


# Abstractions for security protocol verification

**Report****Author(s):**

Nguyen, Binh Thanh; Sprenger, Christoph 

**Publication date:**

2014

**Permanent link:**

<https://doi.org/https://doi.org/10.3929/ethz-a-010144557>

**Rights / license:**

[In Copyright - Non-Commercial Use Permitted](#)

# Abstractions for security protocol verification

Binh Thanh Nguyen and Christoph Sprenger

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

**Abstract.** We present a large class of security protocol abstractions with the aim of improving the scope and efficiency of verification tools. We present type-based abstractions, which use a term’s type to uniformly select the kind of abstraction applied, as well as untyped abstractions, which enable the removal of atomic messages, variables, and redundant terms. We extend existing work in the area by supporting additional abstractions, user-defined types, and untyped variables to cover type flaw attacks. We prove soundness results for an expressive property language that includes secrecy and authentication. Applying our abstractions to a set of realistic IETF protocol models, we achieve dramatic speedups and extend the scope of several modern security protocol analyzers.

## 1 Introduction

Security protocols play a central role in today’s networked applications. Past experience has amply shown that informal arguments justifying the security of such protocols are insufficient. This makes security protocols prime candidates for formal verification. In the last two decades, research in formal security protocol verification has made enormous progress (see [7] for a survey), which is reflected in many state-of-the-art tools including AVANTSSAR [4], ProVerif [9], Maude-NPA [18], Scyther [14], and Tamarin [27]. These tools can verify small to medium-sized protocols in a few seconds or less, sometimes for an unbounded number of sessions. Despite this success, they can still be challenged when verifying real-world protocols such as those defined in standards and deployed on the internet (e.g., TLS, IKE, and ISO/IEC-9798). Such protocols typically have messages with numerous fields, they support many alternatives (e.g., cryptographic setups), and may be composed from more basic protocols (e.g., IKEv2-EAP).

Abstraction [11] is a standard technique to over-approximate complex systems by simpler ones for verification. Sound abstractions preserve counterexamples (or attacks in security terms) from concrete to abstracted systems. In the context of security protocols, abstractions are extensively used. Here, we only mention a few examples. First, the Dolev-Yao model is a standard (not necessarily sound) abstraction of cryptography. Second, many tools use abstractions to map the verification problem into the formalism of an efficient solver or reasoner. We call these *back-end* abstractions. For example, ProVerif [9,10] translates models in the applied pi calculus to a set of Horn clauses, SATMC [6] reduces protocol verification to SAT solving, and Paulson [29] models protocols

as inductively defined trace sets. Finally, some abstractions aim at speeding up verification by simplifying protocols within a given protocol model before feeding them to verifiers [22,28]. Our work belongs to this class of *front-end* abstractions.

Extending Hui and Lowe’s work [22], we proposed in [28] a rich class of protocol abstractions and proved its soundness for a wide range of security properties. We used a type system to uniformly transform all terms of a given type (e.g., a pattern in a role specification and its instance during execution) whereas [22] only covers ground terms. We validated our approach using SATMC. The work in [28] exhibits several limitations. (1) We did not support untyped variables, which are required to capture type flaw attacks and to accurately model Diffie-Hellman exchanges or ticket forwarding. (2) While the types enable a fine-grained matching of messages, their distinction power may eliminate realistic attacks, e.g., by discerning the nonces of two roles. (3) Some soundness conditions involve quantification over substitutions and are thus hard to check in practice.

We see this paper’s contributions as follows. First, we extend our soundness result to a type system with untyped variables, user-defined types, and subtyping. User-defined types are useful to group similar atomic messages (e.g., session keys) and to adjust the granularity of matching in message transformations. Second, we have substantially simplified the specification of our abstractions into a list of recursive equations subject to some conditions on their shape. Moreover, we have extended their scope with additional abstractions, e.g., removing a hashed term completely or removing some of its fields. We also introduce untyped abstractions that complement the typed ones. Third, we pave the way for a tool implementation by providing syntactic criteria for the conditions of the soundness theorem. Thus, we have overcome the limitations mentioned above. Finally, we validate our approach on an extensive set of realistic case studies drawn from the IKEv1, IKEv2, ISO/IEC-9798, and PANA-AKA standard proposals. Our abstractions, which we currently do manually, result in very substantial performance gains for several state-of-the-art verifiers (Scyther, CL-Atse, OFMC, and SATMC) with different underlying techniques.

## 2 Example: The IKEv2-mac protocol

The Internet Key Exchange (IKE) family of protocols is part of the IPsec [25] protocol suite for securing Internet Protocol (IP) communication. IKE establishes a shared key, which is later used for securing IP packets. IKE also realizes mutual authentication and offers identity protection as an option. Its first version (IKEv1) dates back to 1998 [21]. The second version (IKEv2) [23,24] significantly simplifies the first one and has fewer options. However, the protocols in this family are still complex and contain a large number of fields.

*Concrete protocol.* As our running example, we present a member of the IKEv2 family, called IKEv2-mac (or  $\text{IKE}_m$  for short), which sets up a session key using a Diffie-Hellman (DH) key exchange, provides mutual authentication based on MACs, and also offers identity protection. We use Cremers’ models of IKE [15] as

a basis for our presentation and experiments (see Section 5). Our starting point is the following concrete  $\text{IKE}_m$  protocol between an initiator  $A$  and a responder  $B$ .

$$\begin{aligned} \text{IKE}_m(1). \quad & A \rightarrow B : \text{SPIa}, o, \text{sA1}, g^x, Na \\ \text{IKE}_m(2). \quad & B \rightarrow A : \text{SPIa}, \text{SPIb}, \text{sA1}, g^y, Nb \\ \text{IKE}_m(3). \quad & A \rightarrow B : \text{SPIa}, \text{SPIb}, \{A, B, \text{AUTHa}, \text{sA2}, \text{tSa}, \text{tSb}\}_{SK} \\ \text{IKE}_m(4). \quad & B \rightarrow A : \text{SPIa}, \text{SPIb}, \{B, \text{AUTHb}, \text{sA2}, \text{tSa}, \text{tSb}\}_{SK} \end{aligned}$$

Here,  $\text{SPIa}$  and  $\text{SPIb}$  denote the *Security Parameter Indices* that determine cryptographic algorithms,  $o$  is a constant number,  $\text{sA1}$  and  $\text{sA2}$  are *Security Associations*,  $g$  is the DH group generator,  $x$  and  $y$  are secret DH exponents,  $Na$  and  $Nb$  are nonces, and  $\text{tSa}$  and  $\text{tSb}$  denote *Traffic Selectors* specifying certain IP parameters.  $\text{AUTHa}$  and  $\text{AUTHb}$  denote the authenticators of  $A$  and  $B$  and  $SK$  the session key derived from the DH key  $g^{xy}$ . These are defined as follows.

$$\begin{aligned} SK &= \text{KDF}(Na, Nb, g^{xy}, \text{SPIa}, \text{SPIb}) \\ \text{AUTHa} &= \text{MAC}(\text{sh}(A, B), \text{SPIa}, o, \text{sA1}, g^x, Na, Nb, \text{prf}(SK, A)) \\ \text{AUTHb} &= \text{MAC}(\text{sh}(B, A), \text{SPIa}, \text{SPIb}, \text{sA1}, g^y, Nb, Na, \text{prf}(SK, B)) \end{aligned}$$

We model the functions  $\text{MAC}$ ,  $\text{KDF}$ , and  $\text{prf}$  as hash functions and use  $\text{sh}(A, B)$  and  $\text{sh}(B, A)$  to refer to the (single) long-term symmetric key shared by  $A$  and  $B$ .

We consider the following security properties: (P1) the secrecy of the DH key  $g^{xy}$ , which implies the secrecy of  $SK$ , and (P2) mutual non-injective agreement on the nonces  $Na$  and  $Nb$  and the DH half-keys  $g^x$  and  $g^y$ . Scyther verifies these for an unbounded number of sessions in 127 seconds on a 2.6 GHz Intel i7 laptop.

*Abstraction.* To speed up the verification, we simplify the protocol by removing unnecessary fields and operations. We abstract the protocol  $\text{IKE}_m$  in two steps. In the first step, we remove: (i) the symmetric encryptions with the session key  $SK$  (providing identity protection), (ii) from the session key: the application of  $\text{KDF}$  and all fields except the DH key  $g^{xy}$ , and (iii) from the authenticators: the fields  $\text{SPIa}$ ,  $\text{SPIb}$ , and  $\text{sA1}$  and the application of  $\text{prf}$  including the agent names underneath. In a second step, we remove the fields  $o$ ,  $A$ ,  $B$ ,  $\text{SPIa}$ ,  $\text{SPIb}$ ,  $\text{sA1}$ ,  $\text{sA2}$ ,  $\text{tSa}$ , and  $\text{tSb}$  in unprotected positions. The resulting protocol is  $\text{IKE}_m^2$ :

$$\begin{aligned} \text{IKE}_m^2(1). \quad & A \rightarrow B : g^x, Na & \text{IKE}_m^2(3). \quad & A \rightarrow B : \text{AUTHa} \\ \text{IKE}_m^2(2). \quad & B \rightarrow A : g^y, Nb & \text{IKE}_m^2(4). \quad & B \rightarrow A : \text{AUTHb} \end{aligned}$$

where the session key is now reduced to the DH key  $SK = g^{xy}$ , and

$$\begin{aligned} \text{AUTHa} &= \text{MAC}(\text{sh}(A, B), o, g^x, Na, Nb, SK) \\ \text{AUTHb} &= \text{MAC}(\text{sh}(B, A), g^y, Nb, Na, SK). \end{aligned}$$

Scyther verifies the properties (P1) and (P2) on this protocol in 3.8 seconds, which corresponds to a 33-fold speedup. The soundness of our abstractions means that our original protocol  $\text{IKE}_m$  also enjoys these properties (see Section 4).

In many of our experiments (Section 5), our abstractions either (i) resulted in an even more substantial speedup, or (ii) enabled the unbounded verification of a protocol where it timed out on the original protocol. In general, our abstractions may introduce false negatives. Hence, for each abstract attack we have to check its correspondence to a concrete one.

### 3 Security protocol model

We define term algebras  $\mathcal{T}_\Sigma(V)$  over a signature  $\Sigma$  and a set of variables  $V$  in the standard way. Our signature is  $\Sigma = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2$ , where  $\Sigma^0$  is a set of *atoms* of arity 0 that we will instantiate in different ways, and  $\Sigma^1 = \{\text{pk}, \text{pri}\} \cup \Sigma_u$  and  $\Sigma^2 = \{\text{sh}, \langle \cdot, \cdot \rangle, \{\cdot\}_\cdot, \{\cdot\}_\cdot\}$ . Here,  $\Sigma_u$  denotes a set of user-defined hash functions (of arity 1),  $\text{pk}(t), \text{pri}(t), \text{sh}(t, u)$  denote public, private, and shared keys, respectively, and  $\langle t, u \rangle, \{\cdot\}_u, \{t\}_{\text{pk}(u)}, \{t\}_{\text{pri}(u)}$  denote a pair, a symmetric encryption, a public-key encryption, and a signature, respectively. We assume pairs associate to the right, e.g.,  $\langle t, u, v \rangle$  denotes the term  $\langle t, \langle u, v \rangle \rangle$ . By instantiating  $\Sigma_0$  in different ways, we will generate different sets of terms, including terms in protocol descriptions, network messages, and types. Since  $\Sigma_1$  and  $\Sigma_2$  remain fixed, we will write  $\mathcal{T}(V, \Sigma_0)$  rather than  $\mathcal{T}_\Sigma(V)$ . We write  $\text{subs}(t)$  for the set of subterms of  $t$  and define the size of  $t$  by  $|t| = |\text{subs}(t)|$ . We also define  $\text{vars}(t) = \text{subs}(t) \cap V$ . If  $\text{vars}(t) = \emptyset$  then  $t$  is called *ground*. We define the *splitting* function by  $\text{split}(\langle t, u \rangle) = \text{split}(t) \cup \text{split}(u)$  on pairs and  $\text{split}(t) = \{t\}$  on other terms  $t$ .

Let  $\mathcal{V}, \mathcal{A}, \mathcal{F}$ , and  $\mathcal{C}$  be infinite and pairwise disjoint sets of variables, agent names, fresh values, and constants. We partition  $\mathcal{A}$  into non-empty sets of honest and compromised agents:  $\mathcal{A} = \mathcal{A}_H \cup \mathcal{A}_C$ . We assume there is a special constant  $\text{nil} \in \mathcal{C}$ . We define the set of *message terms* by  $\mathcal{M} = \mathcal{T}(\mathcal{V}, \mathcal{A} \cup \mathcal{F} \cup \mathcal{C})$ .

#### 3.1 Type system

We introduce a type system akin to [2] and extend it with subtyping. We define the set of atomic types by  $\mathcal{Y}^{\text{at}} = \mathcal{Y}_0 \cup \{\alpha, \text{msg}\} \cup \{\beta_n \mid n \in \mathcal{F}\} \cup \{\gamma_c \mid c \in \mathcal{C}\}$ , where  $\alpha, \beta_n$ , and  $\gamma_c$  are the types of agents, the fresh value  $n$ , and the constant  $c$ , respectively. Moreover,  $\text{msg}$  is the type of all messages and  $\mathcal{Y}_0$  is a disjoint set of user-defined types. The set of all types is then defined by  $\mathcal{Y} = \mathcal{T}(\emptyset, \mathcal{Y}^{\text{at}})$ .

A *typing environment* is a partial function  $\Gamma : \mathcal{V} \rightarrow \mathcal{Y}^{\text{at}}$  assigning atomic types to variables. We extend these to atoms by defining  $\Gamma(a) = \alpha$ ,  $\Gamma(n) = \beta_n$ , and  $\Gamma(c) = \gamma_c$  for  $a \in \mathcal{A}$ ,  $n \in \mathcal{F}$ , and  $c \in \mathcal{C}$ , and then homomorphically to all terms  $t \in \mathcal{M}$  such that  $\text{vars}(t) \subseteq \text{dom}(\Gamma)$ . If defined,  $\Gamma(t)$  is called the *type of  $t$*  under  $\Gamma$ . We assume that typing environments are nil-free, i.e.,  $\gamma_{\text{nil}} \notin \text{ran}(\Gamma)$ .

The subtyping relation  $\preceq$  on types is defined by the following inference rules and by two additional rules (not shown) defining its reflexivity and transitivity.

$$\frac{\tau \in \mathcal{Y}}{\tau \preceq \text{msg}} \text{S}(\text{msg}) \quad \frac{\tau_1 \preceq_0 \tau_2}{\tau_1 \preceq \tau_2} \text{S}(\preceq_0) \quad \frac{\tau_1 \preceq \tau'_1 \quad \dots \quad \tau_n \preceq \tau'_n \quad f \in \Sigma^n}{f(\tau_1, \dots, \tau_n) \preceq f(\tau'_1, \dots, \tau'_n)} \text{S}(\Sigma^n)$$

The first rule states that every type is a subtype of  $\text{msg}$ . The second rule embeds a user-defined *atomic subtyping relation*  $\preceq_0 \subseteq (\mathcal{Y}^{\text{at}} \setminus \{\gamma_{\text{nil}}, \text{msg}\}) \times \mathcal{Y}_0$ , which relates atomic types (except  $\gamma_{\text{nil}}$  and  $\text{msg}$ ) to user-defined atomic types in  $\mathcal{Y}_0$ . For simplicity, we require that  $\preceq_0$  is a partial function. The third rule ensures that subtyping is preserved by all constructors. For a type  $\tau \in \mathcal{Y}$ , we define the set of its subtypes by  $\tau \downarrow = \{\tau' \in \mathcal{Y} \mid \tau' \preceq \tau\}$ .

**Definition 1 (Well-typed substitutions).** A substitution  $\theta$  is well-typed with respect to a typing environment  $\Gamma$  if  $\Gamma(X)$  and  $\Gamma(X\theta)$  are both defined, and  $\Gamma(X\theta) \preceq \Gamma(X)$  for all  $X \in \text{dom}(\theta)$ .

### 3.2 Protocols

We define the set of events  $\text{Evt}(T) = \{\text{snd}(t), \text{rcv}(t) \mid t \in T\}$  and write  $\text{term}(e)$  for the term in the event  $e$ . A role is a sequence of events from  $\text{Evt}(\mathcal{M})$ .

**Definition 2 (Protocol).** A protocol is a pair  $P = (\Gamma_P, S_P)$  where  $\Gamma_P$  is a typing environment and  $S_P : \Gamma_P^{-1}(\alpha) \rightarrow \text{Evt}(\mathcal{M})^*$  maps agent variables to roles. Let  $\mathcal{M}_P = \text{term}(\text{set}(\text{ran}(S_P)))$  be the set of protocol terms appearing in the roles of  $P$ , and let  $\mathcal{V}_P$ ,  $\mathcal{A}_P$ ,  $\mathcal{F}_P$ , and  $\mathcal{C}_P$  denote the sets of variables, agent names, fresh values, and constants in  $\mathcal{M}_P$ . We require  $\mathcal{V}_P \subseteq \text{dom}(\Gamma_P)$  and  $\text{nil} \notin \mathcal{C}_P$ .

*Example 1 (IKE<sub>m</sub> protocol).* We formalize the IKE<sub>m</sub> protocol from Section 2 as  $\text{IKE}_m = (\Gamma_{\text{IKE}_m}, S_{\text{IKE}_m})$  with typing environment  $\Gamma_{\text{IKE}_m} = [sPIa, sPIb, Na, Nb : \text{nonce}, A, B : \alpha, Ga, Gb : \text{msg}]$ . We use upper-case (lower-case) identifiers for variables (atoms). The user-defined type *nonce* satisfies  $\beta_n \preceq_0 \text{nonce}$  for  $n \in \{sPIa, sPIb, na, nb, x, y\}$ . The terms  $sA1$ ,  $sA2$ ,  $tSa$ ,  $tSb$ , and  $o$  are constants. We represent  $g^x$  and  $(g^x)^y$  by the terms  $g(x)$  and  $h(g(x), y)$  where  $g, h \in \Sigma_u$  are hash functions. We show the initiator  $A$ 's role. The responder  $B$ 's is dual.

$$\begin{aligned} S_{\text{IKE}_m}(A) = & \text{snd}(sPIa, o, sA1, g(x), na) \cdot \text{rcv}(sPIa, sPIb, sA1, Gb, Nb) \cdot \\ & \text{snd}(sPIa, sPIb, \{A, B, AUTHaa, sA2, tSa, tSb\}_{SKa}) \cdot \\ & \text{rcv}(sPIa, sPIb, \{B, AUTHba, sA2, tSa, tSb\}_{SKa}) \end{aligned}$$

where  $SKa = \text{KDF}(na, Nb, h(Gb, x), sPIa, sPIb)$  and

$$\begin{aligned} AUTHaa &= \text{MAC}(\text{sh}(A, B), sPIa, o, sA1, g(x), na, Nb, \text{prf}(SKa, A)) \\ AUTHba &= \text{MAC}(\text{sh}(A, B), sPIa, sPIb, sA1, Gb, Nb, na, \text{prf}(SKa, B)). \end{aligned}$$

The terms  $SKa$ ,  $AUTHaa$ , and  $AUTHba$  represent the initiator  $A$ 's view of the session key and the initiator and the responder's authenticator.

Since our model does not support equational theories, we under-approximate the congruence relation induced by the equation  $h(g(x), y) = h(g(y), x)$  using *oracle roles*, which translate between equivalent term representations. One such role translates  $h(g(x), y)$  into  $h(g(y), x)$ . A second one translates between  $SKa$  and  $SKb$ . A third oracle role ensures the protocol's executability by translating the messages sent by role  $A$  into those received by role  $B$  and vice versa. ♠

### 3.3 Operational semantics

Let  $TID$  be a countably infinite set of thread identifiers. When we instantiate a role into a thread  $tid$  for execution, we mark its variables and fresh values with the thread identifier  $tid$ . We define the instantiation function  $\text{inst}_{tid}$  for  $tid \in TID$  as the homomorphic extension of the following definition to all messages:

$$\text{inst}_{tid}(w) = w^{tid} \text{ for } w \in \mathcal{V} \cup \mathcal{F} \quad \text{and} \quad \text{inst}_{tid}(c) = c \text{ for } c \in \mathcal{A} \cup \mathcal{C}$$

$$\begin{array}{c}
\frac{u \in T}{T \vdash u} \text{ (Ax)} \quad \frac{T \vdash t_1 \quad \cdots \quad T \vdash t_n}{T \vdash f(t_1, \dots, t_n)} \text{ (Comp)} \quad (f \in \Sigma_{\text{pub}}) \\
\frac{T \vdash \langle t_1, t_2 \rangle}{T \vdash t_i} \text{ (Proj}_i\text{)} \quad \frac{T \vdash \{t\}_u \quad T \vdash u}{T \vdash t} \text{ (Sdec)} \quad \frac{T \vdash \{t\}_{\text{pk}(u)} \quad T \vdash \text{pri}(u)}{T \vdash t} \text{ (Adec)}
\end{array}$$

**Fig. 1.** Intruder deduction rules (where  $\Sigma_{\text{pub}} = (\Sigma^1 \cup \Sigma^2) \setminus \{\text{pri}, \text{sh}\}$ )

We define by  $T^\sharp = \{\text{inst}_i(t) \mid t \in T \wedge i \in \text{TID}\}$  the set of instantiations of terms in a set  $T$  and abbreviate  $T^\flat = T \cup T^\sharp$ . For example,  $\mathcal{M}^\sharp$  is the set of instantiated message terms, which we will use to instantiate roles into threads.

We define the set of *network messages* exchanged during protocol execution by  $\mathcal{N} = \mathcal{T}(\mathcal{V}^\sharp, \mathcal{A} \cup \mathcal{F}^\sharp \cup \mathcal{F}^\bullet \cup \mathcal{C})$ , where  $\mathcal{F}^\bullet = \{n_k^\bullet \mid n \in \mathcal{F} \wedge k \in \mathbb{N}\}$  is the set of attacker-generated fresh values. Note that  $\mathcal{M}^\sharp \subseteq \mathcal{N}$ . We abbreviate  $\mathcal{T} = \mathcal{M} \cup \mathcal{N}$ .

We extend the typing environments  $\Gamma$  to instantiated terms in  $\mathcal{N}$  by defining  $\Gamma(n^i) = \Gamma(n_k^\bullet) = \Gamma(n)$  for  $n \in \mathcal{F}$  and  $\Gamma(X^i) = \Gamma(X)$  for all  $X \in \text{dom}(\Gamma)$ . Note that  $\Gamma(t)$  is defined for a term  $t \in \mathcal{T}$  provided that  $\text{vars}(t) \subseteq \text{dom}(\Gamma)^\flat$ .

*Semantics* We use a standard Dolev-Yao attacker model. The intruder's capabilities for network messages are described by the deduction rules in Figure 1. Note that these rules model signatures without message recovery.

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,
- $th : \text{TID} \rightarrow \text{dom}(S_P) \times \text{Evt}(\mathcal{M}_P^\sharp)^*$  are *threads* executing role instances, and
- $\sigma : \text{dom}(\Gamma_P)^\sharp \rightarrow \mathcal{N}$  is a ground substitution from instantiated protocol variables to network messages that is well-typed with respect to  $\Gamma_P$ .

The trace  $tr$  as well as the executing role instance are symbolic (with terms in  $\mathcal{M}^\sharp$ ). 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.,  $IK(tr) = \{t \mid \exists i. (i, \text{snd}(t)) \in tr\}$ . We associate with each protocol  $P$  a ground initial intruder knowledge  $IK_0$  and assume that  $\mathcal{A} \cup \mathcal{C} \cup \mathcal{F}^\bullet \subseteq IK_0$ . In particular, we have  $\text{nil} \in IK_0$ .

The set  $\text{Init}_P$  of initial states of protocol  $P$  contains all  $(\epsilon, th, \sigma)$  satisfying

$$\forall i \in \text{dom}(th). \exists R \in \text{dom}(S_P). th(i) = (R, \text{inst}_i(S_P(R)))$$

where  $\text{inst}_i$  is applied to all terms in the respective protocol role. This means that the substitution  $\sigma$  is chosen non-deterministically in the initial state.

The rules in Figure 2 define the state transitions. In both rules, the first premise states that a send or receive event is at the head thread  $i$ 's role. This event is removed from the role and added together with the thread identifier  $i$  to the trace  $tr$ . The substitution  $\sigma$  remains unchanged. The second premise of

$$\frac{th(i) = (R, \text{snd}(t) \cdot tl)}{(tr, th, \sigma) \rightarrow (tr \cdot (i, \text{snd}(t)), th[i \mapsto (R, tl)], \sigma)} \text{SEND}$$

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

**Fig. 2.** Operational semantics

*RECV* requires that the network message  $t\sigma$  matching the term  $t$  in the receive event is derivable from the intruder's (ground) knowledge  $IK(tr)\sigma \cup IK_0$ . Note that the *SEND* rule implicitly updates this intruder knowledge.

### 3.4 Property language

We introduce a specification language for security properties including secrecy and authentication. It is a first-order logic with formulas in negation normal form (i.e., only atomic formulas can be negated) with variables for terms and for thread identifiers. Quantification is allowed only over thread identifier variables. The atomic predicates and their informal meaning are as follows. Here,  $t, u \in \mathcal{M}^\sharp$  are instantiated messages with variables in  $\text{dom}(\Gamma_P)^\sharp$ ,  $e, e' \in \text{Evt}(\mathcal{M}_P)$  are protocol events,  $i, j$  are thread-id variables, and  $R \in \text{dom}(S_P)$  is a role name.

$A ::= i = j$	thread $i$ and thread $j$ are equal
$t = u$	messages $t$ and $u$ are equal
$\text{role}(i, R)$	thread $i$ executes role $R$
$\text{honest}(i, R)$	the agent playing role $R$ in thread $i$ 's view is honest
$\text{steps}(i, e)$	thread $i$ has executed event $e$
$(i, e) \prec (j, e')$	thread $i$ has executed $e$ before thread $j$ has executed $e'$
$\text{secret}(t)$	the intruder does not know message $t$

To achieve attack preservation, we focus on the fragment of this logic where the predicate  $\text{secret}(t)$  only occurs positively. We call this language  $\mathcal{L}_P$ . A *property* is a formula of  $\mathcal{L}_P$  without free thread-id variables. In examples, we freely use standard abbreviations (e.g., for implication) if there is an equivalent negative normal form in  $\mathcal{L}_P$ . The semantics of our language  $\mathcal{L}_P$  is given in Appendix A.1.

*Example 2 (Properties of  $\text{IKE}_m$ ).* We express the secrecy of the Diffie-Hellman key  $h(Gb, x)$  for role  $A$  of the protocol  $\text{IKE}_m$  of Example 1 as follows.

$$\phi_s = \forall j. (\text{role}(j, A) \wedge \text{honest}(j, [A, B]) \wedge \text{steps}(j, \text{rcv}(t_4))) \Rightarrow \text{secret}(h(Gb^j, x^j)).$$

where  $t_4 = \langle \text{SPIa}, \text{SPIb}, \{[B, \text{AUTHba}, \text{sA2}, \text{tSa}, \text{tSb}]_{SK_a}\} \rangle$  and  $\text{honest}(j, [A, B])$  denotes the expected conjunction. We also formalize non-injective agreement of  $A$  with  $B$  [26] on the nonces  $na$  and  $nb$  and the DH half-keys  $g(x)$  and  $g(y)$  by

$$\begin{aligned} \phi_a &= \forall j. (\text{role}(j, A) \wedge \text{honest}(j, [A, B]) \wedge \text{steps}(j, \text{rcv}(t_4))) \\ &\Rightarrow (\exists k. \text{role}(k, B) \wedge \text{steps}(k, \text{snd}(\langle \text{SPIa}, \text{SPIb}, \text{sA1}, g(y), nb \rangle))) \\ &\quad \wedge \langle A^j, B^j, na^j, Nb^j, g(x^j), Gb^j \rangle = \langle A^k, B^k, Na^k, nb^k, Ga^k, g(y^k) \rangle. \end{aligned}$$

## 4 Security protocols abstractions

We introduce our security protocol abstractions and illustrate their usefulness on our running example. We will present two types of protocol abstractions:

**Type-based abstractions** allow us to remove fields with atomic or hash types from messages and to remove or split cryptographic operations. The same transformations are applied to all terms of a given type and its subtypes.

**Untyped abstractions** complement typed ones with additional simplifications: the removal of unprotected atoms and variables and of redundant subterms.

Our main results are soundness theorems for these abstractions. They ensure that any attack on a given property of the original protocol translates to an attack on a related property in the abstracted protocol. As we will explain, these results only hold under certain conditions on the protocol and the property. Here, we will mainly focus on type-based abstractions, but we will also briefly introduce the untyped ones (for more details, see Appendixes I to K).

### 4.1 Type-based protocol abstractions

Our type-based abstractions are specified by a list of recursive equations subject to some conditions on their shape. We define their semantics in terms of a simple Haskell-style functional program. We use both pattern matching on terms and subtyping on types to select the equation to be applied to a given term. This ensures that terms of related types are transformed in a uniform manner.

In order to remove variables and atomic terms, we map them to the special constant `nil`. We rely on a normalization function  $nf$  to remove the resulting `nil`-subterms. For example,  $nf(\langle t, \text{nil} \rangle) = nf(t)$ ,  $nf(\{\text{nil}\}_k) = \text{nil}$ ,  $nf(\{t\}_{\text{nil}}) = nf(t)$ , and  $nf(t) = t$  for variables and atoms  $t$ . A formal definition is given in Appendix A.2. We say that a term  $t$  is in *normal form* if  $nf(t) = t$ . A substitution  $\sigma$  is in normal form if all elements of  $\text{ran}(\sigma)$  are. We also use  $nf$  on types, where it removes  $\gamma_{\text{nil}}$  instead of `nil`.

**Syntax** We define the set  $\mathcal{P} = \mathcal{T}(\mathcal{V}_{pt}, \emptyset)$  of *patterns*, where  $\mathcal{V}_{pt}$  is a set of *pattern variables* distinct from  $\mathcal{V}$ . A pattern  $p \in \mathcal{P}$  is called *linear* if each (pattern) variable occurs at most once in  $p$ . We use (nil-free) pattern typing environments  $\Gamma : \mathcal{V}_{pt} \rightarrow \mathcal{Y}$  to assign (arbitrary) types to pattern variables and extend them to all patterns in the same way as we did for terms. Our type-based message abstractions are instances of the following recursive function specifications.

**Definition 3.** A function specification  $F_f = (f, \Gamma_f, E_f)$  consists of an unary function symbol  $f \notin \Sigma^1$ , a pattern typing environment  $\Gamma_f$ , and

$$E_f = [f(p_1) = u_1, \dots, f(p_n) = u_n],$$

a list of equations where each  $p_i \in \mathcal{P}$  is a linear pattern such that  $\text{vars}(p_i) \subseteq \text{dom}(\Gamma_f)$  and  $u_i \in \mathcal{T}_{\Sigma^1 \cup \Sigma^2 \cup \{f\}}(\text{vars}(p_i)) \cup \{\text{Nil}\}$  for all  $i \in \{1, \dots, n\}$ , i.e.,  $u_i$  is either built from the pattern variables in  $p_i$ , message constructors, and the function symbol  $f$ , or equals the special variable `Nil`  $\in \mathcal{V}$ .

To ensure the *uniform* abstraction of terms of a type  $\tau$  and its subtypes (e.g., a term  $t$  and its well-typed instance  $t\theta$ ), we require pairwise disjoint patterns.

**Definition 4.** We say that a function specification  $F_f = (f, \Gamma_f, E_f)$  is pattern-disjoint if the types in the set  $\Pi_f = \Gamma_f(\{p_1, \dots, p_n\})$  are pairwise disjoint, i.e., for all  $i, j \in \{1, \dots, n\}$  such that  $i \neq j$ , we have  $\{\Gamma_f(p_i)\} \downarrow \cap \{\Gamma_f(p_j)\} \downarrow = \emptyset$ .

We use vectors (lists of terms)  $\vec{t} = [t_1, \dots, t_n]$ , assuming  $n > 0$  is clear from the context. We define  $set(\vec{t}) = \{t_1, \dots, t_n\}$  and  $\widehat{f}(\vec{t}) = \langle f(t_1), \dots, f(t_n) \rangle$ , the elementwise application of a function  $f$  to a vector where the result is converted to a tuple (with the convention  $\langle t \rangle = t$ ). We extend *split* to vectors by  $split(\vec{t}) = split(set(\vec{t}))$ . We write  $\{\cdot\}$  for either  $\{\cdot\}$  or  $\{\cdot\}$ . and  $g^a(t)$  for the  $a$ -fold application of a function  $g$  to its argument  $t$  where  $g^0(t) = t$ . Below we use this notation with  $g \in \Sigma_u$  and  $g = \{\cdot\}_k$  for a fixed key  $k$ .

