, f_4)$-safe. Since the protocol K4 is splitting and simple-keyed, Theorem 4 guarantees that the transformation $f_4$ preserves well-typed attacks on these properties.

### 4.3 Experimental results

We applied abstractions analogous to those described in this paper for the four-message core versions of Kerberos IV and V, K4 and K5, to the full six-message version of these protocols, K4/6 and K5/6. For the resulting protocols we have verified several secrecy and authentication properties using SATMC [3]. Our results are summarized in Table 1.

The columns denote the protocols and the properties verified. We grouped the properties into three classes: session key secrecy from the perspective of each role (*sec*), authentication properties involving a Kerberos server (*aut*), and key confirmation (*kc*). Those columns where the highest degree of abstraction for a given property class is achieved are marked with a star (∗). The rows show the verification time and the number of clauses and atoms of the SAT encoding (in thousands). The verification time is dominated by the encoding into a SAT problem whereas the SAT solving time is negligible.

We observe a slowdown from K4/6 to K5/6. We attribute this to the unencrypted responder ticket in K5/6, which increases the intruder's possibilities to interfere with the ticket variable $X$. The performance on K3/6 is similar to the one on K4/6. More interesting are the performance gains obtained by the further abstractions and the overall speedups that we achieve for the protocols K4 and K5. For example, verifying secrecy on K1/6 is 148 times faster than on K5/6 and still 10.4 times faster than on K4/6.

Additionally, we also used SATMC to verify a variant of the ISO/IEC 9798-3 three-pass mutual authentication protocol (ISO) and both secrecy and authentication for the TLS protocol (TLS). For both protocols we observed an enormous performance gain. For ISO, verification time for the initiator dropped from 107s to 0.2s (factor 535) by removing the responder's nonce and similarly for the responder. For TLS, we have reduced the verification for each property from more than 120s to less than 0.8s (factor 150) by removing fields that are irrelevant for the verified properties such as the cipher suite offer, session id, and certificate verification.

## 5 Related work

We can classify existing work on protocol transformations into syntactic and semantic approaches. Syntactic approaches use syntactic criteria to delimit a class of transformations for which soundness can be established a priori. Hui and Lowe [11] define several kinds of transformations similar to ours with the aim improving the performance of the CASPER/FDR model checker. They prove soundness of each kind of transformations based on general soundness criteria for secrecy and authentication. Their protocol model is restricted to atomic keys and they establish their results only for ground messages. We work in a more general setting and discuss in detail the non-trivial issue of handling terms with variables as they appear in protocol specifications. Other works [16,7,6] propose a set of syntactic transformations without however formally establishing their soundness.

Semantic approaches generally cover a larger class of transformations, but each transformation requires a separate proof for its justification. Examples are classical refinement and using abstract channels with security properties [17,4] and Guttman's protocol transformations based on strand spaces [9,8]. Sprenger and Basin [17] have recently proposed a refinement strategy for security protocols that spans several different abstraction levels (including, e.g., confidential and authentic channels). The transformations in the present paper belong to their most concrete level of cryptographic protocols. Guttman [9,8] studies the preservation of security properties by a rich class of protocol transformations in the strand space model. His approach to property preservation is based on the simulation of protocol analysis steps instead of execution steps. Each analysis step explains the origin of a received message. However, he does not provide syntactic conditions for the transformations' soundness.

## 6 Conclusions

We presented a large class of protocol transformations which is useful both for abstraction and refinement. We have shown its soundness with respect to an expressive property language. Our results constitute a significant extension of Hui and Lowe's work [11]. To validate our approach, we used our transformations to simplify the Kerberos, ISO, and TLS protocols. As a result, we achieved substantial performance improvements for SATMC. We also showed how to use our transformations in the other direction to refine the abstract protocol K1 into the core Kerberos IV and V protocols.

To handle terms with variables as they occur in protocol specifications, our transformations employ the type system given by Arapinis and Duflot [2]. The use of a type system is also motivated by the fact that there are type-flaw attacks that can be fixed by simple transformations that we would like to cover. For example, Meadows [13] presents such an attack on the full seven-message Needham-Schroeder-Lowe protocol, which can be fixed by swapping the components of a pair. Transformations fixing type-flaw attacks are obviously unsound. In a typed model, this problem is avoided since attacks based on type confusion are ruled out. In practice, well-typedness can be achieved by using appropriate tagging schemes [12,10]. Arapinis and Duflot [2] show that for secrecy properties of well-formed protocols it is sufficient to consider well-typed attacks. Well-formedness can be achieved by a lightweight tagging scheme. In her PhD thesis [1] (in French), Arapinis extends this result to a fragment of PS-LTL.