**Definition 5 (Type-based abstraction).** A pattern-disjoint function specification  $F_f = (f, \Gamma_f, E_f)$  is a type-based abstraction if each equation in  $E_f$  has one of the following shapes:

- (i)  $f(p) = u$  where  $p \in \mathcal{V}_{pt}$ ,  $u \in \{p, Nil\}$ , and  $\Gamma_f(p) \in \{\alpha, msg\}$  implies  $u = p$ ,
- (ii)  $f(g(q)) = g(f(q))$  where  $q \in \mathcal{V}_{pt}$  and  $g \in \{\mathbf{pk}, \mathbf{pri}\}$ ,
- (iii)  $f(\mathbf{sh}(q, r)) = \mathbf{sh}(f(q), f(r))$  where  $\{q, r\} \subseteq \mathcal{V}_{pt}$ ,
- (iv)  $f(h(q)) = \langle h^{a_1}(\widehat{f}(\overline{q_1})), \dots, h^{a_l}(\widehat{f}(\overline{q_l})) \rangle$  for  $h \in \Sigma_u$ ,  $l > 0$  and vectors  $\overline{q_1}, \dots, \overline{q_l}$  such that  $a_i \geq 0$  and  $set(\overline{q_i}) \subseteq split(q)$  for  $i \in \{1, \dots, l\}$ , or  $f(h(q)) = Nil$ ,
- (v)  $f(\langle q, r \rangle) = \widehat{f}(\vec{t})$  where  $set(\vec{t}) = split(\langle q, r \rangle)$ , or
- (vi)  $f(\{\!|q|\!\}_\tau) = \langle \{\!|\widehat{f}(\overline{q_1})|\!\}_{\widehat{f}(\overline{r_1})}^{a_1}, \dots, \{\!|\widehat{f}(\overline{q_d})|\!\}_{\widehat{f}(\overline{r_d})}^{a_d} \rangle$ , for some  $d > 0$ , and vectors  $\overline{q_1}, \dots, \overline{q_d}$ ,  $\overline{r_1}, \dots, \overline{r_d}$  such that  $\bigcup_{i=1}^d set(\overline{q_i}) = split(q)$  and, for all  $i \in \{1, \dots, d\}$ ,  $a_i \geq 0$  and  $set(\overline{r_i}) \subseteq split(r)$ .

Conditions (i)-(vi) constrain the shapes of the transformed terms. Intuitively speaking, the abstractions can only weaken the cryptographic protection of terms, but never strengthen it. Concretely, terms of atomic types are either mapped to *Nil* for removal or kept unchanged (i). We forbid the removal of all terms of types *msg* and  $\alpha$ . Such abstractions are too coarse and make little sense. For keys, the abstractions are homomorphic (ii, iii). For hash functions  $h \in \Sigma_u$ , we have a choice of splitting, adding, or removing applications of  $h$  or removing the hash term completely by mapping it to *Nil* (iv). For (possibly nested) pairs, we require that each component is transformed individually (v). Similar to hashes, we can split, add, or remove encryptions or signatures provided all plaintext tuple components are still present in the transformed form (vi). If the key is a tuple we may also split it or remove some of its components.

*Example 3.* Let  $F_f = (f, \Gamma_f, E_f)$  be the type-based abstraction with environment  $\Gamma_f = [X_1 : \gamma_c, X_2 : \beta_n, X, Y, Z, U, V : msg]$  where  $E_f$  consists of the equations  $f(X_1) = Nil$  and  $f(X_2) = X_2$ , followed by the following equations:

$$\begin{aligned} f(\langle X, Y, Z \rangle) &= \langle f(Y), f(X), f(Z) \rangle \\ f(h(X, Y, U, V)) &= \langle h(f(X), f(Y)), h(f(U)) \rangle \\ f(\{\!|X, Y, Z|\!\}_{\langle U, V \rangle}) &= \langle \{\!|f(X), f(Y)|\!\}_{f(U)}, f(Y), \{\!|f(Z)|\!\}_{f(V)} \rangle \end{aligned}$$

```

1  fun f(t, Γ, ℓ) = frec(nfℓ(t)) where
2  fun frec(t) = case t of
3    |(f(p)=u)∈Ef+ p | Γ(t) ≲ Γf+(p) ⇒ nfℓ(u[frec/f, ℓ/Nil])

```

**Program 1.** Functional program  $f$  resulting from  $F_f = (f, \Gamma_f, E_f)$ .

Clearly,  $(f, \Gamma_f, E_f)$  is pattern-disjoint. The first and second clause respectively remove and preserve atoms and variables of type  $\gamma_c$  and  $\beta_n$ . The third clause swaps the first two fields in  $n$ -tuples for  $n \geq 3$ . The fourth one splits a hash into two hashes, dropping the field  $V$ . The last clause splits the plaintext and key of a symmetric encryption:  $f(X)$  and  $f(Y)$  are encrypted with the key  $f(U)$ ,  $f(Y)$  is pulled out of the encryption, and  $f(Z)$  is separately encrypted with  $f(V)$ . ♠

**Semantics** The semantics of a type-based abstraction  $F_f$  is given by the Haskell-style functional program  $f$  (Program 1).<sup>1</sup> We can apply the same function  $f$  to terms and types. For terms  $t \in \mathcal{T}$ , we use  $f(t, \Gamma, \text{nil})$  and assume  $\Gamma(t)$  is defined, i.e.,  $\text{vars}(t) \subseteq \text{dom}(\Gamma)$ <sup>b</sup>. For types  $\tau \in \mathcal{Y}$ , we use  $f(\tau, \Gamma, \gamma_{\text{nil}})$  and set  $\Gamma(\tau) = \tau$  by convention. We write  $f(t)$  or  $f(\tau)$  for short if no confusion is possible.

The function  $f(t, \Gamma, \ell)$  calls the recursive function  $f^{\text{rec}}(nf_\ell(t))$  on the normalized term  $nf_\ell(t)$ . To ensure the totality of  $f$ , we use the extended function specification  $F_f^+ = (f, \Gamma_f^+, E_f^+) = (f, \Gamma_f \cup \Gamma_f^0, E_f \cdot E_f^0)$ , where  $\Gamma_f^0 = [Z_1, Z_2 : \text{msg}]$  and  $E_f^0$  has a clause  $f(g(Z_1, \dots, Z_n)) = g(f(Z_1), \dots, f(Z_n))$  for each  $g \in \Sigma^1 \cup \Sigma^2$  and the final clause  $f(Z_1) = Z_1$ . We assume  $\Gamma_f$  and  $\Gamma_f^0$  have disjoint domains (and rename variables otherwise). The **case** statement has a clause

$$p \mid \Gamma(t) \preceq \Gamma_f^+(p) \Rightarrow nf_\ell(u[f^{\text{rec}}/f, \ell/\text{Nil}])$$

for each equation  $f(p) = u$  of  $E_f^+$ . Such a clause is enabled if (1) the term  $t$  matches the pattern  $p$ , i.e.,  $t = p\theta$  for some substitution  $\theta$ , and (2) its type  $\Gamma(t)$  is a subtype of  $\Gamma_f^+(p)$ . The first enabled clause is executed. Hence, the equations  $E_f^0$  serve as fall-back clauses, which cover the terms not handled by  $E_f$ .

Next, we extend type-based abstractions to events, roles and protocols. Note that transformed events with nil arguments are removed from the roles.

**Definition 6 (Type-based protocol abstractions).** Let  $F_f = (f, \Gamma_f, E_f)$  be a type-based abstraction. We define

- (i)  $f(\text{ev}(t)) = \text{ev}(f(t))$  for events  $\text{ev}(t) \in \text{Evt}(\mathcal{T})$ .
- (ii) for event sequences,  $f(\epsilon) = \epsilon$  and  $f(e \cdot tl) = f(tl)$  if  $\text{term}(f(e)) = \text{nil}$  and  $f(e \cdot tl) = f(e) \cdot f(tl)$  otherwise; the lifting to traces is defined analogously.
- (iii)  $f(\rho) = \{(X, f(d)) \mid (X, d) \in \rho \wedge f(X) \neq \text{nil}\}$  for maps  $\rho : \mathcal{V} \rightarrow \mathcal{D}$  such as substitutions, typing environments, and the protocol's role mapping.
- (iv)  $f(P) = (f(\Gamma_P), f(S_P))$  for protocols  $P = (\Gamma_P, S_P)$ .

We also extend abstractions  $f$  to formulas  $\phi \in \mathcal{L}_P$  of our property language by applying  $f$  to all terms occurring in  $\phi$  (see [28] for a formal definition).

<sup>1</sup> We are overloading the symbol  $f$  here, but no confusion should arise.

**Finding abstractions** To determine suitable abstractions, we start by identifying the terms that appear in the  $secret(\cdot)$  predicates and equations of the desired properties. Then we determine the cryptographic operations that are essential to achieve these properties and try to remove all other terms and operations. We try to avoid the introduction of new pairs of unifiable protocol terms, since these may lead to false negatives. We now apply this strategy to our example.

*Example 4 (from  $\text{IKE}_m$  to  $\text{IKE}_m^1$ ).* To preserve the secrecy of the DH key  $h(g(x), y)$  and the agreement on  $na, nb, g(x)$ , and  $g(y)$ , we have to keep either the  $MAC$  or the symmetric encryption with  $SK$  (see Examples 1 and 2). We want to remove as many other fields and operations as possible (e.g.,  $prf$  and  $KDF$ ). We choose to remove the encryption as this allows us to later remove additional fields (e.g.,  $sA2$ ) using untyped abstractions. We keep  $o$  in  $AUTHa$  to prevent unifiability with  $AUTHb$ . This leads us to the type-based abstraction  $F_1 = (f_1, \Gamma_1, E_1)$  where we define (omitting the homomorphic clauses for  $g, h$ , and  $\langle \cdot, \cdot \rangle$ )

$$E_1 = [ f_1(X) = X, f_1(X_3) = X_3, f_1(Y_3) = Y_3, f_1(\{X, Y\}_Z) = \langle f_1(X), f_1(Y) \rangle, \\ f_1(MAC(X_1, \dots, X_8)) = MAC(f_1(X_1), f_1(X_3), f_1(X_5), \dots, f_1(X_8)), \\ f_1(MAC(Y_1, \dots, Y_8)) = MAC(f_1(Y_1), f_1(Y_5), \dots, f_1(Y_8)), \\ f_1(KDF(Z_1, \dots, Z_5)) = f_1(Z_3), f_1(prf(U, Z)) = f_1(U), \dots ]$$

and  $\Gamma_1 = [X : \alpha, X_3 : \gamma_o, Y_3 : nonce, Z_3 : h(msg), U : KDF(msg), \dots]$  where all remaining pattern variables are of type  $msg$ . Clearly,  $F_1$  is pattern-disjoint. The types of  $Z_3$  and  $U$  are needed to satisfy soundness conditions (Theorem 3). Applying  $f_1$  to  $\text{IKE}_m$  we obtain  $\text{IKE}_m^1$ . Here is the abstracted initiator role.

$$S_{\text{IKE}_m^1}(A) = \text{snd}(sPIa, o, sA1, g(x), na) \cdot \text{rcv}(sPIa, sPIb, sA1, Gb, Nb) \cdot \\ \text{snd}(sPIa, sPIb, A, B, AUTHaa, sA2, tSa, tSb) \cdot \\ \text{rcv}(sPIa, sPIb, B, AUTHba, sA2, tSa, tSb)$$

where  $SKa = h(Gb, x)$ ,  $AUTHaa = MAC(\text{sh}(A, B), o, g(x), na, Nb, SKa)$ , and  $AUTHba = MAC(\text{sh}(A, B), Gb, Nb, na, SKa)$ . In a second step, we will remove most fields in the roles of  $\text{IKE}_m^1$  using untyped abstractions. ♠

## 4.2 Soundness of type-based abstractions

To justify the soundness of our abstractions, we show that any attack on a property  $\phi$  of the original protocol  $P$  is reflected as an attack on the property  $f(\phi)$  of the abstracted protocol  $f(P)$ . We decompose this into reachability preservation (RP) and an attack preservation (AP) as follows. We require that, for all reachable states  $(tr, th, \sigma)$  of  $P$ , there is a ground substitution  $\sigma'$  such that

- (RP)  $(f(tr), f(th), \sigma')$  is a reachable state of  $f(P)$ , and
- (AP)  $(tr, th, \sigma) \not\models \phi$  implies  $(f(tr), f(th), \sigma') \not\models f(\phi)$ .

These properties will require some assumptions about the protocol  $P$ , the formula  $\phi$ , and the abstraction  $f$ . Before we formally state the soundness theorem, we will introduce and motivate these assumptions while sketching its proof. Detailed proofs of the results in this section are given in Appendixes B to G. For the remainder of this subsection we consider arbitrary but fixed  $P, \phi$ , and  $F_f$ .

**Reachability preservation (RP)** To achieve reachability preservation, we prove that every step of  $P$  can be simulated by a corresponding step of  $f(P)$ . In particular, to simulate receive events, we show that intruder deducibility is preserved under abstractions  $f$  (cf. second premise of rule  $RECV$ ), i.e.,

$$T\theta, IK_0 \vdash u\theta \Rightarrow \exists \theta'. f(T)\theta', f(IK_0) \vdash f(u)\theta'. \quad (1)$$

This property is also required to show the preservation of attacks on secrecy as part of (AP). We first establish deducibility preservation for ground terms:

**Theorem 1 (Deducibility preservation).** *Let  $T \cup \{t\} \subseteq \mathcal{N}$  be a set of ground network messages. Then  $T \vdash t$  implies  $f(T), \text{nil} \vdash f(t)$ .*

In order to extend this result to (1), we prove a *substitution property*, which ensures that  $f$  distributes over  $t\theta$  (modulo normalization). This property requires some assumptions about  $t$  and  $\theta$  in order to guarantee a uniform abstraction of  $t$  and  $t\theta$ . Therefore, we define the set of terms and types that  $F_f$  transforms using only the clauses of  $E_f$ , without recourse to the fall-back clauses of  $E_f^0$ .

**Definition 7 (Uniformity).** *Let  $\text{Rec}(F_f, t)$  be the set of terms (types)  $u$  such that  $f^{\text{rec}}(u)$  is called in the computation of  $f(t)$ . We define the uniform domain of  $F_f$  under  $\Gamma$  by  $\text{udom}(F_f, \Gamma) = \{t \in \mathcal{T} \cup \mathcal{Y} \mid \Gamma(\text{Rec}(F_f, t)) \subseteq \Pi_f \downarrow \cup \{\text{msg}\}\}$ .*

One assumption of the substitution property is that  $t \in \text{udom}(F_f, \Gamma)$ . A second assumption requires that  $\theta$  is a *nil-free* substitution, i.e.,  $\text{nil} \notin \text{subs}(\text{ran}(\theta))$ .

**Theorem 2 (Substitution property).** *Let  $\theta$  be a nil-free substitution that is well-typed with respect to a typing environment  $\Gamma$ . Then, for all terms  $t$  such that  $t \in \text{udom}(F_f, \Gamma)$ , we have that  $f(t\theta) = \text{nf}(f(t)f(\theta))$ .*

In the strictly typed setting without message variables of [28], we proved the simpler substitution property  $f(t\theta) = f(t)f(\theta)$ . In combination with Theorem 1, (1) follows directly for  $\theta' = f(\theta)$ . Appendix A.3 contains three examples showing respectively the necessity of uniformity and normalization in Theorem 2 and of additional assumptions to establish (1) in the presence of message variables. Hence, the combination of Theorems 1 and 2 only yields implication (2) below.

$$T\theta, IK_0 \vdash u\theta \Rightarrow \text{nf}(f(T)f(\theta)), f(IK_0) \vdash \text{nf}(f(u)f(\theta)) \quad (2)$$

$$\Rightarrow f(T)f(\theta), f(IK_0) \vdash f(u)f(\theta) \quad (3)$$

We bridge the gap to property (1) by proving the implication (3). For nil-free (protocol) terms  $T$  and  $u$  and nil-free  $\theta$ , this holds trivially if  $F_f$  is also nil-free, i.e., none of its equations contains  $\text{Nil}$ . Otherwise, we can prove it provided that all message variables in  $T$  and  $u$  are clear. A term  $u$  is *clear* in  $t$  if  $u$  does not occur in  $t$  under any constructor other than pairing, i.e.,  $u \notin \text{subs}(t) \setminus \text{split}(t)$ .

**Definition 8 (msg-clearness).** *A term  $t$  is msg-clear under  $\Gamma$  if all its message variables are clear in  $t$ . A protocol  $P = (\Gamma_P, S_P)$  is msg-clear if all terms in  $\mathcal{M}_P$  are msg-clear under  $\Gamma_P$ .*

Summarizing and slightly extending the exposition above, we can prove reachability preservation (RP) under the assumptions

- $\mathcal{M}_P \cup \text{dom}(\Gamma_P) \subseteq \text{udom}(F_f, \Gamma_P)$  (for Theorem 2), and
- $F_f$  is nil-free or  $f(P)$  is *msg-clear* (for implication (3) above).

Both assumptions are easily checked in linear time (in the size of  $P$ ). The second assumption incurs a quite strong limitation, which is however mitigated by the complementary untyped abstractions as we will further discuss in Section 4.3. The abstraction  $F_1$  in Example 4 is nil-free and it is easy to check that it also satisfies the first assumption with respect to the protocol  $\text{IKE}_m$ .

**Attack preservation (AP)** We now define conditions on formulas that we need to establish attack preservation and then explain the role of each of them. We start by introducing some auxiliary notions.

- $\text{Sec}_\phi$  is the set of all terms  $t$  that occur in formulas  $\text{secret}(t)$  in  $\phi$ ,
- $\text{Eq}_\phi$  is the set of pairs  $(t, u)$  such that the equation  $t = u$  occurs in  $\phi$  and  $\text{EqTerm}_\phi = \pi_1(\text{Eq}_\phi) \cup \pi_2(\text{Eq}_\phi)$  is the set of underlying terms, and
- $\text{Evt}_\phi$  is the set of events occurring in  $\phi$ .

We denote by  $\text{Eq}_\phi^+$  and  $\text{Eq}_\phi^-$  the positive and negated occurrences of equations in  $\phi$  and similarly for  $\text{Evt}_\phi$ .

**Definition 9 (( $P, f$ )-safe formulas).** *A formula  $\phi \in \mathcal{L}_P$  is ( $P, f$ )-safe if for all nil-free ground substitutions  $\sigma$  that are well-typed with respect to  $\Gamma_P$ , we have*

- (i)  $\text{Sec}_\phi \cup \text{EqTerm}_\phi \subseteq \text{udom}(F_f, \Gamma_P)$ ,
- (ii)  $f(\text{Sec}_\phi)$  is *msg-clear* under  $\Gamma_P$  or  $F_f$  is nil-free,
- (iii)  $f(t\sigma) = f(u\sigma)$  implies  $t\sigma = u\sigma$  for all  $(t, u) \in \text{Eq}_\phi^+$ ,
- (iv)  $f(t\sigma) = f(u\sigma)$  implies  $f(t)f(\sigma) = f(u)f(\sigma)$  for all  $(t, u) \in \text{Eq}_\phi^-$ ,
- (v)  $f(t) = f(u)$  implies  $t = u$ , for all  $e(t) \in \text{Evt}_\phi^+$  and  $e(u) \in \text{Evt}(\mathcal{M}_P)$ ,
- (vi)  $f(t) \neq \text{nil}$  for all  $e(t) \in \text{Evt}_\phi$ .

Condition (i) requires that  $F_f$  is uniform for the terms in secrecy statements and equalities. This is needed to apply the substitution property. Condition (ii) requires the secret terms in the transformed property are *msg-clear* unless the abstraction is nil-free. Conditions (i) and (ii) allow us to establish attack preservation for secrecy. Conditions (iii) and (iv) are required to preserve attacks on equalities and disequalities. Condition (iii) expresses injectivity of the abstraction on the terms in positively occurring equalities. This is required, for instance, to preserve attacks on agreement properties. Condition (iv) appears to be less relevant in practice. In Appendix H, we provide syntactic criteria to check conditions (iii) and (iv) that avoid the universal quantification over substitutions. Conditions (v) and (vi) are required for properties involving event orderings and *steps* predicates. Condition (v) states that the abstraction must not identify an event occurring positively in the property with a protocol event. Finally, condition (vi) disallows the removal of event terms in the property.

We can now state the soundness theorem. Below,  $IK_0$  and  $IK'_0$  respectively denote the intruder’s initial knowledge associated with  $P$  and  $f(P)$ .

**Theorem 3 (Soundness).** *Suppose  $P$ ,  $\phi$ , and  $F_f$  satisfy:*

- (i)  $IK'_0$  is in normal form and  $f(IK_0) \subseteq IK'_0$ ,
- (ii)  $\mathcal{M}_P \cup \text{dom}(\Gamma_P) \subseteq \text{udom}(F_f, \Gamma)$ ,
- (iii)  $f(P)$  is *msg-clear* or  $F_f$  is *nil-free*, and
- (iv)  $\phi$  is  $(P, f)$ -safe.

*Then, for all states  $(tr, th, \sigma)$  reachable in  $P$ , there is a nil-free ground substitution  $\sigma'$  such that*

1.  $(f(tr), f(th), \sigma')$  is a reachable state of  $f(P)$ , and
2.  $(tr, th, \sigma) \not\equiv \phi$  implies  $(f(tr), f(th), \sigma') \not\equiv f(\phi)$ .

### 4.3 Untyped abstractions

Type-based abstractions offer a wide range of possibilities to transform cryptographic operations (e.g., removal, splitting, pulling fields outside). However, the *msg-clearness* requirement limits their capabilities to remove variables, atoms, and hashed messages. Moreover, type-based abstractions cannot in general remove message variables or redundant terms that the intruder can reconstruct such as tuple fields that are duplicated or otherwise known to the intruder.

Therefore, we complement type-based abstractions with two kinds of *untyped abstractions*, which allow us to remove (1) redundancy in the form of intruder-derivable subterms, and (2) clear atoms and variables (of any type). Here we content ourselves with an example and refer to Appendixes I to K for the details.

*Example 5* ( $\text{IKE}_m^1$  to  $\text{IKE}_m^2$ ). First, we recall the specification of role  $A$  of  $\text{IKE}_m^1$ .

$$\begin{aligned} S_{\text{IKE}_m^1}(A) = & \text{snd}(sPIa, o, sA1, g(x), na) \cdot \text{rcv}(sPIa, SPIb, sA1, Gb, Nb) \cdot \\ & \text{snd}(sPIa, SPIb, A, B, AUTHaa, sA2, tSa, tSb) \cdot \\ & \text{rcv}(sPIa, SPIb, B, AUTHba, sA2, tSa, tSb) \end{aligned}$$

We compose two untyped abstractions. The first one removes redundancies, namely the intruder-derivable role names  $A$  and  $B$  and the constants  $o$ ,  $sA1$ ,  $sA2$ ,  $tSa$ , and  $tSb$ . The second abstraction removes the fresh values  $sPIa$  and  $SPIb$  and the variables  $SPIa$  and  $SPIb$ , which the intruder can replace with his own values of the same type. The result is the protocol  $\text{IKE}_m^2$  whose initiator role is defined as follows.

$$S_{\text{IKE}_m^2}(A) = \text{snd}(g(x), na) \cdot \text{rcv}(Gb, Nb) \cdot \text{snd}(AUTHaa) \cdot \text{rcv}(AUTHba)$$

We also abstract the properties  $\phi_s$  and  $\phi_a$  from Example 2 in two steps. These only affect the events in the *steps* predicates. In Appendix A.4, we justify the soundness of the first, type-based abstraction step. The second, untyped abstraction step also satisfies the associated soundness conditions. ♠

protocol/prop./#threads	S	A	W	N	3	4	5	6	7	8	$\infty$
IKEv1-pk2-a2	•			•	44.09	310.45	2021.13	12484.88	TO	TO	TO
IKEv1-pk2-a2-abs1	•			•	3.83	16.11	89.30	593.11	3666.50	25441.00	TO
IKEv1-pk-a22	•			•	15.90	83.36	250.33	550.72	1003.73	1731.87	TO
IKEv1-pk-a22-abs1	•			•	0.56	0.98	1.39	2.27	3.66	5.75	TO
IKEv2-eap	•			•	TO	TO	TO	TO	TO	TO	TO
IKEv2-eap-abs2	•			•	22.47	268.86	1894.31	8003.61	25065.15	TO	TO
IKEv2-mac	•			•	1.85	4.91	6.72	8.07	8.42	8.49	8.70
IKEv2-mac-abs1	•			•	0.50	0.71	0.76	0.78	0.81	0.83	0.85
IKEv2-mactosig	•			•	12.17	141.37	1075.46	7440.81	TO	TO	TO
IKEv2-mactosig-abs1	•			•	1.70	5.08	11.33	13.14	18.93	19.23	19.81
IKEv2-sigtomac	•			•	6.22	26.46	66.05	115.9	172.09	212.43	238.43
IKEv2-sigtomac-abs1	•			•	6.20	23.20	49.02	68.68	77.94	80.44	77.83
ISOIEC 9798-2-5	•			•	0.84	10.67	75.64	582.03	4409.73	TO	TO
ISOIEC 9798-2-5-abs2	•			•	0.08	0.15	0.23	0.31	0.28	0.28	0.26
ISOIEC 9798-2-6	•			•	0.60	3.83	18.44	83.85	198.93	488.29	21478.13
ISOIEC 9798-2-6-abs1	•			•	0.28	1.63	7.19	26.69	66.15	161.63	2225.92
ISOIEC 9798-3-6-1		•		•	42.64	803.29	9007.68	MO	MO	MO	MO
ISOIEC 9798-3-6-1-abs2		•		•	0.08	0.12	0.15	0.14	0.14	0.13	0.13
ISOIEC 9798-3-6-2		•		•	3.51	47.09	53.09	46.08	48.05	59.74	69.07
ISOIEC 9798-3-6-2-abs2		•		•	0.11	0.11	0.10	0.10	0.11	0.11	0.11
ISOIEC 9798-3-7-1		•		•	40.19	742.82	7547.40	15893.67	MO	MO	MO
ISOIEC 9798-3-7-1-abs2		•		•	0.07	0.10	0.10	0.10	0.10	0.10	0.10
ISOIEC 9798-3-7-2		•		•	2.41	7.82	17.30	26.21	37.40	50.83	TO
ISOIEC 9798-3-7-2-abs2		•		•	0.05	0.05	0.05	0.05	0.05	0.05	0.05
PANA-AKA	•	•	•	•	5755.68	TO	TO	TO	TO	TO	TO
PANA-AKA-abs1	•	•	•	•	0.24	0.24	0.24	0.25	0.24	0.23	0.23

**Table 1.** Experimental verification results. The time is in seconds. Properties of interest are **S**ecrecy, **A**liveness, **W**eak agreement, and **N**on-injective agreement.

## 5 Experimental results

We have validated the effectiveness of our abstractions on 12 members of the IKE and ISO/IEC 9798 protocol families and on the PANA-AKA protocol [3]. We verify these protocols using four tools based on three different techniques, namely, Scyther [14], CL-Atse [31], OFMC [8], and SATMC [5]. Only Scyther supports verification for an unbounded number of threads. We present here a selection of the experimental results for Scyther (Table 1) and refer to Appendix L for a complete account, including results for other tools. Our models of the IKE and ISO/IEC 9798 protocols are based on Cremers’ [12,13]. The abstracted protocols’ names end in ‘abs $n$ ’ where  $n$  is the abstraction level. We have run the experiments on ETH Zurich’s High Performance Cluster [1] using up to 32 cores of a number of 12-core AMD Opteron 6174 processors with 64 GB of RAM.

For 9 of the 13 original protocols, an unbounded ( $\infty$ ) verification attempt results in a timeout (TO, set to 8h cpu time) or memory exhaustion (MO). In 5 of these cases our abstractions enabled a verification in less than 0.3 seconds and in one case in under 20 seconds. However, for the first three protocols, we still get a timeout for the abstractions. For the large majority of the bounded verification tasks, we significantly push the bound on the number of threads or we achieve massive speedups. For example, our abstractions enable the verification

of the complex nested protocols IKEv2-eap and PANA-AKA. Scyther verifies an abstraction of IKEv2-eap for up to 7 threads where it fails on the original for 3 threads. More strikingly, it completes an unbounded verification of the simplified PANA-AKA in less than 1/4 second whereas it failed on the original for 4 threads. We also achieve dramatic speedups for many other protocols, most notably for the IKEv1-pk-a22 and IKEv2-mactosig protocols and for the ISO/IEC 9798 variants 2-5, 3-7-1, and 3-7-2. Moreover, the verification time for many abstracted protocols increases much slower than for their originals. For instance, we obtain almost constant verification times for the last five protocols, whereas this time can be exponentially increasing on the originals, e.g., for ISOIEC 9798-3-7-1.

However, for a few protocols, the speedup is more modest: the verification time drops from about 4 to 1.3 minutes for the IKEv2-sigtomac protocol and from about 6 hours to 37 minutes for ISO/IEC 9798-2-6 protocol.

## 6 Related work and conclusions

Hui and Lowe [22] define several kinds of abstractions similar to ours with the aim of improving the performance of the CASPER/FDR verifier. They establish soundness only for ground messages and encryption with atomic keys. We work in a more general model, cover additional properties, and treat the non-trivial issue of abstracting the open terms in protocol specifications. Other works [30,17,16] also propose a set of syntactic transformations, however without formally establishing their soundness. Using our results, we can, for instance, justify the soundness of the refinements in [17, Section 3.3]. Guttman [20,19] studies the preservation of security properties for 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 such step explains the origin of a message. He does not have a syntactic soundness check.

In this work, we propose a set of syntactic protocol transformations that allows us to abstract realistic protocols and capture a large class of attacks. Unlike previous work in this area [28,22], our theory and soundness results accommodate untyped variables, user-defined types, and subtyping. These features allow us to accurately model protocols and to capture type-flaw attacks. Methodologically, the combination of abstractions in our running example is typical for protocols with message variables: we first use a type-based abstraction to pull out the fields to be removed to the top-level and then remove them using the untyped ones. For protocols without message variables, type-based abstractions are usually sufficient, since the *msg*-clearness condition holds trivially. Our experiments clearly show that modern protocol verifiers can substantially benefit from our abstractions, which in many cases either enable the completion of verification tasks that have previously hit a resource limit or lead to dramatic speedups.

We plan to implement our technique in a tool. We envision an automatic abstraction-refinement process using heuristics based on an analysis of the protocol's term structure and its desired properties. We also intend to support equational theories and advanced properties such as perfect forward secrecy.

**Acknowledgements** We thank Mathieu Turuani and Michael Rusinowitch for our intense discussions on the subject of this paper and David Basin, Ognjen Maric, and Ralf Sasse for their careful proof-reading. 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.

## References

1. Swiss National Computing Centre: Brutus cluster. <http://www.cscs.ch>.
2. M. Arapinis and M. Dufлот. 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. J. Arkko and H. Haverinen. RFC 4187: Extensible Authentication Protocol Method for 3rd Generation Authentication and Key Agreement (EAP-AKA), 2006. <http://www.ietf.org/rfc/rfc4187>.
4. A. Armando, W. Arzac, T. Avanesov, M. Barletta, A. Calvi, A. Cappai, R. Carbone, Y. Chevalier, L. Compagna, J. Cuéllar, G. Erzse, S. Frau, M. Minea, S. Mödersheim, D. von Oheimb, G. Pellegrino, S. E. Ponta, M. Rocchetto, M. Rusinowitch, M. T. Dashti, M. Turuani, and L. Viganò. The AVANTSSAR platform for the automated validation of trust and security of service-oriented architectures. In C. Flanagan and B. König, editors, *TACAS*, volume 7214 of *Lecture Notes in Computer Science*, pages 267–282. Springer, 2012.
5. A. Armando and L. Compagna. SATMC: A SAT-based model checker for security protocols. In J. J. Alferes and J. A. Leite, editors, *JELIA*, volume 3229 of *Lecture Notes in Computer Science*, pages 730–733. Springer, 2004.
6. A. Armando and L. Compagna. SAT-based model-checking for security protocols analysis. *International Journal of Information Security*, 7(1):3–32, 2008.
7. D. Basin, C. Cremers, and C. Meadows. Model checking security protocols. In E. Clarke, T. Henzinger, and H. Veith, editors, *Handbook of Model Checking*, chapter 24. Springer. To appear.
8. D. A. Basin, S. Mödersheim, and L. Viganò. OFMC: A symbolic model checker for security protocols. *Int. J. Inf. Sec.*, 4(3):181–208, 2005.
9. B. Blanchet. An efficient cryptographic protocol verifier based on Prolog rules. In *CSFW*, pages 82–96. IEEE Computer Society, 2001.
10. B. Blanchet. Security protocols: from linear to classical logic by abstract interpretation. *Inf. Process. Lett.*, 95(5):473–479, 2005.
11. P. Cousot and R. Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In R. M. Graham, M. A. Harrison, and R. Sethi, editors, *POPL*, pages 238–252. ACM, 1977.
12. C. Cremers. IKEv1 and IKEv2 protocol suites, 2011. <https://github.com/cascremers/scyther/tree/master/gui/Protocols/IKE>.
13. C. Cremers. ISO/IEC 9798 authentication protocols, 2012. <https://github.com/cascremers/scyther/tree/master/gui/Protocols/ISO-9798>.
14. C. J. F. Cremers. The Scyther tool: Verification, falsification, and analysis of security protocols. In A. Gupta and S. Malik, editors, *CAV*, volume 5123 of *Lecture Notes in Computer Science*, pages 414–418. Springer, 2008.
15. C. J. F. Cremers. Key exchange in IPsec revisited: Formal analysis of IKEv1 and IKEv2. In V. Atluri and C. Díaz, editors, *ESORICS*, volume 6879 of *Lecture Notes in Computer Science*, pages 315–334. Springer, 2011.

16. 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.
17. A. Datta, A. Derek, J. C. Mitchell, and D. Pavlovic. A derivation system and composition logic for security protocols. *Journal of Computer Security*, 13:423–482, 2005.
18. S. Escobar, C. Meadows, and J. Meseguer. Maude-npa: Cryptographic protocol analysis modulo equational properties. In A. Aldini, G. Barthe, and R. Gorrieri, editors, *FOSAD*, volume 5705 of *Lecture Notes in Computer Science*, pages 1–50. Springer, 2007.
19. J. D. Guttman. Transformations between cryptographic protocols. In P. Degano and L. Viganò, editors, *ARSPA-WITS*, volume 5511 of *LNCS*, pages 107–123. Springer, 2009.
20. 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.
21. D. Harkins and D. Carrel. The Internet Key Exchange (IKE). IETF RFC 2409 (Proposed Standard), November 1998. Obsoleted by RFC 4306, updated by RFC 4109.
22. M. L. Hui and G. Lowe. Fault-preserving simplifying transformations for security protocols. *Journal of Computer Security*, 9(1/2):3–46, 2001.
23. C. Kaufman, P. Hoffman, Y. Nir, and P. Eronen. Internet Key Exchange Protocol Version 2 (IKEv2). IETF RFC 4306, December 2005. Obsoleted by RFC 5996.
24. C. Kaufman, P. Hoffman, Y. Nir, and P. Eronen. Internet Key Exchange Protocol Version 2 (IKEv2). IETF RFC 5996, September 2010.
25. S. Kent and K. Seo. Security architecture for the internet protocol. IETF RFC 4301, December 2005. Updated by RFC 6040.
26. G. Lowe. A hierarchy of authentication specifications. In *IEEE Computer Security Foundations Workshop*, pages 31–43, Los Alamitos, CA, USA, 1997. IEEE Computer Society.
27. S. Meier, B. Schmidt, C. Cremers, and D. A. Basin. The tamarin prover for the symbolic analysis of security protocols. In N. Sharygina and H. Veith, editors, *CAV*, volume 8044 of *Lecture Notes in Computer Science*, pages 696–701. Springer, 2013.
28. B. T. Nguyen and C. Sprenger. Sound security protocol transformations. In D. A. Basin and J. C. Mitchell, editors, *POST*, volume 7796 of *Lecture Notes in Computer Science*, pages 83–104. Springer, 2013.
29. L. Paulson. The inductive approach to verifying cryptographic protocols. *J. Computer Security*, 6:85–128, 1998.
30. 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.
31. M. Turuani. The CL-Atse protocol analyser. In F. Pfenning, editor, *RTA*, volume 4098 of *Lecture Notes in Computer Science*, pages 277–286. Springer, 2006.

## A Additional details

### A.1 Semantics of property language

We define the semantics of our language  $\mathcal{L}_P$ . Recall that  $\mathcal{A}_H$  denotes the set of honest agents. Let  $(tr, th, \sigma)$  be a state of the protocol  $P = (\Gamma_P, S_P)$  and let  $\vartheta$  be a substitution such that  $\text{ran}(\vartheta) \subseteq \text{dom}(th)$ . We define formula satisfaction,  $(tr, th, \sigma, \vartheta) \models \phi$ , as follows (omitting the standard cases for the boolean operators and the dual existential quantifier):

$$\begin{array}{ll}
(tr, th, \sigma, \vartheta) \models i = j & \text{iff } \vartheta(i) = \vartheta(j) \\
(tr, th, \sigma, \vartheta) \models t = u & \text{iff } t\sigma = u\sigma \\
(tr, th, \sigma, \vartheta) \models \text{role}(i, R) & \text{iff } \pi_1(th(\vartheta(i))) = R \\
(tr, th, \sigma, \vartheta) \models \text{honest}(i, R) & \text{iff } R^{\vartheta(i)}\sigma \in \mathcal{A}_H \\
(tr, th, \sigma, \vartheta) \models \text{steps}(i, e) & \text{iff } (\vartheta(i), \text{inst}_{\vartheta(i)}(e)) \in tr \\
(tr, th, \sigma, \vartheta) \models (i, e) \prec (j, e') & \text{iff } (\vartheta(i), \text{inst}_{\vartheta(i)}(e)) \prec_{tr} (\vartheta(j), \text{inst}_{\vartheta(j)}(e')) \\
(tr, th, \sigma, \vartheta) \models \text{secret}(t) & \text{iff } IK(tr)\sigma \cup IK_0 \vdash t\sigma \text{ is not derivable} \\
(tr, th, \sigma, \vartheta) \models \forall i. \phi' & \text{iff } (tr, th, \sigma, \vartheta[i \mapsto tid]) \models \phi' \text{ for all } tid \in \text{dom}(th)
\end{array}$$

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  $(tr, th, \sigma, \vartheta) \not\models \phi$  if  $(tr, th, \sigma, \vartheta) \models \phi$  does not hold. If  $\phi$  is a closed formula, we write  $(tr, th, \sigma) \models \phi$  instead of  $(tr, th, \sigma, \vartheta) \models \phi$ . We call a protocol state  $s = (tr, th, \sigma)$  an *attack on  $\phi$*  if  $s \not\models \phi$ .

### A.2 Normalization function

**Definition 10 (Normalization).** For  $t \in \mathcal{T}(V, \Sigma^0)$ , we define

$$\begin{array}{ll}
nf_\ell(t) = t & \text{if } t \in V \cup \Sigma^0 \\
nf_\ell(g(t)) = \text{if } nf_\ell(t) = \ell \text{ then } \ell \text{ else } g(nf_\ell(t)) & \text{if } g \in \Sigma^1 \\
nf_\ell(\text{sh}(t, u)) = \text{if } nf_\ell(t) = \ell \text{ or } nf_\ell(u) = \ell \text{ then } \ell \\
& \text{else } \text{sh}(nf_\ell(t), nf_\ell(u)) \\
nf_\ell(\langle t_1, t_2 \rangle) = \text{if } nf_\ell(t_1) = \ell \text{ then } nf_\ell(t_2) \\
& \text{else if } nf_\ell(t_2) = \ell \text{ then } nf_\ell(t_1) \\
& \text{else } \langle nf_\ell(t_1), nf_\ell(t_2) \rangle \\
nf_\ell(\{t\}_u) = \text{if } nf_\ell(t) = \ell \text{ then } \ell \\
& \text{else if } nf_\ell(u) = \ell \text{ then } nf_\ell(t) \\
& \text{else } \{nf_\ell(t)\}_{nf_\ell(u)}
\end{array}$$

We say that a term  $t$  is in normal form if  $nf_\ell(t) = t$ . For terms  $t \in \mathcal{T}$  and types  $\tau \in \mathcal{Y}$ , we respectively define  $nf(t) = nf_{\text{nil}}(t)$  and  $nf(\tau) = nf_{\gamma_{\text{nil}}}(\tau)$ . A substitution  $\sigma$  is in normal form if all elements in  $\text{ran}(\sigma)$  are.

### A.3 Examples for Section 4.2

Here is a simple example illustrating the need for the uniformity (Definition 7) assumption in Theorem 2.

*Example 6.* Let  $F_f = (f, \Gamma_f, E_f)$  where  $\Gamma_f = [Z : \gamma_c]$  and  $E_f = [f(h(Z)) = f(Z), f(Z) = Z]$ .  $F_f$  is clearly pattern-disjoint. Furthermore, let  $t = h(X)$  for a  $msg$ -variable  $X$  for  $h \in \Sigma_u$ , and  $\theta = [X \mapsto c]$ . Then  $f(t\theta) = f(h(c)) = c$  but  $f(t)f(\theta) = h(X)[X \mapsto c] = h(c)$ , which is already in normal form. Hence, the substitution property fails.

The following example shows the necessity of normalization in Theorem 2 in the presence of message variables.

*Example 7.* Let  $\Gamma(X) = msg$ ,  $t = \langle X, d \rangle$  and  $\theta = [X \mapsto c]$ , for constants  $c, d \in \mathcal{C}$  and let  $g$  be an abstraction that removes  $c$ , i.e.,  $g(c) = nil$ , but leaves everything else untouched. Then  $g(t\theta) = d$  but  $g(t)g(\theta) = \langle X, d \rangle[X \mapsto nil] = \langle nil, d \rangle$ . ♠

The following example shows the need for additional assumptions to establish deducibility preservation for open terms (1).

*Example 8.* Let  $F_f = (f, \Gamma_f, E_f)$  where (modulo renaming of pattern variables)  $\Gamma_f = \{Z : \beta_{n_B}\} \cup \Gamma_f^0$  and  $E_f = [f(Z) = Nil] \cdot E_f^0$ . Assume that  $t = \{n_A, n_B\}_{pk(B)}$  and  $u = \{X, Y\}_{pk(B)}$ . Let  $\Gamma = \{X : \beta_{n_A}, Y : msg\}$  and  $\theta = \{X \mapsto n_A, Y \mapsto n_B\}$ . Then  $f(\theta) = \{X \mapsto n_A, Y \mapsto nil\}$ ,  $f(t) = \{n_A\}_{pk(B)}$ , and  $f(u) = \{X, Y\}_{pk(B)}$ . Thus, we have  $t\theta \vdash u\theta$ , but there is no  $\theta'$  such that  $f(t)\theta' \vdash f(u)\theta'$ . ♠

#### A.4 Soundness conditions for $\text{IKE}_m$ -to- $\text{IKE}_m^1$ abstraction

Here, we establish the soundness conditions for the abstraction  $F_1 = (f_1, \Gamma_1, E_1)$  in Example 4 with respect to the properties  $\phi_s$  and  $\phi_a$  expressed in Example 2. We assume that  $IK_0 = IK'_0 = \mathcal{A} \cup \mathcal{C} \cup \mathcal{F}^\bullet \cup \text{sh}(\mathcal{A}_C, \mathcal{A}) \cup \text{sh}(\mathcal{A}, \mathcal{A}_C)$ . Our argument relies on the following observations.

- (O1)  $f_1(IK_0) = IK_0 = IK'_0$ .
- (O2)  $F_1$  is nil-free.
- (O3) All subterms of protocol terms are abstracted using only clauses in  $E_1$ .
- (O4) For all atoms  $a$ ,  $g(a)$  is the only term that is mapped to  $g(a)$  by  $f_1$ .

By (O1), (O2), and (O3), the first three conditions required for Theorem 3 are fulfilled. It remains to verify the conditions (i)-(vi) in Definition 9. Condition (i) follows from (O3) and the fact that the terms in  $\text{Sec}_\phi \cup \text{EqTerm}_\phi$  are subterms of protocol terms. Conditions (iv) and (v) are trivially satisfied as equations only appear positively and events only negated in  $\phi_s$  and  $\phi_a$ . Likewise, condition (ii) holds by (O2). It is also easy to see that (vi) is satisfied. To justify condition (iii), note that we can rewrite the equality on the tuples in  $\phi_a$  as a conjunction of equalities on the tuples' components. By (O2),  $f_1$  is the identity on atoms and variables. Therefore, it suffices to establish condition (iii) for the two equalities of the form  $X = g(a)$  with  $X$  is of type  $msg$  and  $a$  is an atom. For these equations, condition (iii) immediately follows from (O4).

In Appendix H, we provide more general syntactic criteria to verify the conditions (iii) and (iv) under different sets of assumptions. These criteria are applicable to all our case studies and pave the way to the efficient automatic checking of soundness conditions.

## B Basic lemmas about the auxiliary functions and the type system

### B.1 Lemmas about the splitting and normal-form functions

**Lemma 1.** *For all  $t, u \in \mathcal{T}(V, \Sigma_0)$  and all substitutions  $\theta$ ,  $\text{split}(t) \subseteq \text{split}(u)$  implies that  $\text{split}(t\theta) \subseteq \text{split}(u\theta)$*

*Proof.* Suppose that  $\text{split}(t) \subseteq \text{split}(u)$  and  $v \in \text{split}(t\theta)$ . To show that  $v \in \text{split}(u\theta)$  we distinguish two cases:

1. There is some  $t' \in \text{split}(t)$  that is not a variable and  $v = t'\theta$ . Then  $t' \in \text{split}(u)$  and thus  $t'$  is not a pair. Since  $t'$  is neither a variable nor a pair, we have  $v \in \text{split}(u\theta)$ .
2. There is a variable  $X \in \text{split}(t)$  such that  $v \in \text{split}(X\theta)$ . Then  $X \in \text{split}(u)$  and thus  $v \in \text{split}(u\theta)$ .

This completes the proof of the lemma. □

The following lemma allows us to freely add and remove applications of the normal form function to any subterm of the term  $t$  in the expression  $\text{nf}_\ell(t)$ .

**Lemma 2.**

- (i)  $\text{nf}_\ell(\text{nf}_\ell(t)) = \text{nf}_\ell(t)$
- (ii)  $\text{nf}_\ell(g(t)) = \text{nf}_\ell(g(\text{nf}_\ell(t)))$  for  $g \in \Sigma^1$
- (iii)  $\text{nf}_\ell(\text{sh}(t, u)) = \text{nf}_\ell(\text{sh}(\text{nf}_\ell(t), u)) = \text{nf}_\ell(\text{sh}(t, \text{nf}_\ell(u)))$
- (iv)  $\text{nf}_\ell(\langle t, u \rangle) = \text{nf}_\ell(\langle \text{nf}_\ell(t), u \rangle) = \text{nf}_\ell(\langle t, \text{nf}_\ell(u) \rangle)$
- (v)  $\text{nf}_\ell(\{\!\{t}\!\}_u) = \text{nf}_\ell(\{\!\{\text{nf}_\ell(t)\!\}_u) = \text{nf}_\ell(\{\!\{t}\!\}_{\text{nf}_\ell(u)})$

**Lemma 3.** *Let  $t \in \mathcal{T}(V, \Sigma_0)$  and  $\theta$  be a substitution. If  $\text{nf}(t) = \text{nil}$ , then  $\text{nf}(t\theta) = \text{nil}$ .*

*Proof.* We prove the lemma by induction on  $t$ .

- If  $t$  is a variable then  $\text{nf}(t) \neq \text{nil}$ . Hence the lemma holds for this case.
- If  $t$  is an atom then  $\text{nf}(t) = \text{nil}$  implies that  $t = \text{nil}$ . Hence  $\text{nf}(t\theta) = \text{nil}$ .
- If  $t = \text{sh}(t_1, t_2)$  then  $\text{nf}(t) = \text{nil}$  implies that there is  $i \in \{1, 2\}$  such that  $\text{nf}(t_i) = \text{nil}$ . By induction hypothesis,  $\text{nf}(t_i) = \text{nil}$  implies  $\text{nf}(t_i\theta) = \text{nil}$  which implies  $\text{nf}(t\theta) = \text{nf}(\text{sh}(t_1\theta, t_2\theta)) = \text{nil}$ .
- If  $t = g(u)$  for  $g \in \Sigma^1$ , then  $\text{nf}(t) = \text{nil}$  implies that  $\text{nf}(u) = \text{nil}$ . Hence by induction hypothesis, we have  $\text{nf}(u\theta) = \text{nil}$ . Thus we obtain  $\text{nf}(t\theta) = \text{nf}(g(u\theta)) = \text{nil}$ .
- If  $t = \langle t_1, t_2 \rangle$ , then  $\text{nf}(t) = \text{nil}$  implies that  $\text{nf}(t_1) = \text{nf}(t_2) = \text{nil}$ . Hence by induction hypothesis, we have  $\text{nf}(t_1\theta) = \text{nf}(t_2\theta) = \text{nil}$ . Thus  $\text{nf}(t\theta) = \text{nf}(\langle t_1\theta, t_2\theta \rangle) = \text{nil}$ .
- If  $t = \{\!\{u}\!\}_k$ , then  $\text{nf}(t) = \text{nil}$  implies that  $\text{nf}(u) = \text{nil}$ . Hence by induction hypothesis, we have  $\text{nf}(u\theta) = \text{nil}$ . Thus  $\text{nf}(t\theta) = \text{nf}(\{\!\{u\theta}\!\}_{k\theta}) = \text{nil}$ .

This completes the proof of the lemma. □

**Lemma 4.** *Let  $t$  be a term and  $\sigma$  be a nil-free substitution. Then we have  $nf(t\sigma) = nf(t)\sigma$ .*

*Proof.* We proceed by induction on  $t$ .

- If  $t$  is an atom then  $nf(t\sigma) = nf(t) = t = nf(t)\sigma$  as required.
- If  $t$  is a variable then since  $\sigma$  is nil-free, we have that  $nf(t\sigma) = t\sigma = nf(t)\sigma$  as required.
- If  $t = g(u)$  for  $g \in \Sigma^1$  then we have  $nf(t\sigma) = nf(g(u\sigma))$ . By induction hypothesis, we have

$$nf(u\sigma) = nf(u)\sigma \quad (4)$$

Let us consider two cases.

- If  $nf(u) = \text{nil}$  then by (4) we have that  $nf(u\sigma) = \text{nil}$ . Hence  $nf(t\sigma) = \text{nil}$ . Moreover, we have  $nf(t) = nf(g(u)) = \text{nil}$ . Therefore, we obtain that  $nf(t\sigma) = nf(t)\sigma$ .
- If  $nf(u) \neq \text{nil}$  then since  $\sigma$  is nil-free, we have that  $nf(u\sigma) \neq \text{nil}$ . Therefore, we obtain that  $nf(t\sigma) = g(nf(u\sigma)) = g(nf(u)\sigma) = g(nf(u))\sigma = nf(t)\sigma$  as desired.
- If  $t = f(u_1, u_2)$  for  $f \in \Sigma^2$  then by a similar reasoning, we conclude that  $nf(t\sigma) = nf(t)\sigma$ .

This completes the proof of the lemma. □

**Lemma 5.** *For all terms  $t \in \mathcal{T}(V, \Sigma_0)$  and substitutions  $\theta$ , we have*

$$nf(nf(t)\theta) = nf(t\theta).$$

*Proof.* We prove the lemma by induction on  $t$ .

- If  $t$  is a variable or an atom then  $nf(t) = t$ . Hence,  $nf(nf(t)\theta) = nf(t\theta)$ .
- If  $t = \text{sh}(t_1, t_2)$  then we distinct two cases.
  - If there is  $i \in \{1, 2\}$  such that  $nf(t_i) = \text{nil}$  then  $nf(t) = \text{nil}$ . Moreover, by induction hypothesis, we have that  $nf(t_i\theta) = nf(nf(t_i)\theta) = \text{nil}$ . Hence  $nf(t\theta) = nf(\text{sh}(t_1\theta, t_2\theta)) = \text{nil}$ . Therefore, we obtain  $nf(nf(t)\theta) = nf(t\theta) = \text{nil}$ .
  - If  $nf(t_i) \neq \text{nil}$  for  $i \in \{1, 2\}$  then we have

$$\begin{aligned} nf(nf(t)\theta) &= nf(\text{sh}(nf(t_1)\theta, nf(t_2)\theta)) \\ &= nf(\text{sh}(nf(nf(t_1)\theta), nf(nf(t_2)\theta))) && \text{by Lemma 2} \\ &= nf(\text{sh}(nf(t_1)\theta, nf(t_2)\theta)) && \text{by ind. hyp.} \\ &= nf(\text{sh}(t_1\theta, t_2\theta)) && \text{by Lemma 2} \\ &= nf(\text{sh}(t_1, t_2)\theta) \\ &= nf(t\theta) \end{aligned}$$

- If  $t = g(u)$  for  $g \in \Sigma^1$  then by Definition 10, there are two cases:
  - $nf(u) = \text{nil}$  and  $nf(t) = \text{nil}$ . Using Lemma 3, we have

$$nf(nf(t)\theta) = \text{nil} = nf(t\theta)$$

- $nf(t) = g(nf(u))$ . Then we have

$$\begin{aligned}
nf(nf(t)\theta) &= nf(g(nf(u)\theta)) \\
&= nf(g(nf(nf(u)\theta))) && \text{by Lemma 2} \\
&= nf(g(nf(u)\theta)) && \text{by induction hypothesis} \\
&= nf(g(u\theta)) && \text{by Lemma 2} \\
&= nf(t\theta)
\end{aligned}$$

- If  $t = \langle t_1, t_2 \rangle$  then, by Definition 10, there are three cases:

- $nf(t_1) = \text{nil}$  and  $nf(t) = nf(t_2)$ . Then we have

$$\begin{aligned}
nf(nf(t)\theta) &= nf(nf(t_2)\theta) \\
&= nf(t_2\theta) && \text{by induction hypothesis} \\
&= nf(\langle t_1\theta, t_2\theta \rangle) && \text{by Lemma 3} \\
&= nf(t\theta)
\end{aligned}$$

- $nf(t_2) = \text{nil}$  and  $nf(t) = nf(t_2)$ . This case is symmetric to the previous one.
- $nf(t) = \langle nf(t_1), nf(t_2) \rangle$ . In this case, we have

$$\begin{aligned}
nf(nf(t)\theta) &= nf(\langle nf(t_1), nf(t_2) \rangle\theta) \\
&= nf(\langle nf(t_1)\theta, nf(t_2)\theta \rangle) \\
&= nf(\langle nf(nf(t_1)\theta), nf(nf(t_2)\theta) \rangle) && \text{by Lemma 2} \\
&= nf(\langle nf(t_1\theta), nf(t_2\theta) \rangle) && \text{by induction hypothesis} \\
&= nf(\langle t_1\theta, t_2\theta \rangle) && \text{by Lemma 2} \\
&= nf(t\theta)
\end{aligned}$$

- If  $t = \{u\}_k$  then, by Definition 10, there are three cases:

- $nf(u) = \text{nil}$  and  $nf(t) = \text{nil}$ . Then we use Lemma 3 to derive

$$nf(nf(t)\theta) = \text{nil} = nf(t\theta)$$

- $nf(k) = \text{nil}$  and  $nf(t) = nf(u)$ . This case is similar to the first two cases for pairs.
- $nf(t) = \{nf(u)\}_{nf(k)}$ . This case is similar to the third case for pairs.

This completes the proof of the lemma.  $\square$

**Lemma 6.** *For all terms  $t, u \in \mathcal{T}(V, \Sigma_0)$ , if  $t \in \text{split}(u)$  then we have*

$$\text{split}(nf(t)) \subseteq \text{split}(nf(u)) \cup \{\text{nil}\}.$$

*Proof.* We prove the lemma by induction on  $u$ .

- If  $u$  is not a pair then the lemma trivially holds.
- If  $u = \langle u_1, u_2 \rangle$  then  $t \in \text{split}(u)$  implies that  $t \in \text{split}(u_i)$  for some  $i \in \{1, 2\}$ . By induction hypothesis, we have

$$\text{split}(nf(t)) \subseteq \text{split}(nf(u_i)) \cup \{\text{nil}\}$$

This together with Definition 10 imply that

$$\text{split}(nf(t)) \subseteq \text{split}(nf(u)) \cup \{\text{nil}\}$$

This completes the proof of the lemma.  $\square$

## B.2 Lemmas about the type system

The subtyping relation respects the types' structures.

**Lemma 7.** *Let  $\tau, \tau' \in \mathcal{Y}$  be such that  $\tau \preceq \tau'$  and  $\tau' \neq \text{msg}$ . Then either*

- (i)  $\tau$  and  $\tau'$  are atomic and  $\tau \neq \text{msg}$ , or
- (ii)  $\tau$  and  $\tau'$  are composed and there are  $n \in \{1, 2\}$  and  $g \in \Sigma^n$  such that  $\tau = g(\tau_1, \dots, \tau_n)$ ,  $\tau' = g(\tau'_1, \dots, \tau'_n)$ , and  $\tau_i \preceq \tau'_i$  for  $i \in \tilde{n}$ .

*Proof.* We prove this lemma by rule induction on the derivation of  $\tau \preceq \tau'$ , depending on the last rule  $R$  that has been applied.

- $R = S(\text{msg})$ : we have  $\tau \in \mathcal{Y}$  and  $\tau' = \text{msg}$ , contradicting our assumption.
- $R = S(\preceq_0)$ : we have  $\tau \preceq_0 \tau'$ . Then it is clear that both  $\tau$  and  $\tau'$  are atomic and  $\tau \neq \text{msg}$  by the definition of  $\preceq_0$ .
- $R = S(\text{refl})$ : we have  $\tau = \tau'$  and thus the conclusion holds trivially.
- $R = S(\text{trans})$ : here, there is a  $\tau''$  such that  $\tau \preceq \tau''$  and  $\tau'' \preceq \tau'$ . Since  $\tau' \neq \text{msg}$ , we derive (i) or (ii) from the induction hypothesis for  $\tau'' \preceq \tau'$  to for  $\tau''$  and  $\tau$ . In both cases, we have  $\tau'' \neq \text{msg}$ . Therefore, we can also apply the induction hypothesis to  $\tau \preceq \tau''$ . Hence, we either have that  $\tau, \tau''$ , and  $\tau'$  are all atomic and  $\tau \neq \text{msg}$  or they all have the same top-level constructor  $g$  and the arguments of  $\tau$  and  $\tau''$  and of  $\tau''$  and  $\tau'$  are in the subtyping relation and we conclude by applying  $S(\text{trans})$  on the argument types.
- $R = S(\Sigma^1)$  or  $R = S(\Sigma^2)$ : In this case, the conclusion (ii) follows directly from the rules' premises and conclusions. □

**Lemma 8.** *For all  $\tau' \in \mathcal{Y} \setminus \mathcal{Y}^{\text{at}}$  and  $\tau \in \mathcal{Y}^{\text{at}}$ ,  $\tau' \preceq \tau$  implies  $\tau = \text{msg}$ .*

*Proof.* Follows immediately from Lemma 7. □

**Lemma 9.** *Let  $\tau, \tau' \in \mathcal{Y}$  such that  $\tau \preceq \tau'$  and  $\text{msg} \notin \text{subs}(\tau')$ . Then we have  $\text{msg} \notin \text{subs}(\tau)$ .*

*Proof.* We proceed by rule induction on the derivation of  $\tau \preceq \tau'$  depending on the last rule  $R$  that has been applied.

- $R = S(\text{msg})$ : we have  $\tau' = \text{msg}$  which contradicts our assumption.
- $R = S(\preceq_0)$ : we have  $(\tau, \tau') \in \preceq_0$ . It follows that  $\text{msg} \notin \text{subs}(\tau)$  as required.
- $R = S(\text{refl})$ : we have  $\tau = \tau'$ . The conclusion holds trivially for this case.
- $R = S(\text{trans})$ : there is  $\tau''$  such that  $\tau \preceq \tau''$  and  $\tau'' \preceq \tau'$ . Then by induction hypothesis, we have that  $\text{msg} \notin \text{subs}(\tau'')$  and  $\text{msg} \notin \text{subs}(\tau)$  as required.
- $R = S(\Sigma^1)$ : there are  $\tau_1, \tau_2$  and  $g \in \Sigma^1$  such that  $\tau = g(\tau_1)$ ,  $\tau' = g(\tau_2)$ , and  $\tau_1 \preceq \tau_2$ . Since  $\text{msg} \notin \text{subs}(\tau')$ , we have  $\text{msg} \notin \text{subs}(\tau_2)$ . Therefore, by induction hypothesis, we have  $\text{msg} \notin \text{subs}(\tau_1)$ . This means  $\text{msg} \notin \text{subs}(\tau)$  as required.
- $R = S(\Sigma^2)$ : this case holds by a similar reasoning as the previous case. □

The following lemma states a property of the type  $\gamma_{\text{nil}}$ .

**Lemma 10.** *For every type  $\tau$ , we have that*

1.  $\tau \preceq \gamma_{\text{nil}}$  *implies*  $\tau = \gamma_{\text{nil}}$ .
2.  $\gamma_{\text{nil}} \preceq \tau$  *implies*  $\tau \in \{\gamma_{\text{nil}}, \text{msg}\}$ .

The next result immediately follows from Lemma 10.

**Corollary 1.** *Let  $\Gamma$  be a typing environment. Then for all types  $\tau \in \text{ran}(\Gamma)$ , we have that  $\gamma_{\text{nil}} \preceq \tau$  *implies*  $\tau = \text{msg}$ .*

The following lemma states that well-typed substitutions respect types.

**Lemma 11.** *Let  $\theta$  be a well-typed substitution with respect to a typing environment  $\Gamma$ . Then for all terms  $t \in \mathcal{T}$ ,  $\Gamma(t) = \tau$  *implies* that  $\Gamma(t\theta) = \tau'$  and  $\tau' \preceq \tau$ .*

## C Basic properties of type-based abstractions

In this section, we prove several properties of type-based abstractions. First, we show that two terms whose types are in a subtyping relation must be transformed by the same clause. Second, we describe the shapes of transformed terms in different cases. At the end, we prove that type inference is preserved under abstractions.

### C.1 Uniform matching

The following lemma states that a term  $t$  typing in a typing environment matches a linear pattern  $p$  whenever  $t$ 's type is a subtype  $p$ 's type.

**Lemma 12.** *Let  $\Gamma' : V \rightarrow \mathcal{Y}$ ,  $\Gamma : \mathcal{V}_{pt} \rightarrow \mathcal{Y}$  be typing environments and  $p \in \mathcal{P}$  be a linear pattern such that  $\text{vars}(p) \subseteq \text{dom}(\Gamma)$ . Then, for all  $t \in \mathcal{T} \cup \mathcal{Y}$  such that  $\Gamma'(t) \preceq \Gamma(p)$  there exists a substitution  $\sigma : \text{vars}(p) \rightarrow \mathcal{T} \cup \mathcal{Y}$  such that  $p\sigma = t$ .*

*Proof.* We prove the lemma by induction on the structure of  $p$ . Below we use the abbreviations  $\tau = \Gamma'(t)$  and  $\pi = \Gamma(p)$ .

- If  $p$  is a pattern variable, then we define  $\sigma = \{p \mapsto t\}$ , hence  $p\sigma = t$ .
- If  $p = g(q)$  for  $g \in \Sigma^1$ , then since  $\Gamma(p) = \pi$ , there exists  $\pi'$  such that

$$\pi = g(\pi') \text{ and } \Gamma(q) = \pi'.$$

Since  $\tau \preceq \pi$ , we have  $\tau \preceq g(\pi')$ . Hence, by Lemma 7, we have  $\tau = g(\tau')$  and  $\tau' \preceq \pi'$ . Since  $\Gamma'(t) = \tau$  and  $\tau$  is composed,  $t$  is not a variable. Therefore, we have

$$t = g(t') \text{ and } \Gamma'(t') = \tau'.$$

Therefore, by induction hypothesis, there exists  $\sigma : \text{vars}(q) \rightarrow \mathcal{T} \cup \mathcal{Y}$  such that  $q\sigma = t'$ . Hence, we have  $p\sigma = t$  as required.

- If  $p = g(p_1, p_2)$  for  $g \in \Sigma^2$ , then since  $\Gamma(p) = \pi$ , there exist  $\pi_1, \pi_2$  such that

$$\pi = g(\pi_1, \pi_2) \text{ and } \Gamma(p_i) = \pi_i \text{ for } i \in \{1, 2\}.$$

Since  $\tau \preceq \pi$ , by Lemma 7, we have

$$\tau = g(\tau_1, \tau_2) \text{ and } \tau_i \preceq \pi_i \text{ for } i \in \{1, 2\}.$$

Since  $\Gamma'(t) = \tau$  and  $\tau$  is composed,  $t$  is not a variable. Therefore, we have

$$t = g(t_1, t_2) \text{ and } \Gamma'(t_i) = \tau_i \text{ for } i \in \{1, 2\}.$$

Hence, by induction hypothesis, there are  $\sigma_i : \text{vars}(p_i) \rightarrow \mathcal{T} \cup \mathcal{Y}$  such that  $t_i = p_i\sigma_i$  for  $i \in \{1, 2\}$ . Since  $p$  is linear, we have  $\text{vars}(p_1) \cap \text{vars}(p_2) = \emptyset$ . We can thus define  $\sigma : \text{vars}(p) \rightarrow \mathcal{T} \cup \mathcal{Y}$  by  $\sigma = \sigma_1 \cup \sigma_2$ . Hence, we obtain  $p\sigma = t$ .

This completes the proof of the lemma. □

**Lemma 13 (Uniform matching).** *Let  $\Gamma$  be a typing environment and  $F_f = (f, \Gamma_f, E_f)$  be a function specification where  $E_f = [f(p_1) = u_1, \dots, f(p_n) = u_n]$ . Let*

$$\text{matches}(t) = \{i \in \tilde{n} \mid \exists \theta. t = p_i \theta \wedge \Gamma(t) \preceq \Gamma_f(p_i)\}.$$

*Then, for all  $t, t' \in \mathcal{T} \cup \mathcal{Y}$  with  $\Gamma(t') \preceq \Gamma(t)$ , we have*

- (i)  $\text{matches}(t) \subseteq \text{matches}(t')$ ,
- (ii)  $\text{matches}(t) = \text{matches}(t') = \{i\}$  for some  $i \in \tilde{n}$  if  $F_f$  is pattern-disjoint and  $\Gamma(t) \in \Pi_f \downarrow$ .

*In particular,  $\text{matches}(t) = \text{matches}(\Gamma(t))$  for all terms  $t \in \mathcal{T}$ .*

*Proof.* Let  $t, t' \in \mathcal{T} \cup \mathcal{Y}$ ,  $\tau$  and  $\tau'$  be types such that  $\Gamma(t) = \tau$ ,  $\Gamma(t') = \tau'$ , and  $\tau' \preceq \tau$ . To see (i), suppose  $i \in \text{matches}(t)$ , i.e.,  $t = p_i \theta$  and  $\Gamma(t) \preceq \Gamma_f(p_i)$  for some substitution  $\theta$ . Since  $\Gamma(t') \preceq \Gamma(t)$ , we also have  $\Gamma(t') \preceq \Gamma_f(p_i)$  and hence  $i \in \text{matches}(t')$ . This shows (i). In particular, since  $\Gamma$  is the identity on types, it trivially follows that  $\text{matches}(t) = \text{matches}(\tau)$  for all terms  $t \in \mathcal{T}$  as claimed at the end of the lemma statement.