In future work, we want to formally justify the restriction to well-typed attacks for all properties expressible in our language $\mathcal{L}_P$. This could be achieved either by embedding our property language $\mathcal{L}_P$ into PS-LTL or by directly proving a similar result for $\mathcal{L}_P$. We also envision several other extensions. First, tool support to automate the abstraction process is needed. This should include automatic abstraction-refinement to find an appropriate abstraction for a given protocol and property. Second, we want to support additional transformations such as the context-dependent removal of message fields and the transformation of composed messages into atomic ones other than nil (e.g., to turn a Diffie-Hellmann exponentiation into a nonce). This will require the inclusion of freshness arguments in the soundness proof. Finally, extensions of the message algebra with equational theories and the adversary model would be useful (e.g., for modeling forward secrecy for Diffie-Hellmann protocols). However, it is not clear how to extend the typed setting to equational theories.

**Acknowledgements** We are grateful to David Basin, Ognjen Maric, and Cas Cremers for their useful comments on earlier drafts of this paper. We also thank the anonymous reviewers for their helpful feedback.

## References

1. M. Arapinis. *Sécurité des protocoles cryptographiques : décidabilité et résultats de réduction.* PhD thesis, Université Paris-Est, November 2008.

2. M. Arapinis and M. Duflot. Bounding messages for free in security protocols. In V. Arvind and S. Prasad, editors, *FSTTCS*, volume 4855 of *Lecture Notes in Computer Science*, pages 376–387. Springer, 2007.
3. A. Armando and L. Compagna. SAT-based model-checking for security protocols analysis. *International Journal of Information Security*, 7(1):3–32, 2008.
4. P. Bieber and N. Boulahia-Cuppens. Formal development of authentication protocols. In *Sixth BCS-FACS Refinement Workshop*, 1994.
5. I. Cervesato, C. Meadows, and D. Pavlovic. An encapsulated authentication logic for reasoning about key distribution protocols. In *CSFW '05: Proceedings of the 18th IEEE workshop on Computer Security Foundations*, pages 48–61, Washington, DC, USA, 2005.
6. A. Datta, A. Derek, J. C. Mitchell, and D. Pavlovic. Abstraction and refinement in protocol derivation. In *Proc. 17th IEEE Computer Security Foundations Workshop (CSFW)*, 2004.
7. A. Datta, A. Derek, J. C. Mitchell, and D. Pavlovic. A derivation system and compositionl logic for security protocols. *Journal of Computer Security*, 13:423–482, 2005.
8. J. D. Guttman. Transformations between cryptographic protocols. In P. Degano and L. Viganó, editors, *ARSPA-WITS*, volume 5511 of *LNCS*, pages 107–123. Springer, 2009.
9. J. D. Guttman. Security goals and protocol transformations. In *Theory of Security and Applications (TOSCA), an ETAPS associated event*, volume 6993 of *LNCS*. Springer, 2011.
10. 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.
11. M. L. Hui and G. Lowe. Fault-preserving simplifying transformations for security protocols. *Journal of Computer Security*, 9(1/2):3–46, 2001.
12. Y. Li, W. Yang, and C.-W. Huang. Preventing type flaw attacks on security protocols with a simplified tagging scheme. In J. Waldron, editor, *ISICT*, volume 90 of *ACM International Conference Proceeding Series*, pages 244–249. Trinity College Dublin, 2004.
13. C. Meadows. Analyzing the Needham-Schroeder public-key protocol: A comparison of two approaches. In E. Bertino, H. Kurth, G. Martella, and E. Montolivo, editors, *ESORICS*, volume 1146 of *Lecture Notes in Computer Science*, pages 351–364. Springer, 1996.
14. S. Meier, C. J. F. Cremers, and D. A. Basin. Strong invariants for the efficient construction of machine-checked protocol security proofs. In *Proc. 23th IEEE Computer Security Foundations Symposium (CSF)*, pages 231–245, 2010.
15. B. T. Nguyen and C. Sprenger. Sound security protocol transformations. Technical Report 781, Department of Computer Science, ETH Zurich, 2012.
16. D. Pavlovic and C. Meadows. Deriving secrecy in key establishment protocols. In *Proc. 11th European Symposium on Research in Computer Security (ESORICS)*, pages 384–403, 2006.
17. C. Sprenger and D. Basin. Refining key establishment. In *Proc. 25th IEEE Computer Security Foundations Symposium (CSF)*, pages 230–246, 2012.