To see (ii), we first derive  $\tau' \in \Pi_f \downarrow$  from the assumptions  $\tau \in \Pi_f \downarrow$  and  $\tau' \preceq \tau$ . Therefore, there are  $i, j \in \tilde{n}$  and  $\{\pi_i, \pi_j\} \subseteq \Pi_f$  such that  $\tau \preceq \pi_i$  and  $\tau' \preceq \pi_j$ . Moreover,  $i$  and  $j$  are unique since  $(\Gamma_f, E_f)$  is pattern-disjoint. By Lemma 12 there are substitutions  $\theta$  and  $\theta'$  (with domains  $\text{vars}(p_i)$  and  $\text{vars}(p_j)$ ) such that  $t = p_i \theta$  and  $t' = p_j \theta'$ . Hence,  $\text{matches}(t) = \{i\}$  and  $\text{matches}(t') = \{j\}$ . Using the result in (i) derive  $i = j$  as required.  $\square$

## C.2 Shape lemma and termination

**Lemma 14 (Shape lemma).** *Let  $\Gamma$  be a typing environment and  $F_f = (f, \Gamma_f, E_f)$  a type-based abstraction. Then for all  $t \in \mathcal{T} \cup \mathcal{Y}$  such that  $t$  is in normal form and  $\Gamma(t)$  is defined, the following holds for  $f(t, \Gamma, \ell)$  (we omit the parameters  $\Gamma$  and  $\ell$  below):*

- (i) *If  $t$  is a variable or an atom, then  $f(t) = t$  or  $f(t) = \ell$ .*
- (ii) *If  $t = g(u)$  for  $g \in \{\text{pk}, \text{pri}\}$  then  $f(t) = \text{nf}_\ell(g(f(u)))$ .*
- (iii) *If  $t = \text{sh}(v_1, v_2)$  then  $f(t) = \text{nf}_\ell(\text{sh}(f(v_1), f(v_2)))$ .*
- (iv) *If  $t = h(u)$  for  $h \in \Sigma_u$  then either  $f(t) = \ell$  or*

$$f(t) = \text{nf}_\ell(\langle h^{a_1}(\widehat{f}(\overline{v}_1)), \dots, h^{a_i}(\widehat{f}(\overline{v}_i)) \rangle)$$

*for some  $l > 0$  and vectors  $\overline{v}_1, \dots, \overline{v}_i$  such that  $\text{split}(\overline{v}_i) \subseteq \text{split}(u)$ ,  $\text{set}(\overline{v}_i) \subseteq \text{subs}(u)$ , and  $a_i \geq 0$  for all  $i \in \tilde{l}$ , and*

- (v) *If  $t = \langle v_1, v_2 \rangle$  then  $f(t) = \text{nf}_\ell(\widehat{f}(\overline{r}))$  for some vector  $\overline{r}$  such that  $\text{split}(\overline{r}) = \text{split}(t)$  and  $\text{set}(\overline{r}) \subseteq \text{subs}(t) \setminus \{t\}$ .*
- (vi) *If  $t = \{w\}_k$  then we have*

$$f(t) = \text{nf}_\ell(\langle \{\widehat{f}(\overline{s}_1)\}_{\widehat{f}(\overline{v}_1)}^{a_1}, \dots, \{\widehat{f}(\overline{s}_d)\}_{\widehat{f}(\overline{v}_d)}^{a_d} \rangle)$$

for some  $d > 0$  and vectors  $\bar{s}_1, \dots, \bar{s}_d$  and  $\bar{v}_1, \dots, \bar{v}_d$  with  $\bigcup_{i=1}^d \text{split}(\bar{s}_i) = \text{split}(w)$ , and, for all  $i \in \tilde{d}$ ,  $\text{set}(\bar{s}_i) \subseteq \text{subs}(w)$ ,  $\text{split}(\bar{v}_i) \subseteq \text{split}(k)$ ,  $\text{set}(\bar{v}_i) \subseteq \text{subs}(k)$ , and  $a_i \geq 0$ .

*Proof.* By case distinction on the shape of the term (or type)  $t \in \mathcal{T} \cup \mathcal{Y}$ . For terms  $t$  in normal form we have  $f(t, \Gamma, \ell) = f^{\text{rec}}(t, \Gamma, \ell)$ . Therefore, we write  $f$  instead of  $f^{\text{rec}}$  below for convenience. Then there exists the first pattern  $f(p_i) = u_i$  in the list  $E_f^1$  such that  $\Gamma(t) \preceq \Gamma_f^1(p_i)$ . By Lemma 12, there is a substitution  $\theta$  such that  $p_i\theta = t$ . Hence, by Program 1, we have

$$f(t) = \text{nf}_\ell(u_i[\ell/\text{Nil}]\theta). \quad (5)$$

Case (i) where  $t$  is a variable or an atom follows immediately from Definition 5, the definition of  $E_f^0$ , and Program 1. Suppose  $t$  is composed. Note that the substitution in  $u_i[\ell/\text{Nil}]$  only applies to case (iv) of hashes. We now prove the cases (ii) to (vi).

- Case (ii) where  $t = g(u)$  for  $g \in \{\text{pk}, \text{pri}\}$ : since  $p_i$  is not a pattern variable, we must have that  $p_i = g(q)$ ,  $u_i = g(f(q))$ , and  $u = q\theta$ . Hence, by (5) we have  $f(t) = \text{nf}_\ell(g(f(q\theta))) = \text{nf}_\ell(g(f(u)))$  as required.
- Case (iii) where  $t = \text{sh}(u, v)$ : This case is similar to the previous one.
- Case (iv) where  $t = h(u)$  for  $h \in \Sigma_u$ : since  $p_i$  is not a pattern variable and  $t = p_i\theta$ , we must have  $p_i = h(q)$  and  $q\theta = u$ . By (5) and Definition 5, one of the following holds:
  - $u_i = \langle h^{a_1}(\widehat{f}(\bar{q}_1)), \dots, h^{a_l}(\widehat{f}(\bar{q}_l)) \rangle$  for some  $l > 0$  and vectors  $\bar{q}_1, \dots, \bar{q}_l$  such that  $\text{set}(\bar{q}_i) \subseteq \text{split}(q)$  and  $a_i > 0$  for all  $i \in \tilde{l}$ . From (5) we have

$$f(t) = \text{nf}_\ell(\langle h^{a_1}(\widehat{f}(\bar{v}_1)), \dots, h^{a_l}(\widehat{f}(\bar{v}_l)) \rangle)$$

where  $\bar{v}_i = \bar{q}_i\theta$  for all  $i \in \tilde{l}$ . From  $\text{set}(\bar{q}_i) \subseteq \text{split}(q)$  we derive, for all  $i \in \tilde{l}$ , (a)  $\text{split}(\bar{q}_i) \subseteq \text{split}(q)$  and hence, by Lemma 1,  $\text{split}(\bar{v}_i) \subseteq \text{split}(u)$ , and (b)  $\text{set}(\bar{q}_i) \subseteq \text{subs}(q)$  and hence  $\text{set}(\bar{v}_i) \subseteq \text{subs}(u)$ .

- $u_i = \text{Nil}$ . Then we have  $f(t) = \text{nf}_\ell(\text{Nil}[\ell/\text{Nil}]) = \ell$  as required.
- Case (v) where  $t = \langle v_1, v_2 \rangle$ : since  $p_i$  is not a pattern variable and  $t = p_i\theta$ , we have  $p_i = \langle q_1, q_2 \rangle$  with  $v_1 = q_1\theta$  and  $v_2 = q_2\theta$ . By Definition 5, there is a vector  $\bar{s}$  such that  $u_i = \widehat{f}(\bar{s})$  and  $\text{set}(\bar{s}) = \text{split}(p_i)$ . Let  $\bar{r} = \bar{s}\theta$ . First, from (5), we have  $f(t) = \text{nf}_\ell(\widehat{f}(\bar{r}))$ . Second, since  $\text{set}(\bar{s}) = \text{split}(p_i)$ , we also have  $\text{split}(\bar{s}) = \text{split}(p_i)$  and hence, by Lemma 1,  $\text{split}(\bar{r}\theta) = \text{split}(t)$ . Finally, since  $\text{set}(\bar{s}) = \text{split}(p_i)$ , we have  $s_j \in \text{subs}(p_i) \setminus \{p_i\}$  and therefore  $r_j \in \text{subs}(t) \setminus \{t\}$  for all  $j \in \tilde{m}$ . Here,  $m$  is the length of  $\bar{r}$  and  $\bar{s}$ . Thus,  $\text{set}(\bar{r}) \subseteq \text{subs}(t) \setminus t$  as required.
- Case (vi) where  $t = \{w\}_k$ : since  $p_i$  is not a pattern variable and  $t = p_i\theta$ , we must have  $p_i = \{q\}_r$  with  $w = q\theta$  and  $k = r\theta$ . By Definition 5, we have

$$u_i = \langle \{\widehat{f}(\bar{q}_1)\}_{\widehat{f}(\bar{r}_1)}^{a_1}, \dots, \{\widehat{f}(\bar{q}_d)\}_{\widehat{f}(\bar{r}_d)}^{a_d} \rangle$$

for some  $d > 0$  and vectors  $\bar{q}_1, \dots, \bar{q}_d$ ,  $\bar{r}_1, \dots, \bar{r}_d$  such that  $\bigcup_{i=1}^d \text{set}(\bar{q}_i) = \text{split}(q)$  and, for all  $i \in \tilde{d}$ ,  $\text{set}(\bar{r}_i) \subseteq \text{split}(r)$  and  $a_i \geq 0$ .

First, from (5) we have

$$f(t) = nf_\ell(\langle \{\widehat{f}(\overline{s}_1)\}_{\widehat{f}(\overline{v}_1)}^{a_1}, \dots, \{\widehat{f}(\overline{s}_d)\}_{\widehat{f}(\overline{v}_d)}^{a_d} \rangle)$$

where  $\overline{s}_i = \overline{q}_i\theta$  and  $\overline{v}_i = \overline{r}_i\theta$ . Second, we have  $\bigcup_{i=1}^d \text{split}(\overline{q}_i) = \text{split}(q)$  and  $\text{split}(\overline{r}_i) \subseteq \text{split}(r)$  for all  $i \in \tilde{d}$ . Hence, by Lemma 1, we have  $\bigcup_{i=1}^d \text{split}(\overline{s}_i) = \text{split}(w)$  and  $\bigcup_{i=1}^d \text{split}(\overline{v}_i) \subseteq \text{split}(k)$ . Finally, from  $\text{set}(\overline{q}_i) \subseteq \text{split}(q)$  we derive  $\text{set}(\overline{q}_i) \subseteq \text{subs}(q)$  and therefore  $\text{set}(\overline{s}_j) \subseteq \text{subs}(w)$  for all  $i \in \tilde{d}$ . Similarly, we can show that  $\text{set}(\overline{v}_j) \subseteq \text{subs}(k)$ .

This completes the proof of the lemma.  $\square$

**Proposition 1 (Termination).** *Let  $\Gamma$  be a typing environment and  $F_f$  be a type-based abstraction. Then the function  $f$  defined by Program 1 terminates on all terms  $t \in \mathcal{T} \cup \mathcal{Y}$  such that  $\Gamma(t)$  is defined.*

*Proof.* We show that  $f(t) = f^{\text{rec}}(nf_\ell(t))$  terminates for all  $t \in \mathcal{T} \cup \mathcal{Y}$  (omitting the parameters  $\Gamma$  and  $\ell$ ). Clearly,  $nf_\ell$  terminates on all inputs, since its recursion is structural. Therefore, it suffices to show that  $f^{\text{rec}}(t)$  terminates on all normal-form terms  $t \in \mathcal{T} \cup \mathcal{Y}$  where  $\Gamma(t)$  is defined. We prove this by induction on the size of  $t$ . If  $\Gamma(t)$  is an atom then the termination of  $f^{\text{rec}}(t)$  is immediate. If  $\Gamma(t)$  is composed then from Lemma 14 we know that that  $f^{\text{rec}}$  is called recursively on subterms of  $t$ , which are thus also in normal form. Hence, these calls terminate by the induction hypothesis. Therefore,  $f^{\text{rec}}(t)$  also terminates. This completes the proof of the proposition.  $\square$

Next, we prove that all abstracted protocols are protocols. This result enables chaining different abstractions to obtain more complex one.

**Proposition 2.** *Let  $P = (\Gamma_P, S_P)$  be a protocol and  $F_f = (f, \Gamma_f, E_f)$  be a type-based abstraction. Then  $f(P)$  is also a protocol.*

*Proof.* We need to show four points.

- (i)  $\Gamma_{f(P)}$  is a typing environment,
- (ii)  $S_{f(P)}$  is a partial function from agent variables to roles and  $\text{dom}(S_{f(P)}) \subseteq \Gamma_{f(P)}^{-1}(\alpha)$ ,
- (iii)  $\mathcal{V}_{f(P)} \subseteq \text{dom}(\Gamma_{f(P)})$ , and
- (iv)  $\text{nil} \notin \mathcal{C}_{f(P)}$ .

By Proposition 1, we know that  $f(t)$  is defined for all terms  $t \in \mathcal{M}_P$ . By Lemma 14, we derive that  $f(\Gamma_P) \subseteq \Gamma_P$ . Hence point (i) immediately follows. To see point (ii), note that  $\text{dom}(S_{f(P)}) = \text{dom}(S_P) \subseteq \Gamma_P^{-1}(\alpha) = \Gamma_{f(P)}^{-1}(\alpha)$  and for all  $R \in \text{dom}(S_P)$ , we have  $S_{f(P)}(R) = f(S_P(R)) \in \text{Evt}^*$ . Therefore, we conclude point (ii). We also obtain point (iii) from the fact that  $\mathcal{V}_{f(P)} = \text{vars}(f(\mathcal{M}_P)) \subseteq \text{dom}(f(\Gamma_P)) = \text{dom}(\Gamma_{f(P)})$ . Moreover, we have  $\text{nil} \notin \mathcal{C}_P$ . Hence by Definition 6, we derive point (iv). This completes the proof of the proposition.  $\square$

### C.3 Type inference preservation

**Lemma 15.**  $\Gamma(nf_{\text{nil}}(t)) = nf_{\gamma_{\text{nil}}}(\Gamma(t))$  for all typing environments  $\Gamma$  and terms  $t \in \mathcal{T}$  such that  $\Gamma(t)$  is defined.

*Proof.* By induction on the structure of  $t$ . For the base case where  $t$  is an atom or a variable, we have  $\Gamma(nf_{\text{nil}}(t)) = \Gamma(t) = nf_{\gamma_{\text{nil}}}(\Gamma(t))$ . To prove the inductive cases, we first observe that  $\Gamma(nf_{\text{nil}}(t)) = nf_{\gamma_{\text{nil}}}(\Gamma(t))$  implies that

$$nf_{\text{nil}}(t) = \text{nil} \text{ if and only if } nf_{\gamma_{\text{nil}}}(\Gamma(t)) = \gamma_{\text{nil}}. \quad (6)$$

Below, we consider the case where  $t = \{t\}_u$ . The other cases are analogous.

$$\begin{aligned} \Gamma(nf_{\text{nil}}(\{t\}_u)) &= \Gamma(\text{if } nf_{\text{nil}}(t) = \text{nil} \text{ then nil} && \text{definition of } nf_{\text{nil}} \\ &\quad \text{else if } nf_{\text{nil}}(u) = \text{nil} \text{ then } nf_{\text{nil}}(t) \\ &\quad \text{else } \{nf_{\text{nil}}(t)\}_{nf_{\text{nil}}(u)}) \\ &= \text{if } nf_{\text{nil}}(t) = \text{nil} \text{ then } \Gamma(\text{nil}) && \text{push } \Gamma \text{ inside} \\ &\quad \text{else if } nf_{\text{nil}}(u) = \text{nil} \text{ then } \Gamma(nf_{\text{nil}}(t)) \\ &\quad \text{else } \{\Gamma(nf_{\text{nil}}(t))\}_{\Gamma(nf_{\text{nil}}(u))}) \\ &= \text{if } nf_{\gamma_{\text{nil}}}(\Gamma(t)) = \gamma_{\text{nil}} \text{ then } \gamma_{\text{nil}} && \text{ind. hyp. and (6)} \\ &\quad \text{else if } nf_{\gamma_{\text{nil}}}(\Gamma(u)) = \gamma_{\text{nil}} \text{ then } nf_{\gamma_{\text{nil}}}(\Gamma(t)) \\ &\quad \text{else } \{nf_{\gamma_{\text{nil}}}(\Gamma(t))\}_{nf_{\gamma_{\text{nil}}}(\Gamma(u))}) \\ &= nf_{\gamma_{\text{nil}}}(\{\Gamma(t)\}_{\Gamma(u)}) && \text{definition of } nf_{\gamma_{\text{nil}}} \\ &= nf_{\gamma_{\text{nil}}}(\Gamma(\{t\}_u)) && \text{pull } \Gamma \text{ outside} \end{aligned}$$

This concludes the proof of the lemma.  $\square$

**Proposition 3 (Type inference preservation).** *Let  $\Gamma$  be a typing environment and  $F_f = (f, \Gamma_f, E_f)$  a type-based abstraction. Then we have  $f(\Gamma)(f(t)) = f(\Gamma(t))$  for all terms  $t \in \mathcal{T}$  such that  $\Gamma(t)$  is defined.*

*Proof.* As the following derivation shows, it is sufficient to show the conclusion of the proposition for normal-form terms:

$$\begin{aligned} f(\Gamma)(f(t)) &= f(\Gamma)(f^{rec}(nf_{\text{nil}}(t))) \\ &= f^{rec}(\Gamma(nf_{\text{nil}}(t))) && (*), \text{ established below} \\ &= f^{rec}(nf_{\gamma_{\text{nil}}}(\Gamma(t))) && \text{by Lemma 15} \\ &= f(\Gamma(t)) \end{aligned}$$

In the following, we show (\*) by induction on the size of normal-form terms  $t$ , writing below  $f$  instead of  $f^{rec}$  and  $f_\tau$  to emphasize where  $f$  is applied to a type. Note that the transformed environment  $f(\Gamma)$  is the restriction of  $\Gamma$  to variables  $X$  with  $f(X) = X$ . If  $t$  is a variable or an atom then we have one of the following cases

(a)  $f_\tau(\Gamma(t)) = \gamma_{\text{nil}}$ . Hence,  $f(t) = \text{nil}$  as required,

$$f(\Gamma)(f(t)) = f(\Gamma)(\text{nil}) = \gamma_{\text{nil}} = f_\tau(\Gamma(t)).$$

(b)  $f_\tau(\Gamma(t)) = \Gamma(t)$ . In this case, we have  $f(t) = t$  and

$$f(\Gamma)(f(t)) = f(\Gamma)(t) = \Gamma(t) = f_\tau(\Gamma(t)).$$

The second equation holds, since  $f(t) = t$  and therefore  $t$  does not contain any variables that  $f$  removes.

We proceed with the inductive cases. In each case, we use Lemmas 13 and 14 in the following way. From Lemma 13 we derive that a term  $t$  and its inferred type  $\Gamma(t)$  match the same clauses in  $E_f$ . From Lemma 14, we determine the shapes of  $f(t)$  and  $f_\tau(\Gamma(t))$ , knowing  $t$  and  $\Gamma(t)$  match the same first clause.

- If  $t = g(u)$  for  $g \in \{\text{pk}, \text{pri}\}$  then  $f(t) = nf_{\text{nil}}(g(f(u)))$  and  $f_\tau(\Gamma(t)) = nf_{\gamma_{\text{nil}}}(g(f(\Gamma(u))))$ . We derive:

$$\begin{aligned} f(\Gamma)(f(g(u))) &= f(\Gamma)(nf_{\text{nil}}(g(f(u)))) \\ &= nf_{\gamma_{\text{nil}}}(f(\Gamma)(g(f(u)))) && \text{by Lemma 15} \\ &= nf_{\gamma_{\text{nil}}}(g(f(\Gamma)(f(u)))) \\ &= nf_{\gamma_{\text{nil}}}(g(f_\tau(\Gamma(u)))) && \text{by induction hypothesis} \\ &= f_\tau(g(\Gamma(u))) \\ &= f_\tau(\Gamma(g(u))) \end{aligned}$$

- If  $t = \text{sh}(u, v)$  then  $f(t) = nf_{\text{nil}}(\text{sh}(f(u), f(v)))$ . This case is similar to the previous one.
- If  $t = h(u)$  for  $h \in \Sigma_u$  then there are two cases:
  1. We have

$$\begin{aligned} f(t) &= nf_{\text{nil}}(\langle h^{a_1}(\widehat{f}(\overline{v}_1)), \dots, h^{a_l}(\widehat{f}(\overline{v}_l)) \rangle) \\ f_\tau(\Gamma(t)) &= nf_{\gamma_{\text{nil}}}(\langle h^{a_1}(\widehat{f}_\tau(\widehat{\Gamma}(\overline{v}_1))), \dots, h^{a_l}(\widehat{f}_\tau(\widehat{\Gamma}(\overline{v}_l))) \rangle) \end{aligned}$$

for some  $l \geq 1$ ,  $a_i \geq 0$ , and vectors  $\overline{v}_1, \dots, \overline{v}_l$  such that  $\text{split}(\overline{v}_i) \subseteq \text{split}(u)$ . Then we derive

$$\begin{aligned} f(\Gamma)(f(t)) &= f(\Gamma)(nf_{\text{nil}}(\langle h^{a_1}(\widehat{f}(\overline{v}_1)), \dots, h^{a_l}(\widehat{f}(\overline{v}_l)) \rangle)) \\ &= nf_{\gamma_{\text{nil}}}(f(\Gamma)(\langle h^{a_1}(\widehat{f}(\overline{v}_1)), \dots, h^{a_l}(\widehat{f}(\overline{v}_l)) \rangle)) && \text{by Lemma 15} \\ &= nf_{\gamma_{\text{nil}}}(\langle h^{a_1}(\widehat{f}(\Gamma)(\widehat{f}(\overline{v}_1))), \dots, h^{a_l}(\widehat{f}(\Gamma)(\widehat{f}(\overline{v}_l))) \rangle) \\ &= nf_{\gamma_{\text{nil}}}(\langle h^{a_1}(\widehat{f}_\tau(\widehat{\Gamma}(\overline{v}_1))), \dots, h^{a_l}(\widehat{f}_\tau(\widehat{\Gamma}(\overline{v}_l))) \rangle) && \text{by induction hyp.} \\ &= f_\tau(\Gamma(t)) \end{aligned}$$

2.  $f(t) = \text{nil}$  and  $f_\tau(\Gamma(t)) = \gamma_{\text{nil}}$ . Then we derive

$$f(\Gamma)(f(t)) = f(\Gamma)(\text{nil}) = \gamma_{\text{nil}} = f_\tau(\Gamma(t))$$

as required.

- If  $t = \langle u, v \rangle$  then this case is similar to the first case for hashes above, where we consider  $a_1 = \dots = a_l = 0$ .

– If  $t = \{u\}_k$  then we have

$$f(t) = nf_{\text{nil}}(\langle \{\widehat{f}(\overline{v}_1)\}_{\widehat{f}(\overline{s}_1)}^{a_1}, \dots, \{\widehat{f}(\overline{v}_l)\}_{\widehat{f}(\overline{s}_l)}^{a_l} \rangle)$$

$$f_\tau(\Gamma(t)) = nf_{\gamma_{\text{nil}}}(\langle \{\widehat{f}_\tau(\widehat{\Gamma}(\overline{v}_1))\}_{\widehat{f}_\tau(\widehat{\Gamma}(\overline{s}_1))}^{a_1}, \dots, \{\widehat{f}_\tau(\widehat{\Gamma}(\overline{v}_l))\}_{\widehat{f}_\tau(\widehat{\Gamma}(\overline{s}_l))}^{a_l} \rangle)$$

for some  $l \geq 1$ ,  $a_i \geq 0$ , and vectors  $\overline{v}_1, \dots, \overline{v}_l, \overline{s}_1, \dots, \overline{s}_l$  such that  $\bigcup_1^l \text{split}(\overline{v}_i) = \text{split}(u)$  and  $\text{split}(\overline{s}_i) \subseteq \text{split}(k)$ . Then we derive

$$\begin{aligned} & f(\Gamma)(f(t)) \\ &= f(\Gamma)(nf_{\text{nil}}(\langle \{\widehat{f}(\overline{v}_1)\}_{\widehat{f}(\overline{s}_1)}^{a_1}, \dots, \{\widehat{f}(\overline{v}_l)\}_{\widehat{f}(\overline{s}_l)}^{a_l} \rangle)) \\ &= nf_{\gamma_{\text{nil}}}(f(\Gamma)(\langle \{\widehat{f}(\overline{v}_1)\}_{\widehat{f}(\overline{s}_1)}^{a_1}, \dots, \{\widehat{f}(\overline{v}_l)\}_{\widehat{f}(\overline{s}_l)}^{a_l} \rangle)) && \text{Lemma 15} \\ &= nf_{\gamma_{\text{nil}}}(\langle \{\widehat{f(\Gamma)}(\widehat{f}(\overline{v}_1))\}_{\widehat{f(\Gamma)}(\widehat{f}(\overline{s}_1))}^{a_1}, \dots, \{\widehat{f(\Gamma)}(\widehat{f}(\overline{v}_l))\}_{\widehat{f(\Gamma)}(\widehat{f}(\overline{s}_l))}^{a_l} \rangle)) \\ &= nf_{\gamma_{\text{nil}}}(\langle \{\widehat{f}_\tau(\widehat{\Gamma}(\overline{v}_1))\}_{\widehat{f}_\tau(\widehat{\Gamma}(\overline{s}_1))}^{a_1}, \dots, \{\widehat{f}_\tau(\widehat{\Gamma}(\overline{v}_l))\}_{\widehat{f}_\tau(\widehat{\Gamma}(\overline{s}_l))}^{a_l} \rangle) && \text{by IH} \\ &= f_\tau(\Gamma(t)) \end{aligned}$$

This concludes the proof of the proposition. □

## D Deducibility preservation for ground terms

*Notation.* For the sake of a lighter notation, we will omit set braces in intruder derivations and write, e.g.,  $T, t \vdash u$  instead of  $T \cup \{t\} \vdash u$  for a set of terms  $T$  and individual terms  $t$  and  $u$ . We also write  $T \vdash U$  for a set of terms  $U$  to mean that all terms in  $U$  are derivable from those in  $T$ .

**Lemma 16.** *Let  $T$  be a set of terms such that  $\text{nil} \in T$ . If one of the following holds*

- (i)  $t = \langle t_1, \dots, t_n \rangle$  and, for all  $i \in \tilde{n}$ ,  $T \vdash t_i$  and  $t_i$  is in normal form, or
- (ii)  $t = \langle h^{a_1}(t_1), \dots, h^{a_n}(t_n) \rangle$  for some  $h \in \Sigma_u$  and, for all  $i \in \tilde{n}$ ,  $T \vdash t_i$  and  $t_i$  is in normal form,
- (iii)  $t = \langle \{\{t_1\}_{k_1}\}^{a_1}, \dots, \{\{t_n\}_{k_n}\}^{a_n} \rangle$  and, for all  $i \in \tilde{n}$ ,  $T \vdash t_i$ ,  $T \vdash k_i$  and  $t_i$  and  $k_i$  are in normal form,

then  $T \vdash nf(t)$ .

*Proof.* We assume w.l.o.g. that  $nf(t) \neq \text{nil}$ , since otherwise the result follows directly from  $\text{nil} \in T$ . We establish the conclusion of each of the cases.

- (i)  $t = \langle t_1, \dots, t_n \rangle$  and, for all  $i \in \tilde{n}$ ,  $T \vdash t_i$  and  $t_i$  is in normal form. By the assumption that  $nf(t) \neq \text{nil}$ , we have  $nf(t) = \langle t_{i_1}, \dots, t_{i_m} \rangle$  for some  $m \geq 1$  such that  $\{i_1, \dots, i_m\} = \{j \in \tilde{n} \mid t_j \neq \text{nil}\}$  and  $1 \leq i_1 < \dots < i_m \leq n$ . Hence, we clearly have  $T \vdash nf(t)$ .
- (ii)  $t = \langle h^{a_1}(t_1), \dots, h^{a_n}(t_n) \rangle$  and, for all  $i \in \tilde{n}$ , we have  $a_i \geq 0$ ,  $T \vdash t_i$  and  $t_i$  are in normal form. By the assumption that  $nf(t) \neq \text{nil}$ , we have  $nf(t) = \langle h^{a_{i_1}}(t_1), \dots, h^{a_{i_m}}(t_m) \rangle$  for some  $m \geq 1$  such that  $\{i_1, \dots, i_m\} = \{j \in \tilde{n} \mid t_j \neq \text{nil}\}$  and  $1 \leq i_1 < \dots < i_m \leq n$ . Thus, we clearly have  $T \vdash nf(t)$ .
- (iii)  $t = \langle \{\{t_1\}_{k_1}\}^{a_1}, \dots, \{\{t_n\}_{k_n}\}^{a_n} \rangle$ , and, for all  $i \in \tilde{n}$ , we have  $a_i \geq 0$ ,  $T \vdash t_i$ ,  $T \vdash k_i$ , and  $t_i$  and  $k_i$  are in normal form. By the assumption that  $nf(t) \neq \text{nil}$ , we have  $nf(t) = \langle \{\{t_{i_1}\}_{k_{i_1}}\}^{c_{i_1}}, \dots, \{\{t_{i_m}\}_{k_{i_m}}\}^{c_{i_m}} \rangle$  for some  $m \geq 1$  such that  $\{i_1, \dots, i_m\} = \{j \in \tilde{n} \mid t_j \neq \text{nil}\}$ ,  $1 \leq i_1 < \dots < i_m \leq n$ , and, for all  $j \in \tilde{m}$ ,  $c_{i_j} = a_{i_j}$  if  $k_{i_j} \neq \text{nil}$  and  $c_{i_j} = 0$  otherwise. Thus, we clearly have  $T \vdash nf(t)$ .

This completes the proof of the lemma.  $\square$

**Lemma 17.** *Let  $\Gamma$  be a typing environment and  $F_f = (f, \Gamma_f, E_f)$  be a type-based abstraction. Let  $t, u \in \mathcal{T}$  be normal-form terms such that  $\text{split}(u) \subseteq \text{split}(t)$ . Then we have*

$$\text{split}(f(u)) \subseteq \text{split}(f(t)) \cup \{\text{nil}\}.$$

*Proof.* Note that  $\text{split}(u) \subseteq \text{split}(t)$  implies  $\text{split}(nf(u)) \subseteq \text{split}(nf(t))$  and that  $f(nf(v)) = f(v)$  for all  $v \in \mathcal{T}$ . Hence, it suffices to prove this lemma for the case that  $u, t$  are normal-form. We proceed by induction on  $|u| + |t|$ .

- If  $|\text{split}(u)| + |\text{split}(t)| = 2$  then  $\text{split}(u) \subseteq \text{split}(t)$  implies that  $u = t$ . Thus the lemma holds for this case.

- Now we assume that  $|split(u)| + |split(t)| > 2$ . There are two cases.
  - If  $u$  is not a pair then  $split(u) = \{u\}$ . Hence we have

$$u \in split(t) \tag{7}$$

Since  $|split(u)| + |split(t)| > 2$ , we have  $t = \langle u_1, u_2 \rangle$ . Hence by Lemma 14, we have  $f(t) = nf(\widehat{f}(\bar{v}))$  for some vector  $\bar{v}$  such that  $split(t) = split(\bar{v})$ . By (7), there is  $t' \in set(\bar{v})$  such that  $u \in split(t')$ . Moreover, we have  $|t'| < |t|$ . Thus by induction hypothesis, we have

$$split(f(u)) \subseteq split(f(t')) \cup \{\text{nil}\}.$$

Since  $split(f(t')) \subseteq split(f(t)) \cup \{\text{nil}\}$ , this implies that

$$split(f(u)) \subseteq split(f(t)) \cup \{\text{nil}\}.$$

- If  $u = \langle u_1, u_2 \rangle$  then by Lemma 14, we have that  $f(u) = nf(\widehat{f}(\bar{r}))$  for some vector  $\bar{r}$  of length  $m$  such that  $split(u) = split(\bar{r})$ . Since  $split(u) \subseteq split(t)$ , we have  $split(r_i) \subseteq split(t)$  for all  $i \in \tilde{m}$ . Moreover, we also have that  $|r_i| < |u|$ . Hence by induction hypothesis, we have  $split(f(r_i)) \subseteq split(f(t)) \cup \{\text{nil}\}$ . Therefore, we obtain  $split(f(u)) \subseteq split(f(t)) \cup \{\text{nil}\}$  as required.

This completes the proof of the lemma.  $\square$

The following lemma is an immediate corollary of Lemma 17.

**Lemma 18.** *Let  $\Gamma$  be a typing environment and  $F_f = (f, \Gamma_f, E_f)$  a type-based abstraction. Then for all  $t \in \mathcal{T}$  in normal form and all  $u \in split(t)$ , we have*

$$f(t), \text{nil} \vdash f(u)$$

The following lemma shows that if the intruder learns all the transformed components of a term, he can also learn the transformed term.

**Lemma 19.** *Let  $\Gamma$  be a typing environment and  $F_f = (f, \Gamma_f, E_f)$  a type-based abstraction. Let  $T \cup \{u\} \subseteq \mathcal{T}$  such that  $\text{nil} \in T$  and  $u$  is in normal form. Suppose  $T \vdash f(t)$  for all  $t \in split(u)$ . Then  $T \vdash f(u)$ .*

*Proof.* We prove the lemma by induction on the size of  $u$ . If  $u$  is not a pair then  $split(u) = \{u\}$  and  $T \vdash f(u)$  follows immediately from the assumption. Otherwise,  $u = \langle u_1, u_2 \rangle$ . Then, by Lemma 14, we have

$$f(u) = nf(\widehat{f}(\bar{r}))$$

for some vector  $\bar{r} = [r_1, \dots, r_m]$  such that  $split(\bar{r}) = split(u)$  and  $set(\bar{r}) \subseteq subs(u) \setminus \{u\}$ . Since  $u$  is in normal form, so are the  $r_i$ . Hence, by Lemma 16(i), the desired  $T \vdash f(u)$  follows from  $T \vdash f(r_i)$  for all  $i \in \tilde{m}$ . Let  $i \in \tilde{m}$ . By assumption and since  $split(r_i) \subseteq split(u)$ , we have  $T \vdash f(t)$  for all  $t \in split(r_i)$ . Since  $r_i$  is in normal form and  $r_i \in subs(u) \setminus \{u\}$  we obtain  $T \vdash f(r_i)$  from the induction hypothesis as required.  $\square$

The following lemma is a consequence of the two previous lemmas.

**Lemma 20.** *Let  $\Gamma$  be a typing environment and  $F_f = (f, \Gamma_f, E_f)$  a type-based abstraction. Then, for all  $t, u \in \mathcal{T}$  in normal form, we have  $\text{split}(t) \subseteq \text{split}(u)$  implies  $f(u), \text{nil} \vdash f(t)$ .*

*Proof.* By Lemma 18, we have  $f(u), \text{nil} \vdash f(p)$  for all  $p \in \text{split}(u)$ . Moreover, since  $\text{split}(t) \subseteq \text{split}(u)$ , we have  $f(u), \text{nil} \vdash f(q)$  for all  $q \in \text{split}(t)$ . Hence, by Lemma 19, we have  $f(u), \text{nil} \vdash f(t)$ .  $\square$

**Lemma 21 (Local intruder derivations).** *Let  $D$  be a derivation tree for  $T \vdash t$  of minimal size. Then this tree contains only terms in  $\text{subs}(T \cup \{t\})$ . In particular, if  $T$  and  $t$  are in normal form then so are all terms on  $D$ .*

**Theorem (Deducibility preservation; Justification of Theorem 1).** *Let  $F_f = (f, \Gamma_f, E_f)$  a type-based abstraction. Let  $T \cup \{t\} \subseteq \mathcal{N}$  be a set of ground network messages in normal form. Then  $T \vdash t$  implies  $f(T), \text{nil} \vdash f(t)$ .<sup>2</sup>*

*Proof.* By induction on the derivation  $D$  of  $T \vdash t$ , depending on the last rule  $R$  applied in this derivation. We assume without loss of generality that  $D$  is of minimal size.

- $R = (\text{Ax})$ . In this case, we have  $t \in T$ . Therefore,  $f(t) \vdash f(T)$  and thus  $f(T), \text{nil} \vdash f(t)$  as required.
- $R = (\text{Comp})$  for some  $g \in \Sigma_{\text{pub}}$  of arity  $n$ . In this case,  $t = g(u_1, \dots, u_n)$  and the rule's premises are  $T \vdash u_i$  for  $i \in \tilde{n}$ . The induction hypotheses are  $f(T), \text{nil} \vdash f(u_i)$  for  $i \in \tilde{n}$ . We show that

$$f(T), \text{nil} \vdash f(g(u_1, \dots, u_n)).$$

We reason by case analysis on the public constructor  $f$ . We assume without loss of generality that  $f(g(u_1, \dots, u_n)) \neq \text{nil}$ , since the result holds trivially otherwise.

- $t = \text{pk}(u)$ . The rule's premise is  $T \vdash u$ . From Lemma 14, we have

$$f(\text{pk}(u)) = nf(\text{pk}(f(u))).$$

Since we assume  $f(\text{pk}(u)) \neq \text{nil}$ , we must have  $f(\text{pk}(u)) = \text{pk}(f(u))$ . Applying rule  $(\text{Comp})$  to the induction hypothesis  $f(T), \text{nil} \vdash f(u)$  thus yields  $f(T), \text{nil} \vdash f(\text{pk}(u))$  as required.

- $t = h(u)$  for  $h \in \Sigma_u$ . The rule's premise is  $T \vdash u$ . By Lemma 14 we have

$$f(h(u)) = nf(\langle h^{a_1}(\widehat{f}(\overline{v}_1)), \dots, h^{a_l}(\widehat{f}(\overline{v}_l)) \rangle)$$

for some  $l > 0$  and vectors  $\overline{v}_1, \dots, \overline{v}_l$  such that, for all  $i \in \tilde{l}$ , we have  $\text{split}(\overline{v}_i) \subseteq \text{split}(u)$  and  $a_i \geq 0$ . Since  $t$  is in normal form so are all

---

<sup>2</sup> Note that this holds with respect to any typing environment.

elements of all  $\bar{v}_i$ 's. By Lemma 16(ii), to establish  $f(T), \text{nil} \vdash f(h(u))$  it suffices to show that  $f(T), \text{nil} \vdash \widehat{f}(\bar{v}_i)$  for all  $i \in \tilde{l}$ .

Let  $i \in \tilde{l}$ . For each component  $v_{ij}$  of  $\bar{v}_i$ , we use  $\text{split}(v_{ij}) \subseteq \text{split}(u)$  to apply Lemma 20 and obtain  $f(u), \text{nil} \vdash f(v_{ij})$ . Combined with the induction hypothesis  $f(T), \text{nil} \vdash f(u)$  this yields  $f(u), \text{nil} \vdash \widehat{f}(\bar{v}_i)$  as required.

- $t = \langle u_1, u_2 \rangle$ . The rule's premises are  $T \vdash u_i$  for  $i = 1, 2$ . By Lemma 14, we have

$$f(\langle u_1, u_2 \rangle) = nf(\widehat{f}(\bar{r}))$$

for some vector  $\bar{r} = [r_1, \dots, r_m]$  such that  $\text{set}(\bar{r}) = \text{split}(t)$ . Since  $t$  is in normal form, so are all the  $r_i$ 's. Hence, we use Lemma 16(i) to reduce  $f(T), \text{nil} \vdash f(\langle u_1, t_2 \rangle)$  to showing that  $f(T), \text{nil} \vdash f(r_i)$  for all  $i \in \tilde{m}$ .

Let  $i \in \tilde{m}$ . We have  $\text{split}(r_i) \subseteq \text{split}(t) = \text{split}(u_1) \cup \text{split}(u_2)$ . Let  $s \in \text{split}(r_i)$ . Then there is a  $j \in \{1, 2\}$  such that  $s \in \text{split}(u_j)$ . By Lemma 18, we have  $f(u_j), \text{nil} \vdash f(s)$ . Therefore, for all  $s \in \text{split}(r_i)$ , we have  $f(u_1), f(u_2), \text{nil} \vdash f(s)$  and, using the induction hypotheses  $f(T), \text{nil} \vdash f(s)$ . Hence, by Lemma 19, we have  $f(T), \text{nil} \vdash f(r_i)$  as required.

- $t = \{u\}_k$ . The rule's premises are  $T \vdash u$  and  $T \vdash k$ . By Lemma 14, we have

$$f(\{u\}_k) = nf(\langle \{\widehat{f}(\bar{s}_1)\}_{\widehat{f}(\bar{v}_1)}^{a_1}, \dots, \{\widehat{f}(\bar{s}_d)\}_{\widehat{f}(\bar{v}_d)}^{a_d} \rangle)$$

for some  $d > 0$  and vectors  $\bar{s}_1, \dots, \bar{s}_d$  and  $\bar{v}_1, \dots, \bar{v}_d$  such that, for all  $i \in \tilde{d}$ , we have  $\text{split}(\bar{s}_i) \subseteq \text{split}(u)$ ,  $\text{split}(\bar{v}_i) \subseteq \text{split}(k)$ , and  $a_i \geq 0$ . Since  $t$  is in normal form so are all the components of the  $\bar{s}_i$  and  $\bar{v}_i$ . We thus use Lemma 16(iii) to reduce  $f(T), \text{nil} \vdash f(\{u\}_k)$  to showing, for all  $i \in \tilde{d}$ ,  $f(T), \text{nil} \vdash \widehat{f}(\bar{s}_i)$  and  $f(T), \text{nil} \vdash \widehat{f}(\bar{v}_i)$ .

Let  $i \in \tilde{d}$  and let  $s_{ij}$  be a component of  $\bar{s}_i$ . Since we have  $\text{split}(s_{ij}) \subseteq \text{split}(u)$ , we obtain  $f(u), \text{nil} \vdash f(s_{ij})$  from Lemma 20. Hence, clearly  $f(u), \text{nil} \vdash \widehat{f}(\bar{s}_i)$ , which in combination with the induction hypothesis  $f(T), \text{nil} \vdash f(u)$  yields the desired  $f(T), \text{nil} \vdash \widehat{f}(\bar{s}_i)$ . By an analogous argument, we can also show that  $f(T), \text{nil} \vdash \widehat{f}(\bar{v}_i)$  as required.

- $R = (\text{Proj}_i)$  for  $i \in \{1, 2\}$ . The rule's premise is  $T \vdash \langle u_1, u_2 \rangle$ . By Lemma 21, we have that  $\langle u_1, u_2 \rangle$  and hence  $u_i$  are in normal form. By Lemma 20, we have  $f(\langle u_1, u_2 \rangle), \text{nil} \vdash f(u_i)$  and from the induction hypothesis we have  $f(T), \text{nil} \vdash f(\langle u_1, u_2 \rangle)$ . Hence,  $f(T), \text{nil} \vdash f(u_i)$  as required.
- $R = (\text{Adec})$ . The rule's premises are  $T \vdash \{u\}_{\text{pk}(k)}$  and  $T \vdash \text{pri}(k)$ . We have to show  $f(T), \text{nil} \vdash f(u)$  from the induction hypotheses  $f(T), \text{nil} \vdash f(\{u\}_k)$  and  $f(T), \text{nil} \vdash f(\text{pri}(k))$ . By Lemma 21, we have that both  $\{u\}_{\text{pk}(k)}$  and  $\text{pri}(k)$  are in normal form. By Lemma 14, we have

$$f(\{u\}_{\text{pk}(k)}) = nf(\langle \{\widehat{f}(\bar{s}_1)\}_{\widehat{f}(\text{pk}(k))}^{a_1}, \dots, \{\widehat{f}(\bar{s}_d)\}_{\widehat{f}(\text{pk}(k))}^{a_d} \rangle)$$

for some  $d > 0$  and vectors  $\bar{s}_1, \dots, \bar{s}_d$  such that  $\bigcup_{i=1}^d \text{split}(\bar{s}_i) = \text{split}(u)$ , and  $a_i \geq 0$  for all  $i \in \tilde{d}$ . Note that all components of each  $\bar{s}_i$  are in normal form.

By Lemma 19, to establish our goal  $f(T), \text{nil} \vdash f(u)$  it is sufficient to show that  $f(T), \text{nil} \vdash f(w)$  for all  $i \in \tilde{d}$  and components  $w \in \text{set}(\overline{s_i})$ . Let  $i \in \tilde{d}$  and  $v \in \text{set}(\overline{s_i})$ . If  $f(w) = \text{nil}$  then we trivially have  $f(T), \text{nil} \vdash f(w)$ . Otherwise, we know from the definition of  $nf$  that  $t = nf(\{\widehat{f}(\overline{s_i})\}_{\widehat{f}(\text{pk}(k))}^{a_i}) \neq \text{nil}$  and is therefore a tuple component of  $f(\{u\}_{\text{pri}(k)})$ . Clearly, we have  $f(T), \text{nil} \vdash t$ . Now we distinguish two cases:

(a) If  $nf(f(\text{pk}(k))) \neq \text{nil}$  and  $a_i > 0$  then we have  $t = \{nf(\widehat{f}(\overline{s_i}))\}_{\widehat{f}(\text{pri}(k))}^{a_i}$ .

From  $f(T), \text{nil} \vdash t$  and the induction hypothesis  $f(T), \text{nil} \vdash f(\text{pri}(k))$ , we derive  $f(T), \text{nil} \vdash nf(\widehat{f}(\overline{s_i}))$  by  $a_i$ -fold decryption. Therefore, we have  $f(T), \text{nil} \vdash f(w)$  as required since  $w \in \text{set}(\overline{s_i})$  and  $f(w) \neq \text{nil}$ .

(b) Otherwise, we directly have  $t = nf(\widehat{f}(\overline{s_i}))$  and the conclusion follows as above.

- $R = (\text{Sdec})$ . The rule's premises are  $T \vdash \{u\}_k$  and  $T \vdash k$ . We have to show  $f(T), \text{nil} \vdash f(u)$  from the induction hypotheses  $f(T), \text{nil} \vdash f(\{u\}_k)$  and  $f(T), \text{nil} \vdash f(k)$ . By Lemma 21, we have that both  $\{u\}_k$  and  $k$  are in normal form. By Lemma 14, we have

$$f(\{u\}_{\text{pk}(k)}) = nf(\langle \{\widehat{f}(\overline{s_1})\}_{\widehat{f}(\overline{v_1})}^{a_1}, \dots, \{\widehat{f}(\overline{s_d})\}_{\widehat{f}(\overline{v_d})}^{a_d} \rangle)$$

for some  $d > 0$  and vectors  $\overline{s_1}, \dots, \overline{s_d}$  and  $\overline{v_1}, \dots, \overline{v_d}$  such that  $\bigcup_{i=1}^d \text{split}(\overline{s_i}) = \text{split}(u)$ , and for all  $i \in \tilde{d}$ ,  $\text{split}(\overline{v_i}) \subseteq \text{split}(k)$  and  $a_i \geq 0$ . Note that all components of each  $\overline{s_i}$  and  $\overline{v_i}$  are in normal form.

This case is very similar to the previous one up to case (a), which we consider in some detail. We thus have to show that  $f(T), \text{nil} \vdash f(w)$  for an arbitrary but fixed  $w \in \text{set}(\overline{s_i})$  and  $i \in \tilde{d}$  such that  $f(w) \neq \text{nil}$ . We know that  $f(T), \text{nil} \vdash t$  for  $t = nf(\{\widehat{f}(\overline{s_i})\}_{\widehat{f}(\overline{v_i})}^{a_i}) \neq \text{nil}$  and we consider the case (a) where  $nf(\widehat{f}(\overline{v_i})) \neq \text{nil}$  and  $a_i > 0$ . From  $\text{split}(\overline{v_i}) \subseteq \text{split}(k)$  and the induction hypothesis  $f(T), \text{nil} \vdash f(k)$ , we derive  $f(T), \text{nil} \vdash \widehat{f}(\overline{v_i})$  using Lemma 20. Hence, we obtain  $f(T), \text{nil} \vdash nf(\widehat{f}(\overline{s_i}))$  by  $a_i$ -fold decryption of  $t$ . Finally, since  $w \in \text{set}(\overline{s_i})$  and  $f(w) \neq \text{nil}$ , we obtain  $f(T), \text{nil} \vdash f(w)$  as required.

This completes the proof of the theorem. □

## E Substitution property

**Theorem (Substitution property; Justification of Theorem 2).** *Let  $\Gamma$  be a typing environment,  $F_f = (f, \Gamma_f, E_f)$  a type-based abstraction, and  $\theta$  a nil-free substitution that is well-typed with respect to  $\Gamma$ . Then, for all terms  $t \in \mathcal{T}$  such that  $t \in \text{udom}(F_f, \Gamma)$ , we have  $f(t\theta) = \text{nf}(f(t)f(\theta))$ .*

*Proof.* To reduce notational clutter in this proof, we rename the functions  $f$  and  $f_{\text{rec}}$  of Program 1 to  $f_0$  and  $f$ , respectively. Hence, we have to show that  $f_0(t\theta) = \text{nf}(f_0(t)f_0(\theta))$ . Since  $\theta$  is nil-free, by Lemma 4, we have  $f_0(t\theta) = f(\text{nf}(t\theta)) = f(\text{nf}(t)\theta)$ . Moreover, we also have  $f_0(\theta) = f(\theta)$  and  $f_0(t) = f(\text{nf}(t))$ . Therefore, our proof goal is equivalent to  $f(\text{nf}(t)\theta) = \text{nf}(f(\text{nf}(t))f(\theta))$ . Thus, it is enough to prove the statement  $f(t\theta) = \text{nf}(f(t)f(\theta))$  for all normal-form terms  $t$ . We do this by induction on the size of  $t$ .

Suppose  $E_f = [f(p_1) = u_1, \dots, f(p_n) = u_n]$  and let  $t$  be a term such that  $t \in \text{udom}(F_f, \Gamma)$ . We distinguish two cases. If  $\Gamma(t) = \text{msg}$  then  $t$  is a variable and thus  $f(t) = t$  using the final identity fall-back clause of  $E_f^0$ . It follows that  $\text{nf}(f(t)f(\theta)) = \text{nf}(tf(\theta)) = tf(\theta) = f(t\theta)$  as required. Otherwise, since  $t$  is normal-form and  $t \in \text{udom}(F_f, \Gamma)$ , we have  $\Gamma(t) \in \Pi_f \downarrow$ . We abbreviate  $\tau = \Gamma(t)$  and  $\tau' = \Gamma(t\theta)$ . Since  $\theta$  is well-typed with respect to  $\Gamma$  and we have  $\tau' \preceq \tau$  by Lemma 11. Since  $\tau \in \Pi_f \downarrow$  we know by Lemma 13 that there is a unique  $i \in \tilde{n}$  and substitutions  $\theta'$  and  $\theta''$  such that  $p_i\theta' = t$  and  $\tau \preceq \pi_i$  as well as  $p_i\theta'' = t\theta$  and  $\tau' \preceq \pi_i$ . Then we also have  $t\theta = p_i\theta'' = p_i\theta'\theta$ . Hence, by Program 1 (modulo renamings), we have

$$f(t) = \text{nf}_{\text{nil}}(u_i[f/f_0, \text{nil}/\text{Nil}]\theta') \quad \text{and} \quad f(t\theta) = \text{nf}_{\text{nil}}(u_i[f/f_0, \text{nil}/\text{Nil}]\theta'\theta).$$

We distinguish three base cases.

- $u_i = \text{Nil}$  and  $t$  is a variable or an atom. Then we have  $f(t) = \text{nil} = f(t\theta)$ . Hence,  $f(t\theta) = \text{nf}(f(t)f(\theta))$ .
- $u_i = p_i$  and  $t = a$  is an atom. Then we obtain that  $f(a\theta) = f(a) = a = \text{nf}(a) = \text{nf}(af(\theta)) = \text{nf}(f(a)f(\theta))$  as required.
- $u_i = p_i$  and  $t = X$  is a variable. Then we have  $f(X) = X$ . Let us consider two cases:
  - If  $X \in \text{dom}(\theta)$ , then we have

$$f(X\theta) = \text{nf}(f(X\theta)) = \text{nf}(Xf(\theta)) = \text{nf}(f(X)f(\theta)).$$

- If  $X \notin \text{dom}(\theta)$ , then since  $\text{dom}(f(\theta)) \subseteq \text{dom}(\theta)$ , we have  $X \notin \text{dom}(f(\theta))$ . Hence, we have

$$f(X\theta) = f(X) = X = \text{nf}(X) = \text{nf}(Xf(\theta)) = \text{nf}(f(X)f(\theta)).$$

For the inductive cases, note that since  $t$  is in normal form and recursive calls of  $f$  have subterms of  $t$  as arguments by Lemma 14, these subterms are also in normal form. Moreover, since  $t \in \text{udom}(F_f, \Gamma)$ , we also have  $u \in \text{udom}(F_f, \Gamma)$  for each term  $u$  occurring as the argument of a recursive call of  $f$  in the computation

of  $f(t)$ . This enables the application of the induction hypotheses (IH) below. We distinguish the following cases. (Note that the substitution  $\text{nil}/\text{Nil}$  is only relevant for the case where  $t$  is a hash.)

- $t = g(v)$  for some  $g \in \{\text{pk}, \text{pri}\}$ . Then we have  $p_i = g(q)$ ,  $v = q\theta'$ , and  $u_i = g(f(q))$ . We prove this case by the following calculation:

$$\begin{aligned} f(t\theta) &= nf(g(f(v\theta))) \\ &= nf(g(f(v)f(\theta))) && \text{by IH and Lemma 2} \\ &= nf(g(f(v))f(\theta)) \\ &= nf(nf(g(f(v)))f(\theta)) && \text{by Lemma 5} \\ &= nf(f(t)f(\theta)) && \text{since } f(t) = nf(g(f(v))) \end{aligned}$$

- $t = \text{sh}(v_1, v_2)$ . By a similar reasoning as above, we have  $f(t\theta) = nf(f(t)f(\theta))$ .
- $t = h(v)$  for some  $h \in \Sigma_u$ . Then  $p_i = g(q)$  and  $v = q\theta'$ . There are two cases.
  - If  $u_i = \langle h^{a_1}(\widehat{f}(\overline{q}_1)), \dots, h^{a_l}(\widehat{f}(\overline{q}_l)) \rangle$  for some  $l \geq 1$  and vectors  $\overline{q}_1, \dots, \overline{q}_l$  such that  $\text{set}(\overline{q}_i) \subseteq \text{split}(q)$  and  $a_i \geq 0$  for all  $i \in \tilde{l}$ . Let  $\overline{v}_i = \overline{q}_i\theta'$  for  $i \in \tilde{l}$ . Then we calculate:

$$\begin{aligned} f(t\theta) &= nf(\langle h^{a_1}(\widehat{f}(\overline{v}_1\theta)), \dots, h^{a_l}(\widehat{f}(\overline{v}_l\theta)) \rangle) \\ &= nf(\langle h^{a_1}(\widehat{f}(\overline{v}_1)f(\theta)), \dots, h^{a_l}(\widehat{f}(\overline{v}_l)f(\theta)) \rangle) && \text{by IH and Lemma 2} \\ &= nf(\langle h^{a_1}(\widehat{f}(\overline{v}_1)), \dots, h^{a_l}(\widehat{f}(\overline{v}_l)) \rangle f(\theta)) \\ &= nf(nf(\langle h^{a_1}(\widehat{f}(\overline{v}_1)), \dots, h^{a_l}(\widehat{f}(\overline{v}_l)) \rangle) f(\theta)) && \text{by Lemma 5} \\ &= nf(f(t)f(\theta)) \end{aligned}$$

- If  $u_i = \text{Nil}$  then  $f(t) = f(t\theta) = \text{nil}$ . Thus, the conclusion holds trivially.
- $t = \langle w_1, w_2 \rangle$ . Here, we have  $p_i = \langle q_1, q_2 \rangle$ ,  $w_1 = q_1\theta'$ ,  $w_2 = q_2\theta'$ , and  $u_i = \widehat{f}(\overline{r})$  for some vector  $\overline{r}$  such that  $\text{set}(\overline{r}) = \text{split}(\langle p_1, p_2 \rangle)$ . Since  $\overline{s} = \overline{r}\theta'$ , we have

$$\begin{aligned} f(t\theta) &= nf(\widehat{f}(\overline{s}\theta)) \\ &= nf(\widehat{f}(\overline{s})f(\theta)) && \text{by IH and Lemma 2} \\ &= nf(nf(\widehat{f}(\overline{s}))f(\theta)) && \text{by Lemma 5} \\ &= nf(f(t)f(\theta)) \end{aligned}$$

- $t = \{w\}_k$ . In this case, we have  $p_i = \{q\}_r$ ,  $w = q\theta'$ ,  $k = r\theta'$ , and

$$u_i = \langle \{\widehat{f}(\overline{q}_1)\}_{\widehat{f}(\overline{r}_1)}^{a_1}, \dots, \{\widehat{f}(\overline{q}_d)\}_{\widehat{f}(\overline{r}_d)}^{a_d} \rangle$$

for some  $d > 0$  and vectors  $\overline{q}_1, \dots, \overline{q}_d$  and  $\overline{r}_1, \dots, \overline{r}_d$  such that  $\bigcup_{i=1}^d \text{set}(\overline{q}_i) = \text{split}(q)$  and, for all  $i \in \tilde{d}$ , we have  $\text{set}(\overline{r}_i) \subseteq \text{split}(r)$  and  $a_i \geq 0$ . Let  $\overline{s}_i = \overline{q}_i\theta'$  and  $\overline{v}_i = \overline{r}_i\theta'$  for all  $i \in \tilde{d}$ . Then we calculate:

$$\begin{aligned} f(t\theta) &= nf(\langle \{\widehat{f}(\overline{s}_1\theta)\}_{\widehat{f}(\overline{v}_1\theta)}^{a_1}, \dots, \{\widehat{f}(\overline{s}_d\theta)\}_{\widehat{f}(\overline{v}_d\theta)}^{a_d} \rangle) \\ &= nf(\langle \{\widehat{f}(\overline{s}_1)f(\theta)\}_{\widehat{f}(\overline{v}_1)f(\theta)}^{a_1}, \dots, \{\widehat{f}(\overline{s}_d)f(\theta)\}_{\widehat{f}(\overline{v}_d)f(\theta)}^{a_d} \rangle) && \text{by IH, Lem. 2} \\ &= nf(\langle \{\widehat{f}(\overline{s}_1)\}_{\widehat{f}(\overline{v}_1)}^{a_1}, \dots, \{\widehat{f}(\overline{s}_d)\}_{\widehat{f}(\overline{v}_d)}^{a_d} \rangle f(\theta)) \\ &= nf(nf(\langle \{\widehat{f}(\overline{s}_1)\}_{\widehat{f}(\overline{v}_1)}^{a_1}, \dots, \{\widehat{f}(\overline{s}_d)\}_{\widehat{f}(\overline{v}_d)}^{a_d} \rangle) f(\theta)) && \text{by Lemma 5} \\ &= nf(f(t)f(\theta)) \end{aligned}$$

This completes the proof of the theorem.  $\square$

## F Deducibility preservation for open terms

In this section, we establish deducibility preservation for open terms. This result is crucial to for reachability preservation.

### F.1 Lemmas about substitutions

We show that deducibility is preserved by substitution.

**Lemma 22.** *Let  $T \subseteq \mathcal{T}(V, \Sigma_0)$  be a set of term,  $t$  a term, and  $\sigma$  a substitution. Then  $T \vdash t$  implies  $T\sigma \vdash t\sigma$ .*

*Proof.* Routine induction on the derivation of  $T \vdash t$ . □

Next, we specialize this lemma a little.

**Lemma 23.** *Let  $S \cup T \cup \{t, u, a\} \subseteq \mathcal{T}(V, \Sigma_0)$  be a set of terms such that  $a$  is an atom. Then  $S, T \vdash t$  and  $T \vdash T[u/a]$  imply  $S[u/a], T \vdash t[u/a]$ .*

*Proof.* Suppose  $S, T \vdash t$  and  $T \vdash T[u/a]$ . Treating the atom  $a$  like a variable and applying Lemma 22 to the first assumption yields  $S[u/a], T[u/a] \vdash t[u/a]$ . The conclusion then follows from the second assumption. □

**Lemma 24.** *Let  $t$  and  $u$  be terms,  $a$  an atomic term, and  $\sigma$  a substitution such that  $\text{vars}(t) \cap \text{dom}(\sigma) = \emptyset$ . Then  $(u\sigma)[t/a] = (u[t/a])\sigma[t/a]$ .*

*Proof.* We prove this lemma by induction on  $u$ .

- If  $u$  is an atom then  $(u\sigma)[t/a] = u[t/a]$ . Moreover, since  $\text{vars}(t) \cap \text{dom}(\sigma) = \emptyset$ , it follows that  $(u[t/a])\sigma[t/a] = u[t/a]$ . Thus the lemma holds for this case.
- If  $u$  is a variable then since  $a$  is atomic, we have  $u[t/a] = u$ . Therefore, we have  $(u\sigma)[t/a] = u(\sigma[t/a]) = (u[t/a])\sigma[t/a]$ .
- If  $u = g(u')$  for  $g \in \Sigma^1$  then we have

$$\begin{aligned}
 (u\sigma)[t/a] &= g((u'\sigma)[t/a]) && \text{since } a \text{ is atomic} \\
 &= g((u'[t/a])\sigma[t/a]) && \text{by induction hypothesis} \\
 &= (g(u')[t/a])\sigma[t/a] && \text{since } a \text{ is atomic} \\
 &= (u[t/a])\sigma[t/a]
 \end{aligned}$$

- Similarly for the cases that  $u = f(u_1, u_2)$  for  $f \in \Sigma^2$ , we obtain that  $(u\sigma)[t/a] = (u[t/a])\sigma[t/a]$ .

This completes the proof of the lemma. □

## F.2 Lemmas about abstracted types

**Lemma 25.** *Let  $F_f = (f, \Gamma_f, E_f)$  be a type-based abstraction and  $\tau, \tau'$  be types such that  $\tau$  is atomic,  $\tau \neq \text{msg}$ ,  $\tau \in \Pi_f \downarrow$ , and  $\tau' \preceq \tau$ . Then one of the following holds:*

1.  $f(\tau') = f(\tau) = \gamma_{\text{nil}}$ , or
2.  $f(\tau') = \tau'$  and  $f(\tau) = \tau$ .

*Proof.* Since  $F_f$  is pattern-disjoint and  $\tau \in \Pi_f \downarrow$ , we know by Lemma 13 that there is an unique pattern  $f(p) = u$  in  $E_f$  such that  $\tau \preceq \Gamma_f(p)$ ,  $p\theta = \tau$ , and  $p\theta' = \tau'$  for some  $\theta$  and  $\theta'$ . Since  $\tau$  is atomic, by Definition 5,  $p$  is some pattern variable  $X$  and  $u \in \{X, \text{Nil}\}$ . Hence, we have (1)  $f(\tau) = f(\tau') = \gamma_{\text{nil}}$  for  $u = \text{Nil}$  and (2)  $f(\tau) = X\theta = \tau$  and  $f(\tau') = X\theta' = \tau'$  for  $u = X$ , as required.  $\square$

**Lemma 26.** *Let  $F_f = (f, \Gamma_f, E_f)$  be a type-based abstraction and  $\tau, \tau'$  be types such that  $\tau$  is atomic,  $\tau \in \Pi_f \downarrow \cup \{\text{msg}\}$ , and  $\tau' \preceq \tau$ . Then we have  $f(\tau') \preceq f(\tau)$ .*

*Proof.* Since  $\tau$  is atomic, so is  $\tau'$  by Lemma 7. If  $\tau = \text{msg}$  the lemma holds trivially. Otherwise, it holds by Lemma 25.  $\square$

**Lemma 27.** *Let  $\Gamma$  be a typing environment,  $\sigma$  be a ground substitution that is well-typed with respect to  $\Gamma$ , and  $F_f = (f, \Gamma_f, E_f)$  be a type-based abstraction such that  $\text{dom}(\Gamma) \subseteq \text{udom}(F_f, \Gamma)$ . Then  $f(\sigma)$  is well-typed with respect to  $f(\Gamma)$ .*

*Proof.* Let  $X \in \text{dom}(f(\sigma))$ . Then we have  $X \in \text{dom}(\sigma)$  and  $f(X) = X$ . Since  $\sigma$  is well-typed with respect to  $\Gamma$ , we know that  $\Gamma(X)$  and  $\Gamma(X\sigma)$  are defined, and  $\Gamma(X\sigma) \preceq \Gamma(X)$ . Since  $\text{dom}(\Gamma) \subseteq \text{udom}(F_f, \Gamma)$ , we have  $\Gamma(X) \in \Pi_f \downarrow \cup \{\text{msg}\}$ . Hence by Lemma 26, we have  $f(\Gamma(X\sigma)) \preceq f(\Gamma(X))$ . Moreover, by Lemma 3, we have

$$\begin{aligned} f(\Gamma(X\sigma)) &= f(\Gamma)(f(X\sigma)) = f(\Gamma)(Xf(\sigma)), \\ f(\Gamma(X)) &= f(\Gamma)(f(X)) = f(\Gamma)(X). \end{aligned}$$

Therefore, we have  $f(\Gamma)(Xf(\sigma)) \preceq f(\Gamma)(X)$ . Thus  $f(\sigma)$  is well-typed with respect to  $f(\Gamma)$ .  $\square$

**Lemma 28.** *Let  $\Gamma$  be a typing environment and  $\sigma$  be a ground substitution that is well-typed with respect to  $\Gamma$ . Then for all  $X \in \text{dom}(\sigma)$  such that  $X\sigma = \text{nil}$ , we have that  $\Gamma(X) = \text{msg}$ .*

*Proof.* Let  $X \in \text{dom}(\sigma)$  such that  $X\sigma = \text{nil}$ . Since  $\sigma$  is well-typed with respect to  $\Gamma$ , we have that  $\Gamma(X\sigma) \preceq \Gamma(X)$ . Therefore  $\gamma_{\text{nil}} \preceq \Gamma(X)$ . By Lemma 10 and since  $\gamma_{\text{nil}} \notin \text{ran}(\Gamma)$ , we have  $\Gamma(X) = \text{msg}$  as required.  $\square$

## F.3 Deducibility preservation for open terms

**Lemma 29.** *Let  $\Gamma$  be a typing environment,  $\sigma$  be a normal-form ground substitution that is well-typed with respect to  $\Gamma$ ,  $T$  be a set of normal-form terms,  $T_0$  be a set of ground terms, and  $u$  be a normal-form term such that  $\text{nil} \in T_0$  and  $T \cup \{u\}$  is msg-clear with respect to  $\Gamma$ . Suppose that  $\text{nf}(T\sigma), T_0 \vdash \text{nf}(u\sigma)$ . Then we have  $T\sigma, T_0 \vdash u\sigma$ .*

*Proof.* Let  $t \in T \cup \{u\}$  and  $X \in \text{vars}(t)$  such that  $X \in \text{dom}(\sigma)$  and  $X\sigma = \text{nil}$ . Then we derive  $\Gamma(X) = \text{msg}$  using Lemma 28. Since  $t$  is *msg*-clear with respect to  $\Gamma$ , we have that  $X$  is clear in  $t$ . Hence, all variables  $X$  in  $t$  that  $\sigma$  maps to nil are clear in  $t$ . Together with the assumption that  $T$ ,  $u$ , and  $\sigma$  are in normal form, we derive that  $\text{split}(t\sigma) = \text{split}(nf(t\sigma)) \cup \{\text{nil}\}$ .

Hence, we have  $T\sigma \vdash nf(T\sigma)$  and  $nf(u\sigma), T_0 \vdash u\sigma$ . Therefore, from the assumption  $nf(T\sigma), T_0 \vdash nf(u\sigma)$ , we obtain  $T\sigma, T_0 \vdash u\sigma$  as required.  $\square$

**Proposition 4 (Deducibility preservation for open terms).** *Let  $\Gamma$  be a typing environment,  $F_f = (f, \Gamma_f, E_f)$  be a type-based abstraction,  $\sigma$  a nil-free ground substitution that is well-typed with respect to  $\Gamma$ ,  $T$  be a set of normal-form terms, and  $u$  be a normal-form term such that*

- (i)  $IK'_0$  is normal-form and  $f(IK_0) \subseteq IK'_0$ ,
- (ii)  $T \cup \{u\} \cup \text{ran}(\Gamma) \subseteq \text{udom}(F_f, \Gamma)$ , and
- (iii) if  $F_f$  is not nil-free then  $f(T \cup \{u\})$  is *msg*-clear with respect to  $f(\Gamma)$ .

Then for all constants  $c \in \mathcal{C}$ , we have that  $T\sigma, IK_0 \vdash u\sigma$  implies

$$f(T)(f(\sigma)[c/\text{nil}]), IK'_0 \vdash f(u)(f(\sigma)[c/\text{nil}]).$$

*Proof.* We have

$$\begin{array}{lll} T\sigma, IK_0 & \vdash u\sigma & \text{by assumption} \\ \Rightarrow f(T\sigma), f(IK_0) & \vdash f(u\sigma) & \text{by Theorem 1} \\ \Rightarrow f(T\sigma), IK'_0 & \vdash f(u\sigma) & \text{since } f(IK_0) \subseteq IK'_0 \\ \Rightarrow nf(f(T)f(\sigma)), IK'_0 \vdash nf(f(u)f(\sigma)) & & \text{by Theorem 2, assumption (i)} \end{array}$$

Let  $\sigma' = f(\sigma)$ . We now show that  $f(T)\sigma', IK'_0 \vdash f(u)\sigma'$ . If  $\sigma'$  is nil-free then clearly  $f(T)\sigma', IK'_0 \vdash f(u)\sigma'$ . Otherwise, there is  $X \in \text{dom}(\sigma')$  such that  $X\sigma' = \text{nil}$ . This implies  $F_f$  is not nil-free. Therefore, by assumption (iii), we have that  $f(T \cup \{u\})$  is *msg*-clear with respect to  $f(\Gamma)$ . Moreover, from assumption (ii)  $\text{ran}(\Gamma) \subseteq \text{udom}(F_f, \Gamma)$ , we derive that  $\text{ran}(\Gamma) \subseteq \Pi_f \downarrow \cup \{\text{msg}\}$ . Hence by Lemma 27, we know that  $\sigma'$  is well-typed with respect to  $f(\Gamma)$ . By Lemma 29, this implies  $f(T)\sigma', IK'_0 \vdash f(u)\sigma'$ . From here, we continue our derivation as follows.

$$\begin{array}{lll} (f(T)\sigma')[c/\text{nil}], IK'_0 & \vdash (f(u)\sigma')[c/\text{nil}] & \text{by Lemma 23} \\ \Rightarrow f(T)[c/\text{nil}](\sigma'[c/\text{nil}]), IK'_0 \vdash f(u)[c/\text{nil}](\sigma'[c/\text{nil}]) & & \text{by Lemma 24} \\ \Rightarrow f(T)(\sigma'[c/\text{nil}]), IK'_0 & \vdash f(u)(\sigma'[c/\text{nil}]) & \end{array}$$

The last step follows, since  $f(T)$  and  $f(u)$  are in normal form and  $\{c, \text{nil}\} \subseteq IK'_0$ . This completes the proof of the proposition.  $\square$

## G Soundness of type-based abstractions (Section 4.2)

### G.1 Reachability preservation

Using the deducibility preservation for open terms, we show that each reachable state in the original protocol can be simulated by one in the abstracted protocol.

**Theorem 4 (Reachability preservation).** *Let  $P = (\Gamma_P, S_P)$  be a protocol,  $F_f$  a type-based abstraction such that*

- (i)  $IK'_0$  is normal-form and  $f(IK_0) \subseteq IK'_0$ ,
- (ii)  $\mathcal{M}_P \cup \text{dom}(\Gamma_P) \subseteq \text{udom}(F_f, \Gamma_P)$ ,
- (iii)  $f(P)$  is msg-clear if  $F_f$  is not nil-free.

*Let  $(tr, th, \sigma)$  be a reachable state of  $P$  such that  $\sigma$  is nil-free and  $\sigma' = f(\sigma)$ . Then for all  $c \in \mathcal{C}$ , it holds that  $(f(tr), f(th), \sigma'[c/\text{nil}])$  is a reachable state of  $f(P)$ .*

*Proof.* Note that  $\sigma'$  is a well-typed ground substitution with respect to  $\Gamma_{f(P)}$  by Lemma 27. Let  $c \in \mathcal{C}$ . Then by Corollary 1, it is clear that for every type  $\tau \in \text{ran}(\Gamma_P)$  and  $\gamma_{\text{nil}} \preceq \tau$ , we have  $\gamma_c \preceq \tau$ . This yields  $\sigma'[c/\text{nil}]$  is well-typed with respect to  $\Gamma_{f(P)}$ . We show that  $(f(tr), f(th), \sigma'[c/\text{nil}])$  is reachable in  $f(P)$  by induction on the number  $n$  of transitions leading to a state  $(tr, th, \sigma)$ .

- Base case ( $n = 0$ ): For all  $i \in \text{dom}(th)$ , there exists  $R \in \text{dom}(S_P)$  such that  $th(i) = (R, S_P(R))$ . Hence we have

$$f(th)(i) = (R, f(S_P(R))) = (R, f(S_P)(R)) \quad (8)$$

Since  $(\epsilon, th, \sigma)$  is reachable, for all  $v \in \text{dom}(S_P)$  and  $i \in TID$  we have  $\text{inst}_i(v)\sigma \in \mathcal{A}$ . Moreover, we have  $\text{inst}_i(v)\sigma'[c/\text{nil}] = \text{inst}_i(v)\sigma$ , we also have

$$\text{inst}_i(v)\sigma'[c/\text{nil}] \in \mathcal{A} \quad (9)$$

By (8), (9) and  $f(\epsilon) = \epsilon$ , it is obvious that  $(f(\epsilon), f(th), \sigma'[c/\text{nil}])$  is reachable in  $f(P)$ .

- Inductive case ( $n = k + 1$ ): Suppose  $(tr', th', \sigma)$  is reachable in  $k$  steps and there is a transition  $(tr', th', \sigma) \rightarrow (tr, th, \sigma)$ . By induction hypothesis, we have

$$(f(tr'), f(th'), \sigma'[c/\text{nil}]) \text{ is reachable in } f(P) \quad (10)$$

We consider two cases according to the rule  $r$  that has been applied in step  $k + 1$ .

- If  $r = \text{SEND}$  then there exists  $i \in TID$  and  $R \in \text{dom}(S_P)$  such that

$$\begin{aligned} th'(i) &= (R, \text{snd}(pt).tl) \\ tr &= tr' \cdot (i, \text{snd}(pt)) \\ th &= th'[i \mapsto (R, tl)] \end{aligned} \quad (11)$$

\* If  $f(pt) = \text{nil}$  then by (11) we have

$$\begin{aligned} f(tr) &= f(tr') \\ f(th) &= f(th')[i \mapsto (R, f(tl))] \end{aligned} \quad (12)$$

By (12) and (11) we have  $f(th) = f(th')$ . Together with (10) we obtain that

$$(f(tr), f(th), \sigma'[c/\text{nil}]) \text{ is reachable in } f(P)$$

\* If  $f(pt) \neq \text{nil}$  then by (11) we have

$$\begin{aligned} f(tr) &= f(tr') \cdot (i, \text{snd}(f(pt))) \\ f(th) &= f(th')[i \mapsto (R, f(tl))] \end{aligned} \quad (13)$$

By (11) we have

$$f(th')(i) = (R, \text{snd}(f(pt)) \cdot f(tl)) \quad (14)$$

By (14), (11), (13) and rule *SEND*, we have

$$(f(tr'), f(th'), \sigma'[c/\text{nil}]) \rightarrow (f(tr), f(th), \sigma'[c/\text{nil}])$$

Together with (10) this implies that  $(f(tr), f(th), \sigma'[c/\text{nil}])$  is reachable in  $P$ .

- If  $r = \text{RECV}$  then there exists  $i \in \text{TID}$  and  $R \in \text{dom}(S_P)$  such that

$$\begin{aligned} th'(i) &= (R, \text{rcv}(u) \cdot tl) \\ IK(tr')\sigma, IK_0 \vdash u\sigma \end{aligned} \quad (15)$$

and

$$\begin{aligned} tr &= tr' \cdot (i, \text{rcv}(u)) \\ th &= th'[i \mapsto (R, tl)] \end{aligned} \quad (16)$$

If  $f(u) = \text{nil}$  then similar as before, we have  $(f(tr), f(th), \sigma'[c/\text{nil}])$  is reachable in  $f(P)$ . Suppose that  $f(u) \neq \text{nil}$ . By (15) and (16) we have

$$\begin{aligned} f(tr) &= f(tr') \cdot (i, \text{rcv}(f(u))) \\ f(th) &= f(th')[i \mapsto (R, f(tl))] \end{aligned}$$

To justify  $(f(tr'), f(th'), \sigma'[c/\text{nil}]) \rightarrow (f(tr), f(th), \sigma'[c/\text{nil}])$ , it is sufficient to establish the following two premises of rule *RECV*:

1.  $f(th')(i) = (R, \text{rcv}(f(u)) \cdot f(tl))$ , which follows from (15), and
2.  $IK(f(tr'))\sigma'[c/\text{nil}], IK'_0 \vdash f(u)\sigma'[c/\text{nil}]$ . This follows from (15), Proposition 4, and the fact that  $f(IK(tr')) \subseteq IK(f(tr')) \cup \{\text{nil}\}$ .

Together with (10) this implies that  $(f(tr), f(th), \sigma'[c/\text{nil}])$  is reachable in  $f(P)$ .

This completes the proof of the theorem.  $\square$

## G.2 Soundness

**Lemma 30.** *Let  $P = (\Gamma_P, S_P)$  be a protocol and  $\phi \in \mathcal{L}_P$  and let  $(tr, th, \sigma)$  be a reachable state of  $P$ . Assume that  $IK_0$  is in normal form. Then there is a nil-free ground substitution  $\rho$  such that*

- (i)  $(tr, th, \rho)$  is a reachable state in  $P$ , and
- (ii) if  $(tr, th, \sigma) \not\equiv \phi$  then  $(tr, th, \rho) \not\equiv \phi$ .

*Proof.* Let  $c \in \mathcal{C} \setminus \{\text{nil}\}$  such that  $c$  does not occur in  $\text{EqTerm}_\phi \cup \text{ran}(\sigma)$ . We define  $\rho = \sigma[c/\text{nil}]$ . First, we show that  $\rho$  is well-typed with respect to  $\Gamma_P$ . Let  $X \in \text{dom}(\sigma)$ ,  $\Gamma_P(X) = \tau$ ,  $\Gamma_P(X\sigma) = \tau'$ , and  $\Gamma_P(X\rho) = \tau''$ . We need to show that  $\tau'' \preceq \tau$ . It is enough to consider the non-trivial case that  $\text{nil} \in \text{subs}(X\sigma)$ . By Corollary 1, we have that  $\gamma_{\text{nil}} \preceq \tau \Rightarrow \tau = \text{msg}$  for all  $\tau \in \text{ran}(\Gamma_P)$ . Hence it follows that

$$\forall \tau \in \text{ran}(\Gamma_P). \gamma_{\text{nil}} \preceq \tau \Rightarrow \gamma_c \preceq \tau \quad (17)$$

Since  $\sigma$  is well-typed with respect to  $\Gamma_P$ , we have that

$$\tau' \preceq \tau \quad (18)$$

By (18) and (17) we derive that  $\tau'[\gamma_c/\gamma_{\text{nil}}] \preceq \tau$ . Together with  $\tau'' = \tau'[\gamma_c/\gamma_{\text{nil}}]$ , we obtain that  $\tau'' \preceq \tau$ . Thus  $\rho$  is well-typed with respect to  $\Gamma_P$ . Moreover,  $\rho$  is nil-free. Therefore, it remains to show reachability and attack preservation for  $\rho$ .

For reachability, we need to show that  $(tr, th, \rho)$  is reachable in  $P$ . We prove this by induction on the number  $n$  of transitions leading to  $(tr, th, \sigma)$ .

- Base case ( $n = 0$ ): Since  $(\epsilon, th, \sigma)$  is reachable, so is  $(\epsilon, th, \rho)$ .
- Inductive case ( $n = k + 1$ ): Suppose  $(tr', th', \sigma)$  is reachable in  $k$  steps and there is a transition  $(tr', th', \sigma) \rightarrow (tr, th, \sigma)$ . By induction hypothesis, we have

$$(tr', th', \rho) \text{ is reachable in } P. \quad (19)$$

We consider two cases according to the rule  $r$  that has been applied in step  $k + 1$ .

- If  $r = \text{SEND}$  then it is obvious that  $(tr', th', \rho) \rightarrow (tr, th, \rho)$ . This by (19) yields that  $(tr, th, \rho)$  is reachable in  $P$ .
- If  $r = \text{RECV}$  then there exists  $i \in \text{TID}$  and  $R \in \text{dom}(S_P)$  such that  $th'(i) = (R, \text{rcv}(u) \cdot tl)$ ,  $tr = tr' \cdot (i, \text{rcv}(u))$ ,  $th = th'[i \mapsto (R, tl)]$ , and

$$IK(tr')\sigma, IK_0 \vdash u\sigma$$

The reachability of  $(tr, th, \rho)$  in  $P$  follows from  $IK(tr')\rho, IK_0 \vdash u\rho$ , which we now show. By Lemma 23 and since  $IK_0 \vdash IK_0[c/\text{nil}]$ , we have

$$(IK(tr')\sigma)[c/\text{nil}], IK_0 \vdash (u\sigma)[c/\text{nil}].$$

Since  $\text{nil} \notin IK(tr')$  and  $u \neq \text{nil}$ , the application of Lemma 24 yields the desired result  $IK(tr')\rho, IK_0 \vdash u\rho$ .

For attack preservation, we need to show that

$$\forall \vartheta. (tr, th, \sigma, \vartheta) \not\models \phi \Rightarrow (tr, th, \rho, \vartheta) \not\models \phi$$

We prove this by induction on the structure of  $\phi$ . Note that the base cases for the literals that do not depend on  $\sigma$  are trivial. Hence, it is enough to consider the following cases.

$$- \phi \equiv m = m' \text{ or } \phi \equiv \neg(m = m').$$

$$\begin{aligned} & (tr, th, \sigma, \vartheta) \models m = m' \\ \Leftrightarrow & m\sigma = m'\sigma \\ \Leftrightarrow & m\rho = m'\rho && \text{since } c \text{ is not in } EqTerm_\phi \cup ran(\sigma) \\ \Leftrightarrow & (tr, th, \rho, \vartheta) \models m = m' \end{aligned}$$

$$- \phi \equiv honest(i, R) \text{ or } \phi \equiv \neg honest(i, R).$$

$$\begin{aligned} & (tr, th, \sigma, \vartheta) \models honest(i, R) \\ \Leftrightarrow & R^{\vartheta(i)}\sigma \in \mathcal{A}_H \\ \Leftrightarrow & R^{\vartheta(i)}\rho \in \mathcal{A}_H && \text{since } R^{\vartheta(i)}\sigma = R^{\vartheta(i)}\rho \\ \Leftrightarrow & (tr, th, \rho, \vartheta) \models honest(i, R) \end{aligned}$$

$$- \phi \equiv secret(m).$$

$$\begin{aligned} & (tr, th, \sigma, \vartheta) \not\models secret(m) \\ \Leftrightarrow & IK(tr)\sigma, IK_0 \vdash m\sigma \\ \Rightarrow & IK(tr)\rho, IK_0 \vdash m\rho && \text{by Lemma 23 and Lemma 24} \\ \Leftrightarrow & (tr, th, \rho, \vartheta) \not\models secret(m) \end{aligned}$$

As above, the implication requires the facts that  $IK_0 \vdash IK_0[c/\text{nil}]$  (for Lemma 23) and  $\text{nil} \notin IK(tr)$  and  $m \neq \text{nil}$  (for Lemma 24).

Here are the inductive cases:

$$- \phi = \phi_1 \wedge \phi_2.$$

$$\begin{aligned} & (tr, th, \sigma, \vartheta) \not\models \phi \\ \Leftrightarrow & (tr, th, \sigma, \vartheta) \not\models \phi_i && \text{for some } i \in \{1, 2\} \\ \Rightarrow & (tr, th, \rho, \vartheta) \not\models \phi_i && \text{by induction hypothesis} \\ \Leftrightarrow & (tr, th, \rho, \vartheta) \not\models \phi \end{aligned}$$

$$- \phi = \phi_1 \vee \phi_2. \text{ Similar to case for conjunction.}$$

$$- \phi = \forall i. \phi'.$$

$$\begin{aligned} & (tr, th, \sigma, \vartheta) \not\models \forall i. \phi' \\ \Leftrightarrow & (tr, th, \sigma, \vartheta[i \mapsto tid]) \not\models \phi' && \text{for some } tid \in dom(th) \\ \Rightarrow & (tr, th, \rho, \vartheta[i \mapsto tid]) \not\models \phi' && \text{by induction hypothesis} \\ \Leftrightarrow & (tr, th, \rho, \vartheta) \not\models \forall i. \phi' \end{aligned}$$

$$- \phi = \exists i. \phi'. \text{ Similar to case for universal quantifier.}$$

This concludes the proof of the lemma.  $\square$

### G.3 Soundness of type-based abstractions

**Theorem (Soundness; Justification of Theorem 3).** *Let  $P = (\Gamma_P, S_P)$  be a protocol,  $\phi \in \mathcal{L}_P$  a property, and  $F_f = (f, \Gamma_f, E_f)$  be a type-based abstraction such that*

- (i)  $IK'_0$  is in normal form and  $f(IK_0) \subseteq IK'_0$ .
- (ii)  $\mathcal{M}_P \cup \text{dom}(\Gamma_P) \subseteq \text{udom}(F_f, \Gamma_P)$ ,
- (iii)  $f(P)$  is msg-clear if  $F_f$  is not nil-free, and
- (iv)  $\phi$  is  $(P, f)$ -safe.

*Then, for all states  $(tr, th, \sigma)$  reachable in  $P$ , there is a nil-free ground substitution  $\sigma'$  such that*

1.  $(f(tr), f(th), \sigma')$  is a reachable state of  $f(P)$ ,
2.  $(tr, th, \sigma) \not\models \phi$  implies  $(f(tr), f(th), \sigma') \not\models f(\phi)$ .

*Proof.* Let  $(tr, th, \sigma)$  be a reachable state of  $P$ . By Lemma 30, there is a nil-free ground substitution  $\sigma''$  such that (a)  $(tr, th, \sigma'')$  is a reachable state of  $P$ , and (b)  $(tr, th, \sigma) \not\models \phi$  implies  $(tr, th, \sigma'') \not\models \phi$ .

Let  $\sigma' = f(\sigma'')[c/\text{nil}]$  for some  $c \in \mathcal{C}$  that does not occur in  $\text{Eq}_\phi \cup \text{ran}(f(\sigma''))$ . Then it is clear that  $\sigma'$  is nil-free. Moreover, point (i) follows from (a) and Theorem 4. Using (b) we reduce point (ii) to showing that  $(tr, th, \sigma'') \not\models \phi$  implies  $(f(tr), f(th), \sigma') \not\models f(\phi)$ , which we establish by proving the following generalized statement by induction on the structure of  $\phi$  (which may now contain free thread-id variables).

$$\forall \vartheta. (tr, th, \sigma'', \vartheta) \not\models \phi \Rightarrow (f(tr), f(th), \sigma', \vartheta) \not\models f(\phi)$$

Note that a formula is  $(P, f)$ -safe if and only if all its subformulas are  $(P, f)$ -safe. The literals form the base cases of the induction. We cover all atoms and their negations (except  $\text{secret}(m)$ ) in a single equivalence-based argument, where the right-to-left direction covers the positive literal and the other direction the corresponding negative literal. We remark that  $(tr, th, \sigma, \vartheta) \not\models A$  is equivalent to  $(tr, th, \sigma, \vartheta) \models \neg A$  for all atoms  $A$  (but not for all formulas, since  $\mathcal{L}_P$  is not closed under negation).

$$- \phi \equiv i = j \text{ or } \phi \equiv \neg(i = j).$$

$$\begin{aligned} & (tr, th, \sigma'', \vartheta) \models i = j \\ \Leftrightarrow & \vartheta(i) = \vartheta(j) \\ \Leftrightarrow & (f(tr), f(th), \sigma', \vartheta) \models f(i = j) \end{aligned}$$

$$- \phi \equiv m = m'.$$

$$\begin{aligned} & (tr, th, \sigma'', \vartheta) \models m = m' \\ \Rightarrow & m\sigma'' = m'\sigma'' \\ \Rightarrow & nf(f(m)f(\sigma'')) = nf(f(m')f(\sigma'')) \\ \Rightarrow & f(m)f(\sigma'') = f(m')f(\sigma'') \\ \Leftrightarrow & f(m)\sigma' = f(m')\sigma' \\ \Leftrightarrow & (f(tr), f(th), \sigma', \vartheta) \models f(m) = f(m') \end{aligned}$$

by Theorem 2  
since  $\phi$  is  $(P, f)$ -safe  
since  $c$  is not in  $\text{Eq}_\phi \cup \text{ran}(f(\sigma''))$

–  $\phi \equiv \neg(m = m')$ .

$$\begin{aligned}
& (f(tr), f(th), \sigma', \vartheta) \models f(m) = f(m') \\
& \Rightarrow f(m)\sigma' = f(m')\sigma' \\
& \Leftrightarrow f(m)f(\sigma'') = f(m')f(\sigma'') && \text{since } c \text{ is not in } Eq_\phi \\
& \Rightarrow nf(f(m)f(\sigma'')) = nf(f(m')f(\sigma'')) \\
& \Rightarrow m\sigma'' = m'\sigma'' && \text{since } \phi \text{ is } (P, f)\text{-safe} \\
& \Rightarrow (tr, th, \sigma'', \vartheta) \models m = m'
\end{aligned}$$

–  $\phi \equiv \text{role}(i, R)$  or  $\phi \equiv \neg\text{role}(i, R)$ .

$$\begin{aligned}
& (tr, th, \sigma'', \vartheta) \models \text{role}(i, R) \\
& \Leftrightarrow \exists \text{seq} \in \text{Evt}^*. th(\vartheta(i)) = (R, \text{seq}) \\
& \Leftrightarrow \exists \text{seq} \in \text{Evt}^*. f(th)(\vartheta(i)) = (R, f(\text{seq})) \\
& \Leftrightarrow (f(tr), f(th), \sigma', \vartheta) \models \text{role}(i, R)
\end{aligned}$$

–  $\phi \equiv \text{honest}(i, R)$  or  $\phi \equiv \neg\text{honest}(i, R)$ .

$$\begin{aligned}
& (tr, th, \sigma'', \vartheta) \models \text{honest}(i, R) \\
& \Leftrightarrow R^{\vartheta(i)}\sigma'' \in \mathcal{A}_H \\
& \Leftrightarrow R^{\vartheta(i)}f(\sigma'') \in \mathcal{A}_H && f \text{ is the identity on } \mathcal{A} \\
& \Leftrightarrow R^{\vartheta(i)}\sigma' \in \mathcal{A}_H && \text{since } R^{\vartheta(i)}\sigma' = R^{\vartheta(i)}f(\sigma'') \\
& \Leftrightarrow (f(tr), f(th), \sigma', \vartheta) \models \text{honest}(i, R)
\end{aligned}$$

–  $\phi \equiv \text{steps}(i, s(m))$  or  $\phi \equiv \neg\text{steps}(i, s(m))$ , where  $s \in \{\text{snd}, \text{rcv}\}$ . We have

$$\begin{aligned}
& (tr, th, \sigma'', \vartheta) \models \text{steps}(i, s(m)) \\
& \Leftrightarrow (\vartheta(i), s(\text{inst}_{\vartheta(i)}(m))) \in tr \\
& \Leftrightarrow (\vartheta(i), s(\text{inst}_{\vartheta(i)}(f(m)))) \in f(tr) && \text{justified below} \\
& \Leftrightarrow (f(tr), f(th), \sigma', \vartheta) \models \text{steps}(i, s(f(m)))
\end{aligned}$$

We show the second equivalence. The left-to-right implication holds, since  $\phi$  is  $(P, f)$ -safe and we therefore have  $f(m) \neq \text{nil}$ . For the inverse direction (covering the positive literal  $\phi \equiv \text{steps}(i, s(m))$ ), suppose that

$$(\vartheta(i), s(\text{inst}_{\vartheta(i)}(f(m)))) \in f(tr).$$

Then there exists  $s(m') \in \text{Evt}(\mathcal{M}_P)$  such that  $(\vartheta(i), s(\text{inst}_{\vartheta(i)}(m'))) \in tr$  and  $f(m') = f(m)$ . Since  $\phi$  is  $(P, f)$ -safe, this implies  $m = m'$  and hence  $(\vartheta(i), s(\text{inst}_{\vartheta(i)}(m))) \in tr$ .

–  $\phi \equiv (i, s(m)) \prec (j, s'(m'))$  or  $\phi \equiv \neg((i, s(m)) \prec (j, s'(m')))$ , where  $s, s' \in \{\text{snd}, \text{rcv}\}$ .

$$\begin{aligned}
& (tr, th, \sigma'', \vartheta) \models (i, s(m)) \prec (j, s'(m')) \\
& \Leftrightarrow (\vartheta(i), s(\text{inst}_{\vartheta(i)}(m))) \prec_{tr} (\vartheta(j), s'(\text{inst}_{\vartheta(j)}(m'))) \\
& \Leftrightarrow (\vartheta(i), s(\text{inst}_{\vartheta(i)}(f(m)))) \prec_{f(tr)} (\vartheta(j), s'(\text{inst}_{\vartheta(j)}(f(m')))) \text{ justified below} \\
& \Leftrightarrow (f(tr), f(th), \sigma', \vartheta) \models (i, s(f(m))) \prec (j, s'(f(m')))
\end{aligned}$$

We show the second equivalence. Note that, since  $\phi$  is  $(P, f)$ -safe and we have that  $f(m) \neq \text{nil}$  and  $f(m') \neq \text{nil}$ . The if-direction immediately follows, since  $f$  is order-preserving for events that do not map to nil.

For the only-if direction (covering the case that  $\phi \equiv (i, s(m)) \prec (j, s'(m'))$ ), suppose  $(\vartheta(i), s(\text{inst}_{\vartheta(i)}(f(m)))) \prec_{f(\text{tr})} (\vartheta(j), s'(\text{inst}_{\vartheta(j)}(f(m'))))$ . Since  $f$  is order-preserving for events that do not map to nil, there are  $s(u), s'(u') \in \text{Evt}(\mathcal{M}_P)$  such that  $(\vartheta(i), s(\text{inst}_{\vartheta(i)}(u))) \prec_{\text{tr}} (\vartheta(j), s'(\text{inst}_{\vartheta(j)}(u')))$  with

$$f(u) = f(m) \text{ and } f(u') = f(m').$$

Since  $\phi$  is  $(P, f)$ -safe, we have  $u = m$  and  $u' = m'$ , completing the proof of this direction.

–  $\phi \equiv \text{secret}(m)$ .

$$\begin{aligned} & (tr, th, \sigma'', \vartheta) \not\models \text{secret}(m) \\ \Leftrightarrow & IK(tr)\sigma'', IK_0 \vdash m\sigma'' \\ \Rightarrow & f(IK(tr))\sigma', IK'_0 \vdash f(m)\sigma' && \text{by Proposition 4} \\ \Rightarrow & IK(f(tr))\sigma', IK'_0 \vdash f(m)\sigma' && f(IK(tr)) \subseteq IK(f(tr)) \cup \{\text{nil}\} \\ & && \text{and } \text{nil} \in IK'_0 \\ \Leftrightarrow & (f(tr), f(th), \sigma', \vartheta) \not\models \text{secret}(f(m)) \end{aligned}$$

Here are the inductive cases:

–  $\phi = \phi_1 \wedge \phi_2$ .

$$\begin{aligned} & (tr, th, \sigma'', \vartheta) \not\models \phi \\ \Leftrightarrow & (tr, th, \sigma'', \vartheta) \not\models \phi_i && \text{for some } i \in \{1, 2\} \\ \Rightarrow & (f(tr), f(th), \sigma', \vartheta) \not\models f(\phi_i) && \text{by induction hypothesis} \\ \Leftrightarrow & (f(tr), f(th), \sigma', \vartheta) \not\models f(\phi) \end{aligned}$$

–  $\phi = \phi_1 \vee \phi_2$ . Similar to case for conjunction.

–  $\phi = \forall i. \phi'$ .

$$\begin{aligned} & (tr, th, \sigma'', \vartheta) \not\models \forall i. \phi' \\ \Leftrightarrow & (tr, th, \sigma'', \vartheta[i \mapsto \text{tid}]) \not\models \phi' && \text{for some } \text{tid} \in \text{dom}(th) \\ \Rightarrow & (f(tr), f(th), \sigma', \vartheta[i \mapsto \text{tid}]) \not\models f(\phi') && \text{by induction hypothesis} \\ \Leftrightarrow & (f(tr), f(th), \sigma', \vartheta) \not\models f(\forall i. \phi') \end{aligned}$$

–  $\phi = \exists i. \phi'$ . Similar to case for universal quantifier.

This concludes the proof of the theorem.  $\square$

## H Syntactic criteria for $(P, f)$ -safety

Conditions (iii) and (iv) in the definition of  $(P, f)$ -safe formulas (Definition 9) are hard to check in practice, since they universally quantify over well-typed nil-free ground substitutions. We therefore propose syntactic criteria for verifying these conditions.

In the following, we establish a criterion that justifies conditions (iii) and (iv) mentioned above for equalities of terms in which no variables of type  $msg$  are present.

**Proposition 5.** *Let  $P = (\Gamma_P, S_P)$  be a protocol,  $F_f = (f, \Gamma_f, E_f)$  a type-based abstraction. Let  $t, u \in \mathcal{M}^\sharp$  such that*

- $\Gamma_P(t)$  and  $\Gamma_P(u)$  are defined,
- $msg \notin \Gamma_P(\text{vars}(t) \cup \text{vars}(u))$ , and
- $t, u \in \text{udom}(F_f, \Gamma_P)$ .

*Let  $\sigma$  be a nil-free ground substitutions that is well-typed with respect to  $\Gamma_P$ . Suppose that  $f(t\sigma) = f(u\sigma)$ . Then the following two points hold.*

1.  $f(t)f(\sigma) = f(u)f(\sigma)$ .
2. *Suppose that the existence of  $\gamma = \text{mgu}(f(t), f(u))$  implies  $t\gamma = u\gamma$ . Then  $t\sigma = u\sigma$ .*

*Proof.* Let  $t, u \in \mathcal{M}^\sharp$  be two messages such that  $\Gamma_P(t)$  and  $\Gamma_P(u)$  are defined and let  $\sigma$  be a nil-free ground substitution that is well-typed with respect to  $\Gamma_P$ . Suppose that  $f(t\sigma) = f(u\sigma)$ .

By Theorem 2, we derive  $f(t\sigma) = nf(f(t)f(\sigma))$ . By Lemma 27, we know that  $f(\sigma)$  is well-typed with respect to  $\Gamma_{f(P)}$ . Hence, by Lemma 28, we derive that for all  $X \in \text{dom}(f(\sigma))$ ,  $Xf(\sigma) = \text{nil}$  implies  $\Gamma_P(X) = msg$ . Since  $msg \notin \Gamma_P(\text{vars}(t) \cup \text{vars}(u))$ , we derive that  $f(\sigma)|_{\text{vars}(t) \cup \text{vars}(u)}$  is nil-free. Therefore, it holds that

$$\begin{aligned} f(t\sigma) &= f(t)f(\sigma) \\ f(u\sigma) &= f(u)f(\sigma) \end{aligned} \tag{20}$$

Next, we establish the points 1 and 2 of the proposition in turn.

1. Since  $f(t\sigma) = f(u\sigma)$ , point 1 holds by (20).
2. We have to show  $t\sigma = u\sigma$ . By point 1, we have  $f(t)f(\sigma) = f(u)f(\sigma)$ . Hence  $\gamma = \text{mgu}(f(t), f(u))$  exists. Hence there must be a substitution  $\eta$  such that  $f(\sigma) = \eta \circ \gamma$ . Since  $\Gamma_P$  is atomic, there is a substitution  $\xi$  such that  $\text{dom}(\xi) \cap \text{dom}(f(\sigma)) = \emptyset$ ,  $\text{dom}(\xi) \cup \text{dom}(f(\sigma)) = \text{dom}(\sigma)$ , and  $\sigma = \xi \circ f(\sigma)$ . Thus, we obtain  $\sigma = \xi \circ \eta \circ \gamma$ . From the existence of  $\gamma$ , we have  $t\gamma = u\gamma$ . Hence, we derive that  $t(\xi \circ \eta \circ \gamma) = u(\xi \circ \eta \circ \gamma)$ . This yields  $t\sigma = u\sigma$  as required.

This completes the proof of the proposition. □

The main restriction of the above result is that message variables are not allowed in properties. To widen the scope, we identify a syntactic criterion for

nil-free type-based abstractions under which condition (iii) is satisfied. For these abstractions, condition (iv) trivially holds as no normalization is needed. First, we introduce some definitions. For each term  $t$ , we use  $ct(t)$  to denote the set of all constructors of non-zero arity in  $t$ . We also denote the top constructor of  $t$  by  $top(t)$ . By  $\Gamma_\emptyset$ , we denote the empty typing environment.

**Definition 11.** *Let  $F_f = (f, \Gamma_f, E_f)$  be a type-based abstraction and  $g \in \Sigma^1 \cup \Sigma^2$  be a constructor. We say that*

- $F_f$  is composite-preserving if for all  $(f(p) = u) \in E_f$ , we have  $p$  is composed implies either
  1.  $u$  is composed, or
  2.  $u = f_1(q)$  and  $\Gamma_f(q) \in \text{udom}(F_f, \Gamma_\emptyset)$  and  $f_1(\Gamma_f(q))$  is composed.
- $F_f$  is constructor-exclusive with respect to  $g$  if for all  $(f(p) = u) \in E_f$  such that  $top(p) \in \Sigma^1 \cup \Sigma^2 \setminus \{g\}$ , we have  $\Gamma_f(p) \in \text{udom}(F_f, \Gamma_\emptyset)$  and  $top(f(\Gamma_f(p))) \in \Sigma \setminus \{g, \text{msg}\}$ .
- $F_f$  is homomorphic with respect to  $g$  if there is  $f(p) = u$  in  $E_f$  with  $p = g(p_1, \dots, p_n)$ ,  $\Gamma_f(p_i) = \text{msg}$  for all  $i \in \tilde{n}$ , and  $u = g(f(p_1), \dots, f(p_n))$ .

Intuitively,  $F_f$  is composite-preserving if it never maps a composed term to an atom, and  $F_f$  is constructor-exclusive with respect to  $g$  if every term with top-level constructor  $g$  can only be obtained from one of the same form.

We extend the above notions to a set of symbols  $S \subseteq \Sigma$  as expected, i.e.  $F_f$  is constructor-exclusive (homomorphic) with respect to  $S$  if it is with respect to each  $g \in S$ .

We show in the following lemma that, under certain conditions, an equality of abstracted terms implies an equality of the original terms.

**Lemma 31.** *Let  $t, u \in \mathcal{N}$  be ground terms and  $F_f = (f, \Gamma_f, E_f)$  be a type-based abstraction that is nil-free, composite-preserving, and constructor-exclusive and homomorphic with respect to  $ct(t)$ . Then, we have that  $f(u) = f(t)$  implies  $u = t$ .*

*Proof.* Suppose that  $f(u) = f(t)$ . We show that  $u = t$  by induction on  $t$ .

- If  $t$  is an atom then since  $F_f$  is nil-free, we have  $f(t) = t$ . Since  $F_f$  is composite-preserving and nil-free, we derive that  $u$  is atomic. We also have  $f(u) = u$  and thus  $u = t$  as required.
- If  $t = g(t_1, \dots, t_n)$  for some  $g \in \Sigma^1 \cup \Sigma^2$ , then since  $F_f$  is nil-free and homomorphic with respect to  $ct(t)$ , we derive that  $f(t) = t$ . Therefore, we have  $f(u) = t$ . Hence  $u$  is composed, i.e.,  $top(u) \in \Sigma^1 \cup \Sigma^2$ . Moreover, there must be  $(f(p) = q) \in E_f^+$  as the first pattern such that  $\Gamma_\emptyset(u) \preceq \Gamma_f(p)$ . Thus  $p$  is composed and  $top(p) = top(u)$ . Moreover, since  $F_f$  is constructor-exclusive and homomorphic with respect to  $ct(t)$ , so is it with respect to  $g$ . Now, we show that  $top(p) = g$ . We distinguish two cases.
  - If  $(f(p) = q) \in E_f^0$  then since this clause is homomorphic and  $F_f$  is nil-free, we have  $top(p) = top(u) = top(f(u)) = g$ .
  - If  $(f(p) = q) \in E_f$  then we have one of the following two cases:

1.  $top(f(\Gamma_f(p))) \in \Sigma^1 \cup \Sigma^2$  and  $top(f(u)) = top(f(\Gamma_f(p)))$ . In this case, we have  $top(f(\Gamma_f(p))) = top(f(u)) = g$ . Since  $F_f$  is constructor-exclusive with respect to  $g$ , we derive that  $top(p) = g$ .
2.  $f(\Gamma_f(p)) \in \Sigma^0$ . In this case, since  $f(u)$  is composed, it must hold that  $f(\Gamma_f(p)) = msg$ . Since  $F_f$  is constructor-exclusive with respect to  $g$ , we must have  $top(p) = g$ .

Therefore, we have  $top(u) = top(p) = g$  and hence  $u = g(u_1, \dots, u_n)$ . Since  $F_f$  is nil-free and homomorphic with respect to  $g$ , we know that

$$f(u) = g(f(u_1), \dots, f(u_n)).$$

Together with  $f(u) = t$ , we have  $f(u_i) = t_i$  for all  $i \in \tilde{n}$ . Moreover, by induction hypothesis, we know that  $u_i = t_i$  for all  $i \in \tilde{n}$ . Therefore, we obtain  $u = t$  as required.

This completes the proof of the lemma. □

We now establish a syntactic criterion to justify condition (iii) for equalities in which message variables can only occur in either side.

**Proposition 6.** *Let  $\Gamma$  be a typing environment,  $t, u \in \mathcal{M}^\sharp$  be terms such that  $\Gamma(t)$  is defined and  $msg \notin \Gamma(vars(t))$ ,  $F_f = (f, \Gamma_f, E_f)$  be a type-based abstraction such that  $F_f$  is nil-free, composite-preserving, constructor-exclusive and homomorphic with respect to  $ct(t)$ . Then for all ground substitution  $\sigma$  that is well-typed with respect to  $\Gamma$  such that  $vars(t) \cup vars(u) \subseteq dom(\sigma)$  and  $f(u\sigma) = f(t\sigma)$ , we have  $u\sigma = t\sigma$ .*

*Proof.* Since  $msg \notin vars(t)$ , all variables in  $t$  are of atomic types. Moreover, since  $\sigma$  is well-typed with respect to  $\Gamma$ , we have  $\Gamma(t\sigma) \preceq \Gamma(t)$ . It follows that  $ct(t\sigma) = ct(t)$ . Since  $F_f$  is constructor-exclusive and homomorphic with respect to  $ct(\Gamma(t\sigma))$ , so is it with respect to  $ct(t\sigma)$ . We also have that  $t\sigma$  and  $u\sigma$  are ground. Thus, by Lemma 31, we obtain that  $u\sigma = t\sigma$  as required. □

# I Redundancy removal abstraction

In this section, we discuss protocol abstractions which allow us to remove redundancies in protocol specifications. For instance, we can remove intruder-derivable terms or repeated occurrences of a term. We call these abstractions *redundancy removal abstractions*.

Let  $\Gamma$  be a typing environment,  $rd : \mathcal{M} \rightarrow \mathcal{M}$  be a total function on messages,  $T$  be a set of terms, and  $S \in \text{Evt}(\mathcal{M})^*$  be a sequence of events. We use the predicate  $RD_\Gamma^{rd}(T, S)$  to realize the necessary conditions for redundancy removal abstractions. This predicate is inductively defined by the following two rules.

$$\frac{}{RD_\Gamma^{rd}(T, \epsilon)} \quad \frac{RD_\Gamma^{rd}(T \cup \{t\}, r) \quad T, \Gamma^{-1}(\alpha), rd(t) \vdash t \quad T, \Gamma^{-1}(\alpha), t \vdash rd(t)}{RD_\Gamma^{rd}(T, ev(t) \cdot r)}$$

Intuitively, a term  $t$  can be transformed into  $u$  if the intruder, given his knowledge, is able to construct  $t$  from  $u$  and vice versa. Now, we define our class of redundancy removal abstractions for protocols.

**Definition 12 (Redundancy removal abstractions for protocols).** *Let  $P = (\Gamma_P, S_P)$  be a protocol. The set of redundancy removal abstractions  $RD_P$  for  $P$  is defined by*

$$RD_P = \{rd : \mathcal{M}_P \rightarrow \mathcal{M} \mid \forall R \in \text{dom}(S_P). RD_{\Gamma_P}^{rd}(IK_0, S_P(R))\}.$$

For all  $rd \in RD_P$ ,  $t \in \mathcal{M}_P$ , and  $i \in \text{TID}$ , we define  $rd(\text{inst}_i(t)) = \text{inst}_i(rd(t))$ . We lift  $rd$  over traces, event sequences, and thread pools of  $P$  as expected. We also define  $rd(P) = (\Gamma_P, rd(S_P))$ .

We overload the notation and use  $\text{term}(tr)$  to denote the set of terms occurring in trace  $tr$ . In the following theorem, we show reachability preservation for redundancy removal abstractions.

**Lemma 32.** *Let  $P = (\Gamma_P, S_P)$  be a protocol and  $rd \in RD_P$ . Then, for all states  $(tr, th, \sigma)$  reachable in  $P$ , we have  $IK(rd(tr))\sigma, IK_0 \vdash \text{term}(tr)\sigma$ .*

*Proof.* We proceed by induction on the number  $n$  of transitions leading to a state  $(tr, th, \sigma)$ . The theorem trivially holds for base case ( $n = 0$ ) where  $tr$  is the empty trace.

For the inductive case ( $n = k + 1$ ), we assume that  $(tr', th', \sigma)$  is reachable in  $k$  steps and there is a transition  $(tr', th', \sigma) \rightarrow (tr, th, \sigma)$ . Suppose that this transition is performed by thread  $i$ . From the transition rules, we know that  $tr = tr' \cdot (i, ev(t))$  for some  $ev \in \{\text{snd}, \text{rcv}\}$ . By the induction hypothesis, we have

$$IK(rd(tr'))\sigma, IK_0 \vdash \text{term}(tr')\sigma. \quad (21)$$

Since it follows from the induction hypothesis that  $IK(rd(tr))\sigma, IK_0 \vdash \text{term}(tr')\sigma$  and we have  $\text{term}(tr) = \text{term}(tr') \cup \{t\}$ , it is sufficient to show

$$IK(rd(tr))\sigma, IK_0 \vdash t\sigma. \quad (22)$$

We do this by case analysis on the rule that justifies transition  $k + 1$ .

- Rule *SEND*. We have  $rd(tr) = rd(tr') \cdot (i, \text{snd}(rd(t)))$  and thus  $IK(rd(tr)) = IK(rd(tr')) \cup \{rd(t)\}$  if  $rd(t) \neq \text{nil}$  and  $rd(tr) = rd(tr')$  otherwise. Hence, we can derive

$$\begin{array}{ll} IK(rd(tr))\sigma, IK_0 \vdash IK(rd(tr'))\sigma, rd(t)\sigma, IK_0 & \text{by above} \\ \vdash term(tr')\sigma, rd(t)\sigma, IK_0 & \text{by induction hyp. (21)} \end{array}$$

Next, since the terms of all events preceding  $\text{snd}(t)$  on  $S_P(R)$  are contained in  $term(tr')$  and  $rd \in RDP$ , we derive  $IK_0, term(tr'), \Gamma_P^{-1}(\alpha), rd(t) \vdash t$ . Instantiating this with  $\sigma$  and observing that  $(\Gamma_P^{-1}(\alpha))\sigma \subseteq \mathcal{A} \subseteq IK_0$  yields

$$term(tr')\sigma, rd(t)\sigma, IK_0 \vdash t\sigma.$$

Combining this with the derivation above yields the desired conclusion (22).

- Rule *RECV*. In this case we can reason as follows.

$$\begin{array}{ll} IK(rd(tr))\sigma, IK_0 \vdash term(tr')\sigma, IK_0 & \text{by induction hypothesis (21)} \\ \vdash IK(tr')\sigma, IK_0 & \text{since } IK(tr) \subseteq term(tr') \\ \vdash t\sigma & \text{by second premise of rule RECV} \end{array}$$

This establishes (22) as required.

This concludes the proof of the lemma.  $\square$

**Proposition 7.** *Let  $P = (\Gamma_P, S_P)$  be a protocol and  $rd \in RDP$ . Suppose that  $IK_0 \subseteq IK'_0$ . Then, for all states  $(tr, th, \sigma)$  reachable in  $P$ ,  $(rd(tr), rd(th), \sigma)$  is a reachable state of  $rd(P)$ .*

*Proof.* We proceed by induction on the number  $n$  of transitions leading to a state  $(tr, th, \sigma)$ . The theorem trivially holds for base case ( $n = 0$ ) where  $tr$  is the empty trace. For the inductive case ( $n = k + 1$ ), we assume that  $(tr', th', \sigma)$  is reachable in  $k$  steps and there is a transition  $(tr', th', \sigma) \rightarrow (tr, th, \sigma)$ . Suppose that this transition is performed by thread  $i$ . From the transition rules, we know that we have  $th'(i) = (R, ev(t) \cdot tl)$  where  $R \in dom(S_P)$ ,  $ev \in \{\text{snd}, \text{rcv}\}$ , and  $tl$  is a suffix of the role  $inst_i(S_P(R))$ . We also have  $tr = tr' \cdot (i, ev(t))$  and  $th = th'[i \mapsto tl]$ .

By the induction hypothesis, we have  $(rd(tr'), rd(th'), \sigma)$  is a reachable state of  $rd(P)$ . If  $rd(t) = \text{nil}$  then we are done, since  $rd(tr') = rd(tr)$  and  $rd(th') = rd(th)$ . Otherwise, we have  $rd(t) \neq \text{nil}$ . In this case, it is sufficient to show that  $rd(P)$  has a transition

$$(rd(tr'), rd(th'), \sigma) \rightarrow (rd(tr), rd(th), \sigma).$$

We proceed by case distinction on the rule applied to justify step  $k + 1$  of  $P$ .

- *SEND* rule. The rule's premise requires  $th'(i) = (R, \text{snd}(t) \cdot tl)$ . Hence, by the definition of  $rd(th')$ , we have  $rd(th')(i) = (R, ev(rd(t)) \cdot rd(tl))$ . Moreover, we have  $rd(tr) = rd(tr') \cdot (i, \text{snd}(rd(t)))$  and  $rd(th) = rd(th')[i \mapsto rd(tl)]$ , which by the *SEND* rule justifies the transition above.

– *RECV* rule. This rule’s premises require that  $th'(i) = (R, \text{rcv}(t) \cdot tl)$  and

$$IK(tr')\sigma, IK_0 \vdash t\sigma.$$

The rule’s conclusion implies that  $tr = tr' \cdot (i, \text{rcv}(t))$  and  $th = th'[i \mapsto tl]$ . In order to apply the *RECV* rule in the state  $(rd(tr'), rd(th'), \sigma)$  two premises must be satisfied: first,  $rd(th')(i) = (R, \text{rcv}(rd(e)) \cdot rd(tl))$ , which holds by the definition of  $rd(th')$ , and, second,

$$IK(rd(tr'))\sigma, IK'_0 \vdash rd(t)\sigma, \quad (23)$$

which we show now. Since  $rd \in RD_P$  and  $term(tr')$  contains the terms of all events preceding  $\text{rcv}(t)$  on  $S_P(R)$ , we have  $IK_0, term(tr'), \Gamma_P^{-1}(\alpha), t \vdash rd(t)$ . Noting that  $term(tr) = term(tr') \cup \{t\}$  and  $(\Gamma_P^{-1}(\alpha))\sigma \subseteq \mathcal{A} \subseteq IK_0$  we derive

$$term(tr)\sigma, IK_0 \vdash rd(t)\sigma.$$

Moreover, from Lemma 32, we have  $IK(rd(tr))\sigma, IK_0 \vdash term(tr)\sigma$ . Combining these facts with the observation that  $IK(rd(tr)) = IK(rd(tr'))$  and the assumption that  $IK_0 \subseteq IK'_0$ , we obtain (23) as required.

This completes the proof of the theorem.  $\square$

Next, we extend  $rd$  to formulas  $\phi \in \mathcal{L}_P$  as follows:

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

Let  $rd \in RD_P$ . In the following, we define the notion of  $(P, rd)$ -safe formulas for which the attack preservation holds.

**Definition 13 (( $P, rd$ )-safe formulas).** Let  $P = (\Gamma_P, S_P)$  be a protocol and  $rd$  a redundancy abstraction for  $P$ . A formula  $\phi$  is  $(P, rd)$ -safe if for all  $ev(t) \in Evt_\phi$ , we have  $rd(t) \neq \text{nil}$ .

**Theorem 5 (Soundness for redundancy removal abstractions).** Let  $P = (\Gamma_P, S_P)$  be a protocol,  $\phi \in \mathcal{L}_P$ , and  $rd \in RD_P$  be a  $(P, rd)$ -safe formula. Suppose that  $IK_0 \subseteq IK'_0$ . Then, for all states  $(tr, th, \sigma)$  reachable in  $P$ , we have

1.  $(rd(tr), rd(th), \sigma)$  is reachable in  $rd(P)$  and
2.  $(tr, th, \sigma) \not\models \phi$  implies  $(rd(tr), rd(th), \sigma) \not\models rd(\phi)$ .

*Proof.* By Proposition 7, we have that  $(rd(tr), rd(th), \sigma)$  is reachable in  $rd(P)$ . It remains to show that

$$\forall \vartheta. (tr, th, \sigma, \vartheta) \not\models \phi \Rightarrow (rd(tr), rd(th), \sigma) \not\models rd(\phi).$$

We proceed by induction on the structure of  $\phi$  and consider the following cases.

–  $\phi \equiv m = m'$  or  $\phi \equiv \neg(m = m')$ .

$$\begin{aligned} & (tr, th, \sigma, \vartheta) \models m = m' \\ \Leftrightarrow & m\sigma = m'\sigma \\ \Leftrightarrow & (rd(tr), rd(th), \sigma, \vartheta) \models rd((m = m')) \end{aligned}$$

–  $\phi = secret(m)$ .

$$\begin{aligned} & (tr, th, \sigma, \vartheta) \not\models secret(m) \\ \Leftrightarrow & IK(tr)\sigma, IK_0 \vdash m\sigma \\ \Rightarrow & IK(rd(tr))\sigma, IK_0 \vdash m\sigma && \text{by Lemma 32} \\ \Leftrightarrow & (rd(tr), rd(th), \sigma, \vartheta) \not\models rd(secret(m)) \end{aligned}$$

The remaining cases are routine. This completes the proof of the theorem.  $\square$

## J Variable removal abstractions

Our type-based abstractions do not allow us to remove message variables. This is because our framework heavily relies on the substitution property, which fails to hold for the removal of message variables. In this section, we establish *variable removal abstractions* separately for a subclass of protocols, namely *well-formed protocols*.

**Definition 14.** *A protocol  $P$  is well-formed if all non-agent variables first occur in receive events, i.e., for all events  $e$  in a role  $S_P(R)$  and all variables  $X \in \text{vars}(\text{term}(e))$  such that  $\Gamma_P(X) \neq \alpha$ , there is an event  $\text{rcv}(t)$  in  $S_P(R)$  such that  $\text{rcv}(t)$  equals or precedes  $e$  in  $S_P(R)$  and  $X \in \text{vars}(t)$ .*

We now define *variable removal abstractions*.

**Definition 15.** *Let  $V$  be a set of variables. The variable removal abstraction  $\text{vrem}_V$  is defined such that for all terms  $t$*

- if  $t$  is an atom then  $\text{vrem}_V(t) = t$ ,
- if  $t$  is a variable and  $t \in V^b$  then  $\text{vrem}_V(t) = \text{nil}$ ,
- if  $t$  is variable and  $t \notin V^b$  then  $\text{vrem}_V(t) = t$ ,
- if  $t = g_1(t')$  for  $g_1 \in \Sigma^1$  then  $\text{vrem}_V(t) = \text{nf}(g_1(\text{vrem}_V(t')))$ , and
- if  $t = g_2(u, u')$  for  $g_2 \in \Sigma^2$  then we have that

$$\text{vrem}_V(t) = \text{nf}(g_2(\text{vrem}_V(u), \text{vrem}_V(u'))).$$

It is clear that we cannot arbitrarily remove variables as the intruder might lose information that he could use to derive certain attack messages. Below we identify some conditions under which the set of messages the intruder can derive (and hence any potential attack) is preserved by variable removal. Before stating these assumptions, we introduce some auxiliary definitions.

First, we extend the notion of clearness to a set of terms  $T$  as expected, i.e.,  $T$  is *clear* in a term  $t$  if every term in  $T$  is. We also say that  $T$  is clear in a set of terms  $T'$  if every  $t \in T$  is clear in every  $t' \in T'$ .

**Definition 16 ( $V$ -deducible terms).** *Let  $\Gamma$  be a typing environment,  $T$  a set of ground terms, and  $V$  a set of variables. A term  $t$  is  $V$ -deducible with respect to  $(\Gamma, T, \text{vrem}_V)$  if we have that*

$$T, (V^b \cap \text{vars}(t)), \text{vars}_\alpha^\Gamma(t), \text{vrem}_V(t) \vdash t.$$

Informally, a term  $t$  is  $V$ -deducible if  $t$  is derivable from  $\text{vrem}_V(t)$  together with other available information such as agent names and a given set of terms (such as the intruder's initial knowledge). In the following lemma, we abuse the notation and use  $\text{vars}(tr)$  to denote the set of variables occurring in  $tr$ .

**Lemma 33.** *Let  $P = (\Gamma_P, S_P)$  be a well-formed protocol and  $(tr, th, \sigma)$  be a reachable state of  $P$ . Let  $V \subseteq \mathcal{V}$  be a set of variables such that*

1.  $V$  is clear in all terms  $t$  occurring in receive events in  $P$ ,

2. all terms  $t$  occurring in send events in  $P$  are  $V$ -deducible with respect to  $(\Gamma_P, IK_0, vrem_V)$ .

Then we have that  $vrem_V(IK(tr))\sigma, IK_0 \vdash (IK(tr) \cup (vars(tr) \cap V^b))\sigma$ .

*Proof.* We proceed by induction on  $tr$ . For the base case,  $tr = \epsilon$ , the lemma holds trivially. For the inductive step, suppose  $(tr', th', \sigma)$  is reachable in  $P$  and there is a transition  $(tr', th', \sigma) \rightarrow (tr, th, \sigma)$  such that  $tr = tr' \cdot (i, ev(t))$  for some  $i \in TID$  and some term  $t$ . By induction hypothesis, we have

$$vrem_V(IK(tr'))\sigma, IK_0 \vdash (IK(tr') \cup (vars(tr') \cap V^b))\sigma.$$

and we have to show  $vrem_V(IK(tr))\sigma, IK_0 \vdash (IK(tr) \cup (vars(tr) \cap V^b))\sigma$ . We reason by a case distinction on the rule  $r$  that has been applied in the last step.

– If  $r = RECV$  then we have that  $IK(tr') = IK(tr)$ . Thus by induction hypothesis, we have  $vrem_V(IK(tr))\sigma, IK_0 \vdash IK(tr)\sigma$ . Therefore, it remains to show that  $vrem_V(IK(tr))\sigma, IK_0 \vdash (vars(tr) \cap V^b)\sigma$ .

Note that  $tr = tr' \cdot (i, rcv(t))$ . If  $vars(t) \cap V^b \subseteq vars(tr')$  then  $vars(tr) \cap V^b = vars(tr') \cap V^b$  and the conclusion follows directly from the induction hypothesis. Otherwise, let  $X^i \in (vars(t) \cap V^b) \setminus vars(tr')$ . Given the induction hypothesis, it is sufficient to establish  $vrem_V(IK(tr))\sigma, IK_0 \vdash X^i\sigma$ .

By the premises of the  $RECV$  rule, we know that  $IK(tr')\sigma, IK_0 \vdash t\sigma$ . Since  $V$  is clear in  $t$ , we also have

$$IK(tr')\sigma, IK_0 \vdash X^i\sigma \quad (24)$$

Since all terms in  $IK(tr')$  are  $V$ -deducible with respect to  $(\Gamma_P, IK_0, vrem_V)$ , we have

$$IK_0, (V^b \cap vars(IK(tr'))), vars_{\alpha}^{\Gamma_P}(IK(tr')), vrem_V(IK(tr')) \vdash IK(tr').$$

By instantiating this with  $\sigma$  and using the fact that  $vars(IK(tr')) \subseteq vars(tr')$ , we obtain

$$IK_0, (V^b \cap vars(tr'))\sigma, vars_{\alpha}^{\Gamma_P}(IK(tr'))\sigma, vrem_V(IK(tr'))\sigma \vdash IK(tr')\sigma$$

Since  $\mathcal{A} \subseteq IK_0$  and  $\sigma$  is well-typed with respect to  $\Gamma_P$ , we can derive  $vars_{\alpha}^{\Gamma_P}(IK(tr'))\sigma \subseteq IK_0$ . Using this together with the induction hypothesis and  $IK(tr) = IK(tr')$ , we derive

$$vrem_V(IK(tr))\sigma, IK_0 \vdash IK(tr')\sigma.$$

Combining this with (24), we obtain  $vrem_V(IK(tr))\sigma, IK_0 \vdash X^i\sigma$  as required.

– If  $r = SEND$  then we have  $tr = tr' \cdot (i, snd(t))$ . Thus, we have that  $IK(tr) = IK(tr') \cup \{t\}$ . By the well-formedness of  $P$ , we have  $vars(tr) = vars(tr')$ . Hence, it follows from the induction hypothesis that

$$vrem_V(IK(tr))\sigma, IK_0 \vdash (vars(tr) \cap V^b)\sigma. \quad (25)$$

We are left to show that  $vrem_V(IK(tr))\sigma, IK_0 \vdash IK(tr)\sigma$ . By instantiating the  $V$ -deducibility condition for  $t$  from assumption 2 with  $\sigma$ , we obtain

$$IK_0, (V^b \cap vars(t))\sigma, vars_{\alpha}^{\Gamma_P}(t)\sigma, vrem_V(t)\sigma \vdash t\sigma. \quad (26)$$

Since  $vars(t) \subseteq vars(tr)$ . By (25), we have

$$vrem_V(IK(tr))\sigma, IK_0 \vdash (vars(t) \cap V^b)\sigma. \quad (27)$$

Moreover, we have that  $vars_{\alpha}^{\Gamma_P}(t)\sigma \subseteq IK_0$ , since  $\sigma$  is well-typed with respect to  $\Gamma_P$  and  $\mathcal{A} \subseteq IK_0$ . Together with (27) and (26), we derive that

$$vrem_V(IK(tr))\sigma, IK_0, vrem_V(t)\sigma \vdash t\sigma.$$

Since  $t \in IK(tr)$ , we have that  $vrem_V(t)\sigma \in vrem_V(IK(tr))\sigma$ . Hence, we obtain that

$$vrem_V(IK(tr))\sigma, IK_0 \vdash t\sigma$$

By induction hypothesis, we have  $vrem_V(IK(tr')\sigma), IK_0 \vdash IK(tr')\sigma$ . Hence, we derive that  $vrem_V(IK(tr))\sigma, IK_0 \vdash IK(tr)\sigma$  as required.

This completes the proof of the lemma.  $\square$

**Proposition 8.** *Let  $P = (\Gamma_P, S_P)$  be a well-formed protocol. Suppose  $V$  is a set of variables and  $u$  a term such that*

- (i)  $V$  is clear in  $u$  and in all terms  $t$  occurring in receive events in  $P$ ,
- (ii) all terms  $t$  occurring in send events in  $P$  are  $V$ -deducible with respect to  $(\Gamma_P, IK_0, vrem_V)$ .

*Suppose  $(tr, th, \sigma)$  is a reachable state of  $P$ . Then  $IK(tr)\sigma, IK_0 \vdash u\sigma$  implies  $IK(vrem_V(tr))\sigma, IK_0 \vdash vrem_V(u)\sigma$ .*

*Proof.* We derive

$$\begin{array}{ll} IK(vrem_V(tr))\sigma, IK_0 \vdash vrem_V(IK(tr')\sigma), IK_0 & \text{since } \text{nil} \in IK_0 \\ \vdash IK(tr)\sigma, IK_0 & \text{by Lemma 33} \\ \vdash u\sigma, IK_0 & \text{by assumption} \\ \vdash vrem_V(u)\sigma & \end{array}$$

The last step follows from  $\text{nil} \in IK_0$  and from  $\{u, \text{nil}\} \vdash vrem_V(u)$ , which is a consequence of assumption (i).  $\square$

The following theorem states the reachability preservation result for variable removal abstractions.

**Theorem 6.** *Let  $P = (\Gamma_P, S_P)$  be a well-formed protocol. Suppose  $IK_0 \subseteq IK'_0$  and  $V \subseteq \mathcal{V}$  is a set of variables such that*

1.  $V$  is clear in all terms  $t$  occurring in receive events in  $P$ ,

2. all terms  $t$  occurring in send events in  $P$  are  $V$ -deducible with respect to  $(\Gamma_P, IK_0, vrem_V)$ .

Let  $(tr, th, \sigma)$  be reachable in  $P$ . Then we have that  $(vrem_V(tr), vrem_V(th), \sigma)$  is reachable in  $vrem_V(P)$ .

*Proof.* Since  $\sigma$  is well-typed with respect to  $\Gamma_P$  and  $vrem_V(\Gamma_P) \subseteq \Gamma_P$ , we derive that  $\sigma$  is well-typed with respect to  $vrem_V(\Gamma_P)$ . We prove the reachability of the state  $(vrem_V(tr), vrem_V(th), \sigma)$  by induction on the number  $n$  of transitions leading to a state  $(tr, th, \sigma)$ . The theorem holds trivially for the empty trace ( $n = 0$ ). For the inductive case ( $n = k + 1$ ), assume that  $(tr', th', \sigma)$  is reachable in  $k$  steps and there is a transition  $(tr', th', \sigma) \rightarrow (tr, th, \sigma)$ . By induction hypothesis, the state  $(vrem_V(tr'), vrem_V(th'), \sigma)$  is reachable in  $vrem_V(P)$ .

We distinguish two cases according to the rule  $r$  used to justify the step  $k + 1$  in  $P$ . We first treat the case of the receive rule ( $r = RECV$ ). The rule's premises require that there are  $i \in TID$ ,  $R \in dom(S_P)$ , and a suffix  $tl$  of the role  $inst_i(S_P(R))$  such that

$$th'(i) = (R, rcv(t) \cdot tl) \quad \text{and} \quad IK(tr')\sigma, IK_0 \vdash t\sigma \quad (28)$$

The rule's conclusion implies that  $tr = tr' \cdot (i, rcv(t))$  and  $th = th'[i \mapsto tl]$ . We consider two cases.

- If  $vrem_V(t) = \text{nil}$  then we have that

$$\begin{aligned} vrem_V(tr) &= vrem_V(tr'), \\ vrem_V(th) &= vrem_V(th'). \end{aligned}$$

Hence, we conclude that  $(vrem_V(tr), vrem_V(th), \sigma)$  is reachable in  $vrem_V(P)$  by the induction hypothesis.

- If  $vrem_V(t) \neq \text{nil}$  then we show that  $vrem_V(P)$  has a transition

$$(vrem_V(tr'), vrem_V(th'), \sigma) \rightarrow (vrem_V(tr), vrem_V(th), \sigma).$$

In order to apply the  $RECV$  rule in the state  $(vrem_V(tr'), vrem_V(th'), \sigma)$  the following two premises must be satisfied:

$$\begin{aligned} vrem_V(th')(i) &= (R, rcv(vrem_V(t)) \cdot vrem_V(tl)), \quad \text{and} \\ IK(vrem_V(tr'))\sigma, IK'_0 &\vdash vrem_V(t)\sigma. \end{aligned}$$

The first premise holds by application of  $vrem_V$  to  $th'$ . The second one follows from Proposition 8 and the assumption that  $IK_0 \subseteq IK'_0$ . The successor state in the conclusion of the  $RECV$  rule is

$$(vrem_V(tr') \cdot (i, rcv(vrem_V(t))), vrem_V(th')[i \mapsto (R, vrem_V(tl))], \sigma),$$

which is identical to the state  $(vrem_V(tr), vrem_V(th), \sigma)$ , whose reachability in  $vrem_V(P)$  we have hereby established.

The case of the send rule ( $r = SEND$ ) is similar but simpler, since the deducibility condition falls away. This completes the proof of the theorem.  $\square$

**Definition 17 (( $P, V, vrem_V$ )-safe formulas).** *Let  $P$  be a protocol and let  $V \subseteq \mathcal{V}$  be a set of variables. A formula  $\phi \in \mathcal{L}_P$  is ( $P, vrem_V$ )-safe iff the following holds.*

1. for all  $(m, m') \in Eq_\phi$ , we have  $vars(\{m, m'\}) \cap V^\# = \emptyset$ ,
2. for all terms  $t \in Sec_\phi$ , we have  $vars(t) \cap V^\# = \emptyset$ ,
3. for all events  $e(t) \in Evt_\phi$ , we have  $vrem_V(t) \neq \text{nil}$ , and
4.  $vrem_V(m) = vrem_V(m')$  implies  $m = m'$  for all  $s(m) \in Evt_\phi^+$  and  $s(m') \in Evt(\mathcal{M}_P)$ .

We now state our soundness theorem as follows.

**Theorem 7 (Soundness for variable removal abstractions).** *Let  $P = (\Gamma_P, S_P)$  be a well-formed protocol and  $\phi \in \mathcal{L}_P$  be a ( $P, V, vrem_V$ )-safe formula. Assume that  $V \subseteq \mathcal{V}$  is a set of variables such that*

1.  $V$  is clear in all terms  $t$  occurring in receive events in  $P$ ,
2. all terms  $t$  occurring in send events in  $P$  are  $V$ -deducible with respect to  $(\Gamma_P, IK_0, vrem_V)$ .

*Let  $(tr, th, \sigma)$  be reachable in  $P$  and suppose  $IK_0 \subseteq IK'_0$ . Then we have that  $(vrem_V(tr), vrem_V(th), \sigma)$  is a reachable state in  $vrem_V(P)$ . Moreover, if  $(tr, th, \sigma) \not\models \phi$  then  $(vrem_V(tr), vrem_V(th), \sigma) \not\models \phi$ .*

*Proof.* By Theorem 6, we have that  $(vrem_V(tr), vrem_V(th), \sigma)$  is a reachable state in  $vrem_V(P)$ . It remains to show that

$$\forall \vartheta. (tr, th, \sigma, \vartheta) \not\models \phi \Rightarrow (vrem_V(tr), vrem_V(th), \sigma) \not\models vrem_V(\phi).$$

We proceed by induction on the structure of  $\phi$  and consider the following non-trivial cases.

- $\phi \equiv m = m'$  or  $\phi \equiv \neg(m = m')$ .

$$\begin{aligned} & (tr, th, \sigma, \vartheta) \models m = m' \\ \Leftrightarrow & m\sigma = m'\sigma \\ \Leftrightarrow & vrem_V(m)\sigma = vrem_V(m')\sigma \quad (\text{since } \phi \text{ is } (P, V, vrem_V)\text{-safe}) \\ \Leftrightarrow & (vrem_V(tr), vrem_V(th), \sigma, \vartheta) \models vrem_V(m) = vrem_V(m') \end{aligned}$$

- $\phi = \text{secret}(m)$ .

$$\begin{aligned} & (tr, th, \sigma, \vartheta) \not\models \text{secret}(m) \\ \Leftrightarrow & IK(tr)\sigma, IK_0 \vdash m\sigma \\ \Rightarrow & vrem_V(IK(tr))\sigma, IK_0 \vdash m\sigma && \text{by Proposition 8} \\ \Leftrightarrow & vrem_V(IK(tr))\sigma, IK_0 \vdash vrem_V(m)\sigma && \text{since } vrem_V(m) = m \\ \Rightarrow & IK(vrem_V(tr))\sigma, IK'_0 \vdash vrem_V(m)\sigma && \text{since } \text{nil} \in IK_0 \subseteq IK'_0 \\ \Leftrightarrow & (vrem_V(tr), vrem_V(th), \sigma, \vartheta) \not\models \text{secret}(vrem_V(m)) \end{aligned}$$

This completes the proof of the theorem.  $\square$

## K Atom removal abstraction

In this section, we introduce a new class of abstractions that allows to remove atoms in clear.

**Definition 18.** Let  $At \subseteq \mathcal{A} \cup \mathcal{C} \cup \mathcal{F}$  be a set of atoms. The atom removal abstraction  $arem_{At}$  is defined as follows:

- $arem_{At}(u) = \text{nil}$  if  $u \in At^b$ ,
- $arem_{At}(\langle t_1, t_2 \rangle) = \text{nf}(\langle arem_{At}(t_1), arem_{At}(t_2) \rangle)$ , and
- $arem_{At}(t) = t$  for all other terms.

Before we prove deducibility preservation for our atom removal abstractions, we introduce some auxiliary definitions. For any term  $t$ , we use  $atoms(t)$  and  $fresh(t)$  to denote the set of atoms and fresh values occurring in  $t$ , respectively. Recall that the attacker has a countably infinite choice of nonces  $n_i^\bullet \in \mathcal{F}^\bullet$  for each type  $\beta_n$ . Given a set of atoms  $At \subseteq \mathcal{A} \cup \mathcal{C} \cup \mathcal{F}$  and a set of terms  $T$ , we define  $imap(At, T)$  as the set of injective maps  $\rho : tID(At, T) \rightarrow \mathbb{N}$  where  $tID(At, T) = \{i \in TID \mid f^i \in fresh(At^\sharp) \cap split(T)\}$ . We extend  $\rho$  to atoms and variables as follows.

- $\rho(n^i) = n_{\rho(i)}^\bullet$  for all  $n^i \in fresh(At^\sharp) \cap split(T)$ ,
- $\rho(u) = u$  for all remaining atomic messages  $u \in \mathcal{A} \cup \mathcal{C} \cup \mathcal{V}^b \cup \mathcal{F}^b \cup \mathcal{F}^\bullet$ .

We extend  $\rho$  homomorphically to all terms. For the sake of uniformity, we treat  $\rho$  as a substitution below.

**Lemma 34.** Let  $T$  be set of normal-form terms,  $t$  a normal-form term,  $At$  a set of atoms such that

- (i)  $At^\sharp$  is clear in  $T \cup \{t\}$ , and
- (ii)  $At^\sharp \cap fresh(IK_0) = \emptyset$ .

Let  $\sigma$  be a substitution such that  $T\sigma, IK_0 \vdash t\sigma$ . Then, for all  $\rho \in imap(At, T \cup \{t\})$ , we have

$$arem_{At}(T)(\rho \circ \sigma), IK_0 \vdash arem_{At}(t)(\rho \circ \sigma).$$

*Proof.* We start by observing that under assumption (i) we have

$$arem_{At}(u), \mathcal{F}^\bullet \vdash u\rho \quad \text{and} \quad u\rho, \text{nil} \vdash arem_{At}(u) \tag{29}$$

for all terms  $u \in T \cup \{t\}$  and  $\rho \in imap(At, T \cup \{t\})$ .

Suppose  $T\sigma, IK_0 \vdash t\sigma$ . We show that  $arem_{At}(T)\theta, IK_0 \vdash arem_{At}(t)\theta$  where  $\theta = \rho \circ \sigma$ . Since  $\mathcal{F}^\bullet\theta = \mathcal{F}^\bullet$  and  $\mathcal{F}^\bullet \subseteq IK_0$ , we can use Lemma 23 to derive the following from (29):

$$arem_{At}(T)\theta, IK_0 \vdash (T\rho)\theta \quad \text{and} \quad (t\rho)\theta, \text{nil} \vdash arem_{At}(t)\theta \tag{30}$$

Since  $IK_0\rho = IK_0$  by assumption (ii), we can also apply Lemma 23 to the assumption  $T\sigma, IK_0 \vdash t\sigma$  and deduce  $(T\sigma)\rho, IK_0 \vdash (t\sigma)\rho$ . Next, we use Lemma 24 to deduce  $(T\rho)\theta, IK_0 \vdash (t\rho)\theta$ . Combining this with (30) above yields the desired result  $arem_{At}(T)\theta, IK_0 \vdash arem_{At}(t)\theta$ .  $\square$

**Theorem 8 (Reachability preservation for atom removal abstractions).**

Let  $P$  be a protocol and  $At \subseteq \text{atoms}(\mathcal{M}_P)$  a set of atoms such that

- (i)  $At$  is clear in  $\mathcal{M}_P$ , and
- (ii)  $At^\# \cap \text{fresh}(IK_0) = \emptyset$ .

Let  $(tr, th, \sigma)$  be a reachable state of  $P$  and  $\rho \in \text{imap}(At, \text{term}(tr))$ . Then the state  $(\text{arem}_{At}(tr), \text{arem}_{At}(th), \rho \circ \sigma)$  is a reachable state of  $\text{arem}_{At}(P)$ .

*Proof.* Let  $\theta = \rho \circ \sigma$ . It is clear that  $\theta$  is well-typed with respect to  $\Gamma_P = \Gamma_{\text{arem}_{At}(P)}$ . We now prove the first conclusion by induction on the number  $n$  of transitions leading to a state  $(tr, th, \sigma)$ . For the empty trace ( $n = 0$ ), the theorem holds trivially. For the inductive case ( $n = k + 1$ ), assume that  $(tr', th', \sigma)$  is reachable in  $k$  steps and there is a transition  $(tr', th', \sigma) \rightarrow (tr, th, \sigma)$ . By induction hypothesis,  $(\text{arem}_{At}(tr'), \text{arem}_{At}(th'), \theta)$  is reachable in  $\text{arem}_{At}(P)$ .

There are two cases according to the rule  $r$  that has been applied in the step  $k + 1$  in  $P$ . The case of the send rule is easy. We consider the case of receive rule. The rule's premises require that there are  $i \in TID$ ,  $R \in \text{dom}(S_P)$ , and a suffix  $tl$  of the role  $\text{inst}_i(S_P(R))$  such that

$$th'(i) = (R, \text{rcv}(t) \cdot tl) \quad \text{and} \quad IK(tr')\sigma, IK_0 \vdash t\sigma \quad (31)$$

The rule's conclusion implies that  $tr = tr' \cdot (i, \text{rcv}(t))$  and  $th = th'[i \mapsto tl]$ . In order to apply the  $RECV$  rule in the state  $(\text{arem}_{At}(tr'), \text{arem}_{At}(th'), \theta)$ , we must show the following two premises

$$\begin{aligned} \text{arem}_{At}(th')(i) &= (R, \text{rcv}(\text{arem}_{At}(t)) \cdot \text{arem}_{At}(tl)), \text{ and} \\ IK(\text{arem}_{At}(tr'))\theta, IK'_0 &\vdash \text{arem}_{At}(t)\theta. \end{aligned}$$

Clearly, the first premise is satisfied by application of  $\text{arem}_{At}$  to  $th'$ . We now show the second one. Using Lemma 34, we deduce from (31) that

$$\text{arem}_{At}(IK(tr'))\theta, IK_0 \vdash \text{arem}_{At}(t)\theta$$

Since  $\text{arem}_{At}(IK(tr')) \cup \{\text{nil}\} = IK(\text{arem}_{At}(tr')) \cup \{\text{nil}\}$  and  $\text{nil} \in IK_0$ , we obtain

$$IK(\text{arem}_{At}(tr'))\theta, IK_0 \vdash \text{arem}_{At}(t)\theta.$$

Moreover, the successor state in the conclusion of the  $RECV$  rule is

$$(\text{arem}_{At}(tr') \cdot (i, \text{rcv}(\text{arem}_{At}(t))), \text{arem}_{At}(th')[i \mapsto (R, \text{arem}_{At}(tl))], \sigma),$$

which is identical to the state  $(\text{arem}_{At}(tr), \text{arem}_{At}(th), \sigma)$ . This concludes the proof of the theorem.  $\square$

**Definition 19 (( $P, \text{arem}_{At}$ )-safe formulas).** Let  $P$  be a protocol. We assume a set of atoms  $At \subseteq \text{atoms}(\mathcal{M}_P)$ . A formula  $\phi \in \mathcal{L}_P$  is  $(P, \text{arem}_{At})$ -safe if the following conditions holds:

- (i)  $At^\# \cap \text{fresh}(\text{Sec}_\phi \cup \text{EqTerm}_\phi) = \emptyset$ ,

- (ii) for all  $e(t) \in \text{Evt}_\phi$ , we have  $\text{arem}_{At}(t) \neq \text{nil}$ , and
- (iii)  $\text{arem}_{At}(m) = \text{arem}_{At}(m')$  implies  $m = m'$  for all  $s(m) \in \text{Evt}_\phi^+$  and  $s(m') \in \text{Evt}(\mathcal{M}_P)$ .

We now show soundness for atom removal abstractions.

**Theorem 9 (Soundness for atom removal abstractions).** *Let  $P$  be a protocol,  $At \subseteq \text{atoms}(\mathcal{M}_P)$  and  $\phi \in \mathcal{L}_P$  a formula such that*

- (i)  $At$  is clear in  $\mathcal{M}_P$ ,
- (ii)  $At^\# \cap \text{fresh}(IK_0) = \emptyset$ , and
- (iii)  $\phi$  is  $(P, \text{arem}_{At})$ -safe

Then, for all reachable states  $(tr, th, \sigma)$  of  $P$ , there is a nil-free ground substitution  $\theta$  such that

1. the state  $(\text{arem}_{At}(tr), \text{arem}_{At}(th), \theta)$  is reachable in  $\text{arem}_{At}(P)$ , and
2.  $(tr, th, \sigma) \not\models \phi$  implies  $(\text{arem}_{At}(tr), \text{arem}_{At}(th), \theta) \not\models \text{arem}_{At}(\phi)$ .

*Proof.* Let  $\rho \in \text{imap}(At, \text{term}(tr))$  such that

$$\rho \text{ is injective on } \text{fresh}(\text{ran}(\sigma)). \quad (32)$$

By Lemma 30, we can, without loss of generality, assume that  $\sigma$  is nil-free. Let  $\theta = \rho \circ \sigma$ . Then  $\theta$  is nil-free. By Theorem 8, we have that  $(\text{arem}_{At}(tr), \text{arem}_{At}(th), \theta)$  is reachable in  $\text{arem}_{At}(P)$  (and thus  $\theta$  is well-typed with respect to  $\Gamma_{\text{arem}_{At}(P)}$ ). Hence, it remains to show that

$$\forall \vartheta. (tr, th, \sigma, \vartheta) \not\models \phi \Rightarrow (\text{arem}_{At}(tr), \text{arem}_{At}(th), \theta) \not\models \text{arem}_{At}(\phi).$$

We proceed by induction on the structure of  $\phi$  and consider the following non-trivial cases.

$$- \phi \equiv m = m' \text{ or } \phi \equiv \neg(m = m').$$

$$\begin{aligned} & (tr, th, \sigma, \vartheta) \models m = m' \\ \Leftrightarrow & m\sigma = m'\sigma \\ \Leftrightarrow & (m\sigma)\rho = (m'\sigma)\rho && \text{(by (32) and (iii))} \\ \Leftrightarrow & (m\rho)\theta = (m'\rho)\theta && \text{(by Lemma 24)} \\ \Leftrightarrow & \text{arem}_{At}(m)\theta = \text{arem}_{At}(m')\theta && \text{(by Def 19(ii))} \\ \Leftrightarrow & (\text{arem}_{At}(tr), \text{arem}_{At}(th), \theta, \vartheta) \models \text{arem}_{At}(m) = \text{arem}_{At}(m') \end{aligned}$$

$$- \phi = \text{secret}(m).$$

$$\begin{aligned} & (tr, th, \sigma, \vartheta) \not\models \text{secret}(m) \\ \Leftrightarrow & IK(tr)\sigma, IK_0 \vdash m\sigma \\ \Rightarrow & \text{arem}_{At}(IK(tr))\theta, IK_0 \vdash \text{arem}_{At}(m)\theta && \text{(by Lemma 34, Def 19(i))} \\ \Rightarrow & IK(\text{arem}_{At}(tr))\theta, IK_0 \vdash \text{arem}_{At}(m)\theta && \text{(since nil} \in IK_0) \\ \Leftrightarrow & (\text{arem}_{At}(tr), \text{arem}_{At}(th), \theta, \vartheta) \not\models \text{secret}(\text{arem}_{At}(m)) \end{aligned}$$

This completes the proof of the theorem. □

## L Experimental results

### L.1 Scyther tool

The Scyther tool is based on symbolic backwards search and supports verification of both a bounded and an unbounded number of threads. We have demonstrated our abstraction method on a variety of protocols, mostly from the IKE and ISO/IEC 9798 families. Our results with the Scyther tool (version 1.1.2) are summarized in Table 2. Our experiments show substantial performance gains. The abstractions enable Scyther to verify seven protocols for an unbounded number of threads within few seconds whereas it fails on the original protocols.

All but one protocol in the ISO/IEC 9798 family are verified under 0.3 seconds. The outlier is the ISO/IEC 9798-2-6 whose verification takes about 37 minutes. We attribute this to the unencrypted ticket in the abstracted protocol which provides the intruder more possibilities to manipulate the ticket variable. This also explains the big differences in running time between the first abstraction level (where we still have clear message variables) and the second level (where the message variables are removed). However, for the ISO/IEC 9798-2-6 protocol, we can not remove the ticket as it is essential for the responder to obtain the shared key.

For the IKE protocols, we approximate the Diffie-Hellman equations in Scyther using oracle roles. This complicates the verification task and, as a consequence, the average performance gain appears to be smaller than with the ISO/IEC protocols or the PANA-AKA protocol. In particular, the unbounded verification of (abstractions of) the first six IKE protocols in Table 2 still results in a timeout. However, we are able to either significantly improve the bounds on the number of threads that can be covered for these protocols or, for IKEv2-eap and IKEv2-eap2, enable protocol verification for a bounded number of threads where it timed out before even for three threads.

Scyther verifies the abstracted PANA-AKA protocol in only 0.23 seconds whereas the verification of the original protocol takes 96 minutes for three threads and times out for four.

Apart from the dramatic speedups we achieve in most cases, we also observe that for many protocols the verification time increases much slower than their originals. For the last five protocols in the table, this time is almost constant whereas it can be exponential, e.g., for ISO/IEC 9798-3-7-1 and PANA-AKA. Moreover, our abstractions greatly reduce memory consumption. In particular, Scyther runs out of memory for ISO/IEC 9798-3-6-1 and ISO/IEC 9798-3-7-1 after 5 and 6 threads, respectively. However, abstraction enables it to run up to an unbounded number of threads.

protocol/prop./#threads	S	A	W	N	3	4	5	6	7	8	$\infty$
IKEv1-pk2-a2	•			•	44.09	310.45	2021.13	12484.88	TO	TO	TO
IKEv1-pk2-a2-abs1	•			•	3.83	16.11	89.30	593.11	3666.50	25441.00	TO
IKEv1-pk2-a	•			•	1036.33	TO	TO	TO	TO	TO	TO
IKEv1-pk2-a-abs1	•			•	82.94	2108.91	TO	TO	TO	TO	TO
IKEv1-pk-a2	•			•	13.00	50.69	118.21	230.58	421.06	705.26	TO
IKEv1-pk-a2-abs1	•			•	0.51	0.80	1.40	2.22	3.67	5.72	TO
IKEv1-pk-a22	•			•	15.90	83.36	250.33	550.72	1003.73	1731.87	TO
IKEv1-pk-a22-abs1	•			•	0.56	0.98	1.39	2.27	3.66	5.75	TO
IKEv2-eap	•			•	TO	TO	TO	TO	TO	TO	TO
IKEv2-eap-abs2	•			•	22.47	268.86	1894.31	8003.61	25065.15	TO	TO
IKEv2-eap2	•			•	TO	TO	TO	TO	TO	TO	TO
IKEv2-eap2-abs2	•			•	226.13	10568.54	TO	TO	TO	TO	TO
IKEv2-mac	•			•	1.85	4.91	6.72	8.07	8.42	8.49	8.70
IKEv2-mac-abs1	•			•	0.50	0.71	0.76	0.78	0.81	0.83	0.85
IKEv2-mac2	•			•	1.87	4.39	6.54	7.63	8.20	8.46	8.69
IKEv2-mac2-abs1	•			•	0.44	0.72	0.77	0.83	0.79	0.76	0.86
IKEv2-mactosig	•			•	12.17	141.37	1075.46	7440.81	TO	TO	TO
IKEv2-mactosig-abs1	•			•	1.70	5.08	11.33	13.14	18.93	19.23	19.81
IKEv2-mactosig2	•			•	11.55	133.74	1052.96	7112.24	TO	TO	TO
IKEv2-mactosig2-abs1	•			•	1.59	5.52	9.23	13.05	18.74	17.57	19.31
IKEv2-sigtomac	•			•	6.22	26.46	66.05	115.9	172.09	212.43	238.43
IKEv2-sigtomac-abs1	•			•	6.20	23.20	49.02	68.68	77.94	80.44	77.83
ISOIEC 9798-2-5	•				0.84	10.67	75.64	582.03	4409.73	TO	TO
ISOIEC 9798-2-5-abs1	•				0.26	1.60	13.25	103.57	781.61	7694.41	MO
ISOIEC 9798-2-5-abs2	•				0.08	0.15	0.23	0.31	0.28	0.28	0.26
ISOIEC 9798-2-6	•				0.60	3.83	18.44	83.85	198.93	488.29	21478.13
ISOIEC 9798-2-6-abs1	•				0.28	1.63	7.19	26.69	66.15	161.63	2225.92
ISOIEC 9798-3-6-1		•	•	•	42.64	803.29	9007.68	MO	MO	MO	MO
ISOIEC 9798-3-6-1-abs1		•	•	•	2.89	16.4	51.82	121.3	234.69	400.78	603.22
ISOIEC 9798-3-6-1-abs2		•	•	•	0.08	0.12	0.15	0.14	0.14	0.13	0.13
ISOIEC 9798-3-6-2		•	•	•	3.51	47.09	53.09	46.08	48.05	59.74	69.07
ISOIEC 9798-3-6-2-abs1		•	•	•	0.77	2.59	6.83	15.41	19.05	23.08	26.18
ISOIEC 9798-3-6-2-abs2		•	•	•	0.11	0.11	0.10	0.10	0.11	0.11	0.11
ISOIEC 9798-3-7-1		•	•	•	40.19	742.82	7547.40	15893.67	MO	MO	MO
ISOIEC 9798-3-7-1-abs1		•	•	•	2.58	13.48	35.78	80.92	99.73	94.1	92.14
ISOIEC 9798-3-7-1-abs2		•	•	•	0.07	0.10	0.10	0.10	0.10	0.10	0.10
ISOIEC 9798-3-7-2		•	•	•	2.41	7.82	17.30	26.21	37.40	50.83	TO
ISOIEC 9798-3-7-2-abs1		•	•	•	0.89	3.27	9.80	15.31	21.62	34.09	TO
ISOIEC 9798-3-7-2-abs2		•	•	•	0.05	0.05	0.05	0.05	0.05	0.05	0.05
PANA-AKA	•	•	•	•	5755.68	TO	TO	TO	TO	TO	TO
PANA-AKA-abs1	•	•	•	•	0.24	0.24	0.24	0.25	0.24	0.23	0.23

**Table 2.** Experimental verification results for Scyther. The time is in seconds.

## L.2 Avantssar tools

The AVANTSSAR platform is an integrated toolset for the formal specification and **A**utomated **V**Alidation**N** of **T**rust and **S**ecurity of **S**ervice-oriented **A**Rchitectures. It provides three validation back-ends (CL-Atse, OFMC, and SATMC) which share the input languages for specifying protocols. The validators are based on two different techniques. SATMC reduces protocol insecurity problems to the satisfiability of propositional formulas which can then be checked by modern SAT solvers. CL-Atse and OFMC both use constraint solving techniques to search for attacks. However, they use different optimization strategies to reduce the search space. All these tools can verify protocols only for a bounded number of threads.

We have experimented with CL-Atse (version 2.5-21), OFMC (version 2013b), and SATMC (version 3.4) on several protocols from IKE and ISO/IEC 9798 families. Moreover, we have performed experiments on variants of the TLS and basic Kerberos protocols. For TLS, we distinguish two instances according to different security properties of interest. So far, we have not modelled IKE protocols for SATMC, as this requires substantial effort to encode oracles for Diffie-Hellman equations. We therefore defer extended experiments with SATMC to future work.

In our experiments, we measure the verification time for different numbers of sessions. Note that a session in CL-Atse, SATMC, and OFMC differs from a thread in Scyther. CL-Atse and SATMC specify a session as an instantiation of all protocol roles, not just a single role. For instance, a session of a protocol with three different roles results in three role instances (or three threads in Scyther) where a concrete agent is assigned to each role. In contrast, OFMC works with symbolic sessions where the agents executing the roles are not concretely specified but kept as variables.

For the AVANTSSAR tools, our experimental results generally exhibit smaller speedups than for Scyther. There is also a considerable variance between the different tools.

**CL-Atse** (Table 3) CL-Atse shows minor performance gains for the two IKEv1 protocols (pk2-a and pk-a2). However, abstraction enables the verifications of first three IKEv2 protocols (eap, eap2, and mac) for four sessions in less than 2 hours and dramatically speeds up the verification of three sessions of the eap and eap2 variants by factors greater than 690 and 900, respectively. For the last two IKEv2 protocols, the performance gains are still substantial: for four sessions we achieve a speedup factor of 7 for IKEv2-mactosig and of 107 for IKEv2-sigtomac. The best result in the ISO/IEC family is achieved for the ISO/IEC 9798-2-5 protocol where we can turn a timeout for 10 sessions into a time less than 0.2 seconds. The speedup for the 2-5 variant is less impressive and for the two 3-7 variants, we even observe an increase in verification time as we do for the basic Kerberos protocol. For TLS, the verification time of secrecy up to five sessions drops from 260 minutes to 6 minutes (factor 42), whereas that of authentication is sped up by a factor of 1.5 for four sessions.

**OFMC** (Table 4) Surprisingly, the experimental results for OFMC are almost dual to those for CL-Atse. In particular, for the two IKEv1 protocols, OFMC

loses performance on the abstracted protocols compared to the originals. Nevertheless, the abstractions save a lot of effort for the remaining protocols. We are able to increase the number of tractable sessions for 8 protocols: for 2 out of 7 from the IKE family, 5 out of 6 from the ISO/IEC 9798 family, and for the basic Kerberos protocol. For TLS, the verification of authentication is 1.7 times faster (up to 3 sessions). For secrecy, the tool achieves a 20-fold speedup (up to 4 sessions). As a typical case, OFMC verifies an abstraction of ISO/IEC 9798-2-5 for 5 sessions within less than 4 seconds whereas it times out on the original for more than 2 sessions.

**SATMC** (Table 5) The abstractions enable the verification of the Kerberos and TLS protocols for 5 and even 10 sessions. In particular, the tool takes less than 21 seconds to verify the abstracted TLS protocol for 10 sessions whereas it times out for 5 sessions of the original protocol. On the negative side, SATMC loses performance for the protocols in the ISO/IEC family.

Apart from positive results, our experiments also provide an evidence that protocol abstractions are not always helpful. This is typically the case when an abstraction removes sensitive information. In particular, the performance degradation for the AVANTSSAR tools can possibly be attributed to an interference with the highly refined optimization techniques used in these tools. More precisely, an abstraction may get rid of data that is crucial to eliminate redundancies (for CL-Atse) or to limit the number of branching nodes in the symbolic search tree (for OFMC). As a result, the search space becomes larger in the abstracted protocols than in the originals. However, the influence of abstraction on the SATMC's performance is not clear. A further investigation is therefore desirable.

protocol/prop./#sessions	S	A	W	N	3	4	5	10
IKEv1-pk2-a	•			•	0.06	0.11	0.53	TO
IKEv1-pk2-a-abs1	•			•	0.05	0.09	0.32	TO
IKEv1-pk-a2	•			•	0.05	0.09	1.79	TO
IKEv1-pk-a2-abs1	•			•	0.05	0.07	1.17	MO
IKEv2-eap	•			•	649.10	TO	TO	TO
IKEv2-eap-abs2	•			•	0.94	1150.97	TO	TO
IKEv2-eap2	•			•	1744.99	TO	TO	TO
IKEv2-eap2-abs2	•			•	1.92	2468.91	TO	TO
IKEv2-mac	•			•	2.78	TO	TO	TO
IKEv2-mac-abs1	•			•	0.89	6292.36	TO	TO
IKEv2-mactosig	•			•	0.24	1056.31	TO	TO
IKEv2-mactosig-abs1	•			•	0.10	155.63	TO	TO
IKEv2-sigtomac	•			•	2.52	16710.31	TO	TO
IKEv2-sigtomac-abs1	•			•	0.10	155.63	TO	TO
ISOIEC 9798-2-5	•				18.68	TO	TO	TO
ISOIEC 9798-2-5-abs2	•				0.17	0.17	0.18	0.17
ISOIEC 9798-2-6	•				1639.32	TO	TO	TO
ISOIEC 9798-2-6-abs1	•				703.55	TO	TO	TO
ISOIEC 9798-3-7-1		•		•	1.21	4495.43	TO	TO
ISOIEC 9798-3-7-1-abs2		•		•	1973.78	TO	TO	TO
ISOIEC 9798-3-7-2		•		•	29.95	TO	TO	TO
ISOIEC 9798-3-7-2-abs2		•		•	TO	TO	TO	TO
Kerb-basic			•		0.30	0.29	22473.21	TO
Kerb-basic-abs1			•		0.18	0.18	TO	TO
TLS-auth				•	0.10	60.02	TO	TO
TLS-auth-abs2				•	0.08	39.42	TO	TO
TLS-sec				•	0.07	8.63	15551.63	TO
TLS-sec-abs2				•	0.05	0.51	369.57	TO

**Table 3.** Experimental verification results for CL-Atse. The time is in seconds.

protocol/prop./#sessions	$S$	$A$	$W$	$N$	2	3	4	5
IKEv1-pk2-a	•			•	36.28	27745.29	TO	TO
IKEv1-pk2-a-abs1	•			•	59.10	TO	TO	TO
IKEv1-pk-a2	•			•	4.28	849.46	TO	TO
IKEv1-pk-a2-abs1	•			•	12.09	9192.14	TO	TO
IKEv2-eap	•			•	8920.57	TO	TO	TO
IKEv2-eap-abs2	•			•	10.07	8942.94	TO	TO
IKEv2-eap2	•			•	5407.00	TO	TO	TO
IKEv2-eap2-abs2	•			•	46.14	TO	TO	TO
IKEv2-mac	•			•	18.59	22547.87	TO	TO
IKEv2-mac-abs1	•			•	11.19	16139.98	TO	TO
IKEv2-mactosig	•			•	22.08	15561.69	TO	TO
IKEv2-mactosig-abs1	•			•	9.27	10605.58	11782.39	TO
IKEv2-sigtomac	•			•	18.58	13617.91	TO	TO
IKEv2-sigtomac-abs1	•			•	12.36	12408.54	TO	TO
ISOIEC 9798-2-5	•				805.64	TO	TO	TO
ISOIEC 9798-2-5-abs2	•				3.61	3.43	3.85	3.59
ISOIEC 9798-2-6	•				7232.17	TO	TO	TO
ISOIEC 9798-2-6-abs1	•				144.06	TO	TO	TO
ISOIEC 9798-3-6-1		•		•	17941.80	TO	TO	TO
ISOIEC 9798-3-6-1-abs2		•		•	27.92	18019.32	TO	TO
ISOIEC 9798-3-6-2		•		•	TO	TO	TO	TO
ISOIEC 9798-3-6-2-abs2		•		•	12.97	3673.20	TO	TO
ISOIEC 9798-3-7-1		•		•	TO	TO	TO	TO
ISOIEC 9798-3-7-1-abs2		•		•	50.52	TO	TO	TO
ISOIEC 9798-3-7-2		•		•	TO	TO	TO	TO
ISOIEC 9798-3-7-2-abs2		•		•	11.61	4010.64	TO	TO
Kerb-basic			•		20.63	TO	TO	TO
Kerb-basic-abs1			•		8.07	28699.72	TO	TO
TLS-auth				•	9.12	6002.38	TO	TO
TLS-auth-abs1				•	8.88	3549.25	TO	TO
TLS-sec				•	0.27	13.62	1304.21	TO
TLS-sec-abs2				•	0.15	1.97	59.87	TO

**Table 4.** Experimental verification results for OFMC. The time is in seconds.

number of sessions	$S$	$A$	$W$	$N$	3	4	5	10
ISOIEC 9798-2-5	•				0.44	0.42	0.45	0.50
ISOIEC 9798-2-5-abs2	•				0.58	0.64	0.90	3.70
ISOIEC 9798-2-6	•				0.45	0.46	0.48	0.50
ISOIEC 9798-2-6-abs1	•				35.36	247.67	2155.28	23740.06
ISOIEC 9798-3-7-1		•		•	0.46	0.47	0.48	0.53
ISOIEC 9798-3-7-1-abs2		•		•	0.78	0.95	1.31	8.17
ISOIEC 9798-3-7-2		•		•	0.47	0.47	0.64	0.60
ISOIEC 9798-3-7-2-abs2		•		•	2.64	5.83	11.61	121.17
Kerb-basic			•		100.88	107.66	MO	TO
Kerb-basic-abs1			•		3.32	3.46	51.15	23396.15
TLS-auth				•	163.51	4464.73	TO	TO
TLS-auth-abs2				•	1.52	1.90	2.65	20.74
TLS-sec				•	148.21	4002.34	TO	TO
TLS-sec-abs2				•	1.71	1.90	2.30	8.85

**Table 5.** Experimental verification results for SATMC. The time is in seconds